



_FNV1A_Hash_Jesteress PROC

```
...  
$LL6@FNV1A_Hash@8:  
    mov     edi, DWORD PTR [eax]  
    rol     edi, 5  
    xor     edi, DWORD PTR [eax+4]  
    sub     edx, 8  
    xor     ecx, edi  
    imul    ecx, 709607  
    add     eax, 8  
    dec     esi  
    jne     SHORT $LL6@FNV1A_Hash@8  
    pop     edi  
$LN4@FNV1A_Hash@8:  
    test    dl, 4  
    je      SHORT $LN3@FNV1A_Hash@8  
    xor     ecx, DWORD PTR [eax]  
    imul    ecx, 709607  
    add     eax, 4  
$LN3@FNV1A_Hash@8:  
    test    dl, 2  
    je      SHORT $LN2@FNV1A_Hash@8  
    movzx   esi, WORD PTR [eax]  
    xor     ecx, esi  
    imul    ecx, 709607  
    add     eax, 2  
$LN2@FNV1A_Hash@8:  
    pop     esi  
    test    dl, 1  
    je      SHORT $LN1@FNV1A_Hash@8  
    movsx   eax, BYTE PTR [eax]  
    xor     ecx, eax  
    imul    ecx, 709607  
$LN1@FNV1A_Hash@8:  
...  
_FNV1A_Hash_Jesteress ENDP
```

LEPRECHAUN

AN ENGLISH-WORDLIST RIPPER, REVISION 14-

Free download at www.sanmayce.com — on Intel Merom-1M 2166 MHz it rips **wikipedia** at 5,235,758 words per second.

```

0001 /*
0002 This is source of Leprechaun revision 14_minus quadruplet_r1, copyleft Sanmayce, 2011-Jun-01. 2011-Mar-07: Fixed a small command line
    parsing bug.
0003
0004 Comment/Uncomment accordingly in order to compile:
0005 // #define _WIN32_ENVIRONMENT_
0006 #define _POSIX_ENVIRONMENT_
0007
0008 Linux compile(uncomment #include <io.h> line, ignore warnings):
0009 gcc -D_FILE_OFFSET_BITS=64 -m32 -static -O3 -mtune=generic Leprechaun.c -o Leprechaun_r13_7pluses_generic_32bits.elf
0010
0011 windows compile(uncomment #include <io.h> line, ignore warnings):
0012 For Microsoft (R) 32-bit C/C++ Optimizing Compiler Version 13.10.3077 for 80x86 use:
0013 cl /Ox /Wp64 /TcLeprechaun.c /FaLeprechaun
0014
0015 windows compile(comment #include <io.h> line, ignore warnings):
0016 For Intel(R) C++ Compiler Professional for applications running on IA-32, Version 11.1 use:
0017 icl /Ox /Wp64 /TcLeprechaun.c /FaLeprechaun /w /QxHOST
0018
0019 [It's a little weird(Intel boosts the sort while falls behind in parsing, tested on T3400):]
0020
0021 Leprechaun_r13_7pluses_Microsoft_32-bit_16.00.30319.01.exe_vs_ Wikipedia_22,202,980_LATIN-words:
0022 words per second performance: 1,679,585w/s
0023 Time for making unsorted wordlist: 30 second(s)
0024 Time for sorting unsorted wordlist: 25 second(s)
0025
0026 Leprechaun_r13_7pluses_Intel_IA-32_11.1.exe_vs_ Wikipedia_22,202,980_LATIN-words:
0027 words per second performance: 1,603,240w/s
0028 Time for making unsorted wordlist: 31 second(s)
0029 Time for sorting unsorted wordlist: 19 second(s)
0030
0031 Due to my ignorance(calderas in my C knowledge): 64bit code cannot be generated, for now.
0032 Any improvement is welcome.
0033 Enjoy!
0034 */
0035
0036 // C:\workTemp\Leprechaun_r13++\Visual C++ Toolkit 2003\Leprechaun_r13+++++_C_EXE>cl /Ox /Wp64 /TcLeprechaun.c /FaLeprechaun
0037 // Microsoft (R) 32-bit C/C++ Optimizing Compiler Version 13.10.3077 for 80x86
0038 // Copyright (C) Microsoft Corporation 1984-2002. All rights reserved.
0039 //
0040 // Leprechaun.c
0041 // Leprechaun.c(829) : warning C4312: 'type cast' : conversion from 'int' to 'string' of greater size
0042 // Leprechaun.c(849) : warning C4312: 'type cast' : conversion from 'int' to 'string *' of greater size
0043 // Leprechaun.c(2048) : warning C4312: 'type cast' : conversion from 'int' to 'char *' of greater size
0044 // Leprechaun.c(2063) : warning C4311: 'type cast' : pointer truncation from 'char *' to 'unsigned long'
0045 // Leprechaun.c(2068) : warning C4311: 'type cast' : pointer truncation from 'char *' to 'unsigned long'
0046 // Leprechaun.c(2371) : warning C4312: 'type cast' : conversion from 'unsigned long' to 'char *' of greater size
0047 // Leprechaun.c(2570) : warning C4312: 'type cast' : conversion from 'unsigned long' to 'char *' of greater size
0048 // Leprechaun.c(2626) : warning C4312: 'type cast' : conversion from 'unsigned long' to 'char *' of greater size
0049 // Leprechaun.c(2657) : warning C4312: 'type cast' : conversion from 'unsigned long' to 'char *' of greater size
0050 // Leprechaun.c(2663) : warning C4312: 'type cast' : conversion from 'unsigned long' to 'char *' of greater size
0051 // Leprechaun.c(2668) : warning C4312: 'type cast' : conversion from 'unsigned long' to 'char *' of greater size
0052 // Leprechaun.c(2696) : warning C4312: 'type cast' : conversion from 'unsigned long' to 'char *' of greater size
0053 // Leprechaun.c(2729) : warning C4312: 'type cast' : conversion from 'unsigned long' to 'char *' of greater size
0054 // Leprechaun.c(2743) : warning C4312: 'type cast' : conversion from 'unsigned long' to 'char *' of greater size
0055 // Leprechaun.c(2755) : warning C4312: 'type cast' : conversion from 'unsigned long' to 'char *' of greater size
0056 // Microsoft (R) Incremental Linker Version 7.10.3077
0057 // Copyright (C) Microsoft Corporation. All rights reserved.
0058 //
0059 // /out:Leprechaun.exe
0060 // Leprechaun.obj
0061 //
0062 // C:\workTemp\Leprechaun_r13++\Visual C++ Toolkit 2003\Leprechaun_r13+++++_C_EXE>
0063
0064 /*
0065 Below is the gain in 13++ and 13+++:
0066
0067 words per second performance: 5,974,513w/s
0068 word count: 4,582,451,898 of them 9,177,221 distinct
0069 Number Of Trees(GREATER THE BETTER): 2855919
0070 Number Of Hash Collisions(Distinct WORDs - Number Of Trees): 6321302
0071
0072 words per second performance: 6,329,353w/s
0073 word count: 4,582,451,898 of them 9,177,221 distinct
0074 Number Of Trees(GREATER THE BETTER): 2958681
0075 Number Of Hash Collisions(Distinct WORDs - Number Of Trees): 6218540
0076
0077 The aftermath: 6,321,302 - 6,218,540 = 102,762 less collisions while the speed of hash is not slower for sure - I call this: double trouble
    avoidance.
0078 Thanks to Fowler/No11/Vo hash inventors.
0079 */
0080
0081 /*
0082 Let's see the supplementary-clash on Intel Pentium T3400 Merom-1M 2166MHz:
0083 Binary-Search-Trees vs B-Trees of order 3
0084
0085 C:\workTemp\Leprechaun_r13++\Visual C++ Toolkit 2003\Leprechaun_step_1_PAIR-QUEST>Leprechaun_Microsoft.exe Leprechaun_vs_wikipedia_en-
    WORDS.lst Leprechaun_vs_wikipedia_en-WORDS.wrd 4777 x

```

```

0086 Leprechaun(Fast Greedy word-Ripper), revision 13+++++, written by Svalqyatchx.
0087 Leprechaun: 'Oh, well, didn't you hear? Bigger is good, but jumbo is dear.'
0088 Kaze: Let's see what a 3-way hash + 6,602,752 Binary-Search-Trees can give us,
0089     also the performance of a 3-way hash + 6,602,752 B-Trees of order 3.
0090 Size of input file with files for Leprechauning: 27
0091 Allocating memory 1863MB ... OK
0092 Size of Input TEXTual file: 146,973,879
0093 \; Word count: 12,561,874 of them 12,561,874 distinct; Done: 64/64
0094 Bytes per second performance: 14,697,387B/s
0095 Words per second performance: 1,256,187w/s
0096 Flushing unsorted words ...
0097 Time for making unsorted wordlist: 15 second(s)
0098 Deallocated memory in MB: 1863
0099 Allocated memory for words in MB: 141
0100 Allocated memory for pointers-to-words in MB: 48
0101 Sorting(with 'MultiKeyQuickSortX26Sort' by J. Bentley and R. Sedgewick) ...
0102 Sort pass 26/26 ...
0103 Flushing sorted words ...
0104 Time for sorting unsorted wordlist: 14 second(s)
0105 Leprechaun: Done.
0106
0107 [An excerpt of Leprechaun.LOG:]
0108 Number Of Trees(GREATER THE BETTER): 2786806
0109 Total Attempts to Find/Put WORDS into Binary-Search-Trees: 58,935,172
0110 Total Number of LEAFs in Binary-Search-Trees(GREATER THE BETTER): 2,786,806
0111
0112 C:\workTemp\Leprechaun_r13++\Visual C++ Toolkit 2003\Leprechaun_step_1_PAIR-QUEST>Leprechaun_Microsoft.exe Leprechaun_vs_wikipedia_en-
WORDS.lst Leprechaun_vs_wikipedia_en-WORDS.wrd 4777 y
0113 Leprechaun(Fast Greedy Word-Ripper), revision 13+++++, written by Svalqyatchx.
0114 Leprechaun: 'Oh, well, didn't you hear? Bigger is good, but jumbo is dear.'
0115 Kaze: Let's see what a 3-way hash + 6,602,752 Binary-Search-Trees can give us,
0116     also the performance of a 3-way hash + 6,602,752 B-Trees of order 3.
0117 Size of input file with files for Leprechauning: 27
0118 Allocating memory 1863MB ... OK
0119 Size of Input TEXTual file: 146,973,879
0120 \; Word count: 12,561,874 of them 12,561,874 distinct; Done: 64/64
0121 Bytes per second performance: 24,495,646B/s
0122 Words per second performance: 2,093,645w/s
0123 Flushing unsorted words ...
0124 Time for making unsorted wordlist: 12 second(s)
0125 Deallocated memory in MB: 1863
0126 Allocated memory for words in MB: 141
0127 Allocated memory for pointers-to-words in MB: 48
0128 Sorting(with 'MultiKeyQuickSortX26Sort' by J. Bentley and R. Sedgewick) ...
0129 Sort pass 26/26 ...
0130 Flushing sorted words ...
0131 Time for sorting unsorted wordlist: 14 second(s)
0132 Leprechaun: Done.
0133
0134 [An excerpt of Leprechaun.LOG:]
0135 Number Of Trees(GREATER THE BETTER): 2786806
0136 Total Attempts to Find/Put WORDS into B-trees order 3: 18,534,910
0137
0138 C:\workTemp\Leprechaun_r13++\Visual C++ Toolkit 2003\Leprechaun_step_1_PAIR-QUEST>type Leprechaun_vs_wikipedia_en-WORDS.lst
0139 wikipedia-en-html.tar.wrd
0140
0141 C:\workTemp\Leprechaun_r13++\Visual C++ Toolkit 2003\Leprechaun_step_1_PAIR-QUEST>dir Leprechaun_vs_wikipedia_en-WORDS.*
0142 Volume in drive C is H320_Vol2
0143 Volume Serial Number is A094-FAE2
0144
0145 Directory of C:\workTemp\Leprechaun_r13++\Visual C++ Toolkit 2003\Leprechaun_step_1_PAIR-QUEST
0146
0147 09/14/2010  06:04 AM                27 Leprechaun_vs_wikipedia_en-WORDS.lst
0148 09/15/2010  02:51 AM       146,973,879 Leprechaun_vs_wikipedia_en-WORDS.wrd
0149             2 File(s)       146,973,906 bytes
0150             0 Dir(s)        965,787,648 bytes free
0151
0152 Conclusion:
0153 18,534,910/12,561,874=1.475 Average Attempts to Find/Put WORDS into B-trees order 3, not bad at all.
0154 */
0155
0156 // To do: must learn how to align, at last.
0157 /*
0158 Matt Mahoney ZPAQ fragment:
0159     T *data; // allocated memory
0160     int offset;
0161     ...
0162     offset=64-int((long)data&63);
0163     data=(T*)((char*)data+offset); // adjust to 64 byte boundary
0164
0165 quicklz.c fragment:
0166 #define QLZ_ALIGNMENT_PADD 8
0167 unsigned char *scratch_aligned = (unsigned char *)scratch_compress + QLZ_ALIGNMENT_PADD - (((size_t)scratch_compress) % QLZ_ALIGNMENT_PADD);
0168 size_t *buffersize = (size_t *)scratch_aligned;
0169
0170 minilzo.c fragment:
0171 #define lzo_uintptr_t      unsigned long
0172 #define PTR(a)             ((lzo_uintptr_t) (a))

```

```

0173 #define PTR_LINEAR(a)      PTR(a)
0174 #define PTR_ALIGNED_4(a)    ((PTR_LINEAR(a) & 3) == 0)
0175 */
0176
0177 //__declspec(aligned(64)) int BigArray[1024]; // windows syntax
0178 //or
0179 //int BigArray[1024] __attribute__((aligned(64))); // Linux syntax
0180
0181 #if defined(_WIN32_ENVIRONMENT_)
0182 __declspec(aligned(64))
0183 #else
0184 //__attribute__((aligned(64)));
0185 #endif /* defined(_WIN32_ENVIRONMENT_) */
0186
0187 typedef unsigned short WORD; // As for 'with *(DWORD*)', a buffer overrun is possible at the end of a memory page.' I knew about it but was
    fooled by assembly code generated by VS2010 which translates it to a word access:
0188 //; 792 : hash32 = FNV_32A_OP32(hash32, *(UINT*)p&0xFFFF);
0189
0190 typedef unsigned int UINT;
0191 typedef unsigned int DWORD;
0192
0193 /*
0194 Enter-the-BESTer or an alchemical clash of pairs of primes.
0195
0196 When an x-bit hash where x < 16 and is not a power of 2 is needed,
0197 here comes 'FNV1A_Hash_4_OCTETS': a slightly tuned FNV1A hash for a huge(22,202,980) wordlist of latin-letters-words.
0198
0199 Two improvements for the generic(base) FNV1A hash:
0200 - first, better speed: by reducing 'imul' instructions when string is 4++ chars
0201 - second, better dispersion: by experimenting(superficially-lite test done, so far) with 'FNV1_32_PRIME'
0202
0203 Or more concretely:
0204 - For FNV1_32_INIT = 2166136261
0205 - Giving to 'FNV1_32_PRIME' all primes between 2 and 11987
0206 - Shifting by 16bits instead of 13bits, when 8192 slots are used
0207
0208 C code:
0209 typedef unsigned char u_int8_t;
0210 typedef unsigned long u_int32_t;
0211
0212 #define FNV1_32_INIT ((u_int32_t)2166136261)
0213 #define FNV1_32_PRIME ((u_int32_t)1607)
0214
0215 #define FNV_32A_OP(hash, octet) \
0216     (((u_int32_t)(hash) ^ (u_int8_t)(octet)) * FNV1_32_PRIME)
0217
0218 #define FNV_32A_OP32(hash, octet) \
0219     (((u_int32_t)(hash) ^ (u_int32_t)(octet)) * FNV1_32_PRIME)
0220
0221 0800 // Invoking: FNV1A_Hash_4_OCTETS(wrd, wrdlen>>2) // = 0,1,2,3,4,5,6,7 [1..31]
0222 0801 int FNV1A_Hash_4_OCTETS(char *str, int wrdlen_QUADRUPLSTS)
0223 0802 {
0224 0803     u_int32_t hash;
0225 0804     char *p;
0226 0805
0227 0806     hash = FNV1_32_INIT;
0228 0807     p=str;
0229 0808
0230 0809     // The goal of stage #1: to reduce number of 'imul's.
0231 0810
0232 0811     // Stage #1:
0233 0812     for (; wrdlen_QUADRUPLSTS != 0; --wrdlen_QUADRUPLSTS) {
0234 0813         hash = FNV_32A_OP32(hash, (unsigned long)*(long *)p); // mov edi, DWORD PTR [eax]
0235 0814         p=p+4; // add eax, 4
0236 0815     }
0237 0816
0238 0817     // Stage #2:
0239 0818     for (; *p; ++p) {
0240 0819         hash = FNV_32A_OP(hash, *p); // mov dl, BYTE PTR [eax]
0241 0820     }
0242 0821
0243 0822     //return ((hash>>13) ^ hash) & 8191; // (((u_int32_t)1<<(x))-1) where x=13
0244 0823     return ((hash>>16) ^ hash) & 8191; // 00..8191 i.e. 2^13=8192
0245 0824 }
0246
0247 Assembler code:
0248 _FNV1A_Hash_4_OCTETS PROC NEAR
0249 ; Line 812
0250 mov     edx, DWORD PTR _wrdlen_QUADRUPLSTS$[esp-4]
0251 test    edx, edx
0252 mov     eax, DWORD PTR _str$[esp-4]
0253 push    esi
0254 mov     esi, DWORD PTR _FNV1_32_PRIME
0255 mov     ecx, -2128831035
0256 je      SHORT $L1612
0257 push    edi
0258 npad    7
0259 $L1610:

```

```

0260 ; Line 813
0261 mov edi, DWORD PTR [eax]
0262 xor edi, ecx
0263 imul edi, esi
0264 ; Line 814
0265 add eax, 4
0266 dec edx
0267 mov ecx, edi
0268 jne SHORT $L1610
0269 pop edi
0270 $L1612:
0271 ; Line 818
0272 mov dl, BYTE PTR [eax]
0273 test dl, dl
0274 je SHORT $L1619
0275 $L1617:
0276 ; Line 819
0277 movzx     edx, dl
0278 xor edx, ecx
0279 imul edx, esi
0280 inc eax
0281 mov ecx, edx
0282 mov dl, BYTE PTR [eax]
0283 test dl, dl
0284 jne SHORT $L1617
0285 $L1619:
0286 ; Line 823
0287 mov eax, ecx
0288 shr eax, 16
0289 xor eax, ecx
0290 and eax, 8191
0291 pop esi
0292 ; Line 824
0293 ret 0
0294 _FNV1A_Hash_4_OCTETS ENDP
0295
0296
0297 So, 'FNV1A_Hash_4_OCTETS' calculates faster and gives better distribution(3549448 for 1607), which is 0.6% better(less collisions), than
generic 'FNV1A_Hash' with 3527916.
0298
0299 FNV proves to be great, dealing with 4x8bits(four octets) at once doesn't hurt distribution at all, I was amazed by consistency(stable
behaviour) of 'FNV1A_Hash_4_OCTETS'.
0300
0301 I want to make a total clash of all possible pairs 'FNV1_32_INIT' & 'FNV1_32_PRIME' in order to lessen even a few thousand collisions.
0302 This is critical for speed performance e.g. when 30,974,750,142 words, the case of wikipedia-en-html.tar, must be hashed.
0303 The current obstacle is needed-time: each filling (26 slots x 31 sub-slots x 8192 sub-sub-slots) executes in 32-36 seconds for each pair.
0304 Such an easy task, but I can't see how to get done, it is not hard but slow even with 15 times faster testbed.
0305
0306 Between 1..1166136247 there are 58,834,113 primes (inclusive).
0307 Between 1..16777619 there are 1,077,891 primes (inclusive).
0308 Or 58834113*1077891 = 63,416,760,895,683 pairs or 2,010,932 years needed at one-pair-per-second rate.
0309
0310 Finding THE best pair in my opinion is a total alchemy, due to the very nature of hashing: which is mainly alchemical and partly scientific.
0311 Since the magnum corpus of words is static-enough, THE pair is worthy to be found.
0312
0313 It doesn't take a think-tank to see the superiority of FNV, Fowler/Noll/Vo did reveal a thing of beauty.
0314
0315 Performance of 'FNV1A_Hash_4_OCTETS': 10236 words/clock or 105 MB/s|3,549,448 used slots (best)
0316
0317 CASE #1: with 'if (strlen(backup[j]) != 0)' before each execution
0318 Performance of 'kuxHash3plus' aka '2in1': 8076 words/clock or 82 MB/s|3,410,463 used slots (worst)
0319 Performance of 'FNV1A_Hash': 8079 words/clock or 83 MB/s|3,527,916 used slots
0320 Performance of 'FNV1A_Hash_SHIFTless_XORless': 8109 words/clock or 83 MB/s|3,540,323 used slots
0321
0322 CASE #2: without 'if (strlen(backup[j]) != 0)' before each execution
0323 Performance of 'kuxHash3plus' aka '2in1': 11673 words/clock or 119 MB/s|3,410,463 used slots (worst)
0324 Performance of 'FNV1A_Hash': 11558 words/clock or 118 MB/s|3,527,916 used slots
0325 Performance of 'FNV1A_Hash_SHIFTless_XORless': 11570 words/clock or 118 MB/s|3,540,323 used slots
0326
0327 Note:
0328 The 'strlen' overhead(CASE #1) is necessary due to priorly(before hash invocation) needed len-of-string for 'FNV1A_Hash_4_OCTETS'.
0329 Almost always, that is the case, since parsing of incoming text must know length of words/lines/files.
0330 In case of not knowing this length: ((119-105)/105)*100% = 13% degradation is unacceptable.
0331 The 'strlen' is an awful brake.
0332 Also whether the code overhead(one additional cycle) of 'FNV1A_Hash_4_OCTETS' is so successful(as a trade-off) or the testbed is deceiving I
do not know, here I am not so sure regardless of notorious delays caused by 'imul' and 'div' instructions.
0333 */
0334
0335 /*
0336 FNV1_32_PRIME: //?: 16777619
0337
0338 Above Binary-Search-Tree with MaxPEAK = 61 has NODES = 61 and LEAFS = 1
0339 Words per second performance: 1,046,822w/s
0340 Input File with a list of TEXTual Files: Leprechaun_vs_Wikipedia_en-WORDS.lst
0341 Size of all TEXTual Files: 146,973,879
0342 Word count: 12,561,874 of them 12,561,874 distinct
0343 Number Of Trees(GREATER THE BETTER): 2775839
0344

```

0345 Above Binary-Search-Tree with MaxPEAK = 39 has NODEs = 72 and LEAFs = 15
 0346 words per second performance: 1,356,588w/s
 0347 Input File with a list of TEXTual Files: Leprechaun_vs_Wikipedia_LATIN-WORDS.lst
 0348 Size of all TEXTual Files: 415,982,896
 0349 word count: 35,271,297 of them 22,202,980 distinct
 0350 Number Of Trees(GREATER THE BETTER): 3539690
 0351
 0352 FNV1_32_PRIME: //3549448: 1607
 0353
 0354 Above Binary-Search-Tree with MaxPEAK = 61 has NODEs = 61 and LEAFs = 1
 0355 words per second performance: 1,046,822w/s
 0356 Input File with a list of TEXTual Files: Leprechaun_vs_Wikipedia_en-WORDS.lst
 0357 Size of all TEXTual Files: 146,973,879
 0358 word count: 12,561,874 of them 12,561,874 distinct
 0359 Number Of Trees(GREATER THE BETTER): 2783970
 0360
 0361 Above Binary-Search-Tree with MaxPEAK = 38 has NODEs = 50 and LEAFs = 11
 0362 words per second performance: 1,410,851w/s
 0363 Input File with a list of TEXTual Files: Leprechaun_vs_Wikipedia_LATIN-WORDS.lst
 0364 Size of all TEXTual Files: 415,982,896
 0365 word count: 35,271,297 of them 22,202,980 distinct
 0366 Number Of Trees(GREATER THE BETTER): 3549395
 0367
 0368 FNV1_32_PRIME: //3550132: 175757909
 0369
 0370 Above Binary-Search-Tree with MaxPEAK = 60 has NODEs = 60 and LEAFs = 1
 0371 words per second performance: 966,298w/s
 0372 Input File with a list of TEXTual Files: Leprechaun_vs_Wikipedia_en-WORDS.lst
 0373 Size of all TEXTual Files: 146,973,879
 0374 word count: 12,561,874 of them 12,561,874 distinct
 0375 Number Of Trees(GREATER THE BETTER): 2784479
 0376
 0377 Above Binary-Search-Tree with MaxPEAK = 39 has NODEs = 64 and LEAFs = 12
 0378 words per second performance: 1,410,851w/s
 0379 Input File with a list of TEXTual Files: Leprechaun_vs_Wikipedia_LATIN-WORDS.lst
 0380 Size of all TEXTual Files: 415,982,896
 0381 word count: 35,271,297 of them 22,202,980 distinct
 0382 Number Of Trees(GREATER THE BETTER): 3550115
 0383
 0384 FNV1_32_PRIME: //3550687: 201887489
 0385
 0386 Above Binary-Search-Tree with MaxPEAK = 60 has NODEs = 60 and LEAFs = 1
 0387 words per second performance: 966,298w/s
 0388 Input File with a list of TEXTual Files: Leprechaun_vs_Wikipedia_en-WORDS.lst
 0389 Size of all TEXTual Files: 146,973,879
 0390 word count: 12,561,874 of them 12,561,874 distinct
 0391 Number Of Trees(GREATER THE BETTER): 2784377
 0392
 0393 Above Binary-Search-Tree with MaxPEAK = 40 has NODEs = 55 and LEAFs = 11
 0394 words per second performance: 1,356,588w/s
 0395 Input File with a list of TEXTual Files: Leprechaun_vs_Wikipedia_LATIN-WORDS.lst
 0396 Size of all TEXTual Files: 415,982,896
 0397 word count: 35,271,297 of them 22,202,980 distinct
 0398 Number Of Trees(GREATER THE BETTER): 3550528
 0399
 0400 FNV1_32_PRIME: //3550733: 172783361
 0401
 0402 Above Binary-Search-Tree with MaxPEAK = 59 has NODEs = 59 and LEAFs = 1
 0403 words per second performance: 1,046,822w/s
 0404 Input File with a list of TEXTual Files: Leprechaun_vs_Wikipedia_en-WORDS.lst
 0405 Size of all TEXTual Files: 146,973,879
 0406 word count: 12,561,874 of them 12,561,874 distinct
 0407 Number Of Trees(GREATER THE BETTER): 2786362
 0408
 0409 Above Binary-Search-Tree with MaxPEAK = 38 has NODEs = 70 and LEAFs = 17
 0410 words per second performance: 1,410,851w/s
 0411 Input File with a list of TEXTual Files: Leprechaun_vs_Wikipedia_LATIN-WORDS.lst
 0412 Size of all TEXTual Files: 415,982,896
 0413 word count: 35,271,297 of them 22,202,980 distinct
 0414 Number Of Trees(GREATER THE BETTER): 3550746
 0415
 0416 FNV1_32_PRIME: //3550929: 204312319
 0417
 0418 Above Binary-Search-Tree with MaxPEAK = 61 has NODEs = 61 and LEAFs = 1
 0419 words per second performance: 966,298w/s
 0420 Input File with a list of TEXTual Files: Leprechaun_vs_Wikipedia_en-WORDS.lst
 0421 Size of all TEXTual Files: 146,973,879
 0422 word count: 12,561,874 of them 12,561,874 distinct
 0423 Number Of Trees(GREATER THE BETTER): 2785581
 0424
 0425 Above Binary-Search-Tree with MaxPEAK = 37 has NODEs = 55 and LEAFs = 12
 0426 words per second performance: 1,356,588w/s
 0427 Input File with a list of TEXTual Files: Leprechaun_vs_Wikipedia_LATIN-WORDS.lst
 0428 Size of all TEXTual Files: 415,982,896
 0429 word count: 35,271,297 of them 22,202,980 distinct
 0430 Number Of Trees(GREATER THE BETTER): 3550886
 0431
 0432 Leprechaun_Microsoft.exe: FNV1_32_PRIME: //3551736: 107712257

```

0433
0434 Above Binary-Search-Tree with MaxPEAK = 61 has NODEs = 61 and LEAFs = 1
0435 words per second performance: 1,046,822w/s
0436 Input File with a list of TEXTual Files: Leprechaun_vs_wikipedia_en-WORDS.lst
0437 Size of all TEXTual Files: 146,973,879
0438 word count: 12,561,874 of them 12,561,874 distinct
0439 Number Of Trees(GREATER THE BETTER): 2786515
0440
0441 Above Binary-Search-Tree with MaxPEAK = 36 has NODEs = 64 and LEAFs = 15
0442 words per second performance: 1,356,588w/s
0443 Input File with a list of TEXTual Files: Leprechaun_vs_wikipedia_LATIN-WORDS.lst
0444 Size of all TEXTual Files: 415,982,896
0445 word count: 35,271,297 of them 22,202,980 distinct
0446 Number Of Trees(GREATER THE BETTER): 3551744
0447
0448 Leprechaun_Intel.exe: FNV1_32_PRIME: //3551736: 107712257
0449
0450 Above Binary-Search-Tree with MaxPEAK = 61 has NODEs = 61 and LEAFs = 1
0451 words per second performance: 1,256,187w/s
0452 Input File with a list of TEXTual Files: Leprechaun_vs_wikipedia_en-WORDS.lst
0453 Size of all TEXTual Files: 146,973,879
0454 word count: 12,561,874 of them 12,561,874 distinct
0455 Number Of Trees(GREATER THE BETTER): 2786515
0456
0457 Above Binary-Search-Tree with MaxPEAK = 36 has NODEs = 64 and LEAFs = 15
0458 words per second performance: 1,603,240w/s
0459 Input File with a list of TEXTual Files: Leprechaun_vs_wikipedia_LATIN-WORDS.lst
0460 Size of all TEXTual Files: 415,982,896
0461 word count: 35,271,297 of them 22,202,980 distinct
0462 Number Of Trees(GREATER THE BETTER): 3551744
0463
0464 wow: 1,603,240w/s vs 1,356,588w/s respectively Leprechaun_Intel.exe vs Leprechaun_Microsoft.exe, i.e. 18% betterment, no joke!
0465
0466 Alchemical search for best PRIME-PAIR revision uses next line:
0467 Slot = FNV1A_Hash_4_OCTETS(wrd, wrdlen>>2)<<2; //13+++
0468 This revision uses next lines:
0469 if (wrdlen<=19) // 4x4+3=19 i.e. last contains 7 clashes
0470     Slot = FNV1A_Hash_Granularity(wrd, wrdlen>>2, 2)<<2; //13++++
0471 else // 2x8+4=20 i.e. first contains 6 clashes
0472     Slot = FNV1A_Hash_Granularity(wrd, wrdlen>>3, 3)<<2; //13++++
0473
0474 ! An expected but unpleasant degradation for 3551961: 428904191 compared to 3551736: 107712257, this shows 'FNV1A_Hash_4_OCTETS' has only
    figurative purpose - the 4 lines of 'FNV1A_Hash_Granularity' decide the last usefulness.
0475
0476 Leprechaun.exe: FNV1_32_PRIME: //3551961: 428904191
0477
0478 Above Binary-Search-Tree with MaxPEAK = 60 has NODEs = 60 and LEAFs = 1
0479 words per second performance: 966,298w/s
0480 Input File with a list of TEXTual Files: Leprechaun_vs_wikipedia_en-WORDS.lst
0481 Size of all TEXTual Files: 146,973,879
0482 word count: 12,561,874 of them 12,561,874 distinct
0483 Number Of Trees(GREATER THE BETTER): 2786383
0484
0485 Above Binary-Search-Tree with MaxPEAK = 39 has NODEs = 71 and LEAFs = 16
0486 words per second performance: 1,410,851w/s
0487 Input File with a list of TEXTual Files: Leprechaun_vs_wikipedia_LATIN-WORDS.lst
0488 Size of all TEXTual Files: 415,982,896
0489 word count: 35,271,297 of them 22,202,980 distinct
0490 Number Of Trees(GREATER THE BETTER): 3551503
0491
0492 Leprechaun.exe: FNV1_32_PRIME: //3552103: 588411137
0493
0494 Above Binary-Search-Tree with MaxPEAK = 6 has NODEs = 6 and LEAFs = 1
0495 Size of all TEXTual Files: 4,067,439
0496 word count: 358,798 of them 351,116 distinct
0497 Number Of Trees(GREATER THE BETTER): 310622
0498 Total Number of LEAFs in Binary-Search-Trees(GREATER THE BETTER): 310,622
0499
0500 Above Binary-Search-Tree with MaxPEAK = 60 has NODEs = 60 and LEAFs = 1
0501 Size of all TEXTual Files: 146,973,879
0502 word count: 12,561,874 of them 12,561,874 distinct
0503 Number Of Trees(GREATER THE BETTER): 2786485
0504 Total Number of LEAFs in Binary-Search-Trees(GREATER THE BETTER): 2,786,485
0505
0506 Above Binary-Search-Tree with MaxPEAK = 39 has NODEs = 62 and LEAFs = 15
0507 Size of all TEXTual Files: 415,982,896
0508 word count: 35,271,297 of them 22,202,980 distinct
0509 Number Of Trees(GREATER THE BETTER): 3551956
0510 Total Number of LEAFs in Binary-Search-Trees(GREATER THE BETTER): 8,072,131
0511
0512 Leprechaun.exe: FNV1_32_PRIME: //3552039: 602173697 !!!GOODEST so far!!!
0513
0514 Above Binary-Search-Tree with MaxPEAK = 6 has NODEs = 6 and LEAFs = 1
0515 Size of all TEXTual Files: 4,067,439
0516 word count: 358,798 of them 351,116 distinct
0517 Number Of Trees(GREATER THE BETTER): 310948
0518 Total Number of LEAFs in Binary-Search-Trees(GREATER THE BETTER): 310,948
0519

```

```

0520 Above Binary-Search-Tree with MaxPEAK = 63 has NODES = 63 and LEAFs = 1
0521 Size of all TEXTual Files: 146,973,879
0522 word count: 12,561,874 of them 12,561,874 distinct
0523 Number Of Trees(GREATER THE BETTER): 2786806
0524 Total Number of LEAFs in Binary-Search-Trees(GREATER THE BETTER): 2,786,806
0525
0526 Above Binary-Search-Tree with MaxPEAK = 36 has NODES = 52 and LEAFs = 9
0527 Input File with a list of TEXTual Files: Leprechaun_vs_wikipedia_LATIN-WORDS.lst
0528 Size of all TEXTual Files: 415,982,896
0529 word count: 35,271,297 of them 22,202,980 distinct
0530 Number Of Trees(GREATER THE BETTER): 3552296
0531 Total Number of LEAFs in Binary-Search-Trees(GREATER THE BETTER): 8,072,899
0532
0533 Between 1 and 602392027 at step 100 following FNV1_32_PRIMES(for FNV1_32_INIT=2166136261) give(FNV1A_Hash_4_OCTETS) dispersion:
0534 3550022: 423779327
0535 3550028: 513793537
0536 3550053: 434840321
0537 3550067: 437062229
0538 3550080: 420344321
0539 3550090: 304777471
0540 3550097: 496547839
0541 3550129: 390809599
0542 3550132: 175757909
0543 3550163: 353712127
0544 3550231: 334434817
0545 3550237: 272789761
0546 3550247: 590341121
0547 3550255: 358814207
0548 3550277: 437182721
0549 3550326: 521795327
0550 3550347: 311867393
0551 3550447: 456137729
0552 3550458: 418208767
0553 3550516: 602048767
0554 3550525: 513597697
0555 3550526: 347283199
0556 3550528: 598773503
0557 3550592: 598139137
0558 3550598: 242448127
0559 3550611: 571481087
0560 3550628: 457012993
0561 3550664: 482822143
0562 3550666: 249098753
0563 3550687: 201887489
0564 3550702: 489976063
0565 3550710: 272961023
0566 3550733: 172783361
0567 3550734: 431562497
0568 3550929: 204312319
0569 3550984: 562853633
0570 3550991: 551362303
0571 3551359: 332820737
0572 3551484: 354126079
0573 3551514: 407138561
0574 3551523: 442058753
0575 3551701: 449230849
0576 3551736: 107712257
0577 3551961: 428904191
0578 3552039: 602173697
0579 3552103: 588411137
0580 */
0581
0582 // windows: ~~~~~
0583 // _CRTIMP size_t __cdecl fread(void *, size_t, size_t, FILE *);
0584 // _CRTIMP size_t __cdecl fwrite(const void *, size_t, size_t, FILE *);
0585 // _CRTIMP int __cdecl fgetpos(FILE *, fpos_t *);
0586 // _CRTIMP int __cdecl fsetpos(FILE *, const fpos_t *);
0587
0588 // _CRTIMP __int64 __cdecl _lseeki64(int, __int64, int);
0589 // _CRTIMP __int64 __cdecl _telli64(int);
0590 // _CRTIMP __int64 __cdecl _filelengthi64(int);
0591 // above 3 are in 'io.h'
0592
0593 // _CRTIMP int __cdecl fseek(FILE *, long, int);
0594 // _CRTIMP long __cdecl ftell(FILE *);
0595 // _CRTIMP int __cdecl fclose(FILE *);
0596
0597 // #ifndef _SIZE_T_DEFINED
0598 // #ifdef _WIN64
0599 // typedef unsigned __int64 size_t;
0600 // #else
0601 // typedef _w64 unsigned int size_t;
0602 // #endif
0603 // #define _SIZE_T_DEFINED
0604 // #endif
0605
0606 // typedef __int64 fpos_t;
0607

```



```

0608 // Linux: ~~~~~
0609 // size_t fread (void *data, size_t size, size_t count, FILE *stream)
0610 // size_t fwrite (const void *data, size_t size, size_t count, FILE *stream)
0611 // int fgetpos (FILE *stream, fpos_t *position)
0612 // int fsetpos (FILE *stream, const fpos_t *position)
0613
0614 // FILE * fopen64 (const char *filename, const char *opentype)
0615 // int fseeko64 (FILE *stream, off64_t offset, int whence)
0616 // off64_t ftello64 (FILE *stream)
0617 // int fclose (FILE *stream)
0618
0619 // off_t lseek (int filedes, off_t offset, int whence)
0620 // above 1 is in 'unistd.h'
0621
0622
0623 // ===== MUST work both for windows and Linux =====
0624 #define _WIN32_ENVIRONMENT_
0625 // #define _POSIX_ENVIRONMENT_
0626
0627
0628 #ifndef NULL
0629 #ifdef __cplusplus
0630 #define NULL 0
0631 #else
0632 #define NULL ((void*)0)
0633 #endif
0634 #endif
0635
0636 // To do #1: Put this 31 in MAXw1: 'int MAXw1 = 31;'
0637 // To do #2: No need of flushing unsorted words to file: make backup[] array
0638 //           instead of writing. And mostly sort 26 times!
0639 // HEAVY BUG in r.7: unsigned long Hll(unsigned long n)
0640 //                 is NOT identical with
0641 //                 unsigned long GRMBLhll[32]; // 00 not used, only 01..31
0642 //                 BECAUSE DUMBEST DUMB Array GRMBLhll expects 'int' not
0643 //                 'unsigned long' !!!
0644
0645 #include <stdio.h>
0646 #include <ctype.h>
0647 #include <time.h>
0648 #if defined(_WIN32_ENVIRONMENT_)
0649 #include <io.h> // needed for windows 'lseeki64' and 'telli64'
0650 // Above line must be commented in order to compile with Intel C compiler: an error "can't find io.h" occurs.
0651 #else
0652 #endif /* defined(_WIN32_ENVIRONMENT_) */
0653
0654 typedef unsigned char char_t;
0655 typedef char_t *string;
0656
0657 clock_t clocks1, clocks2;
0658 int Bozan;
0659
0660 typedef unsigned char u_int8_t; //FNV only
0661 typedef unsigned long u_int32_t; //FNV only
0662 typedef unsigned long long u_int64_t; //FNV only
0663
0664 // SINHA fragment[
0665
0666 #define swapKAZE(a, b) { t = *(a); *(a) = *(b); *(b) = t; }
0667
0668 static void InsertSortKAZE(string *a, int n, int d) //void inssort(unsigned char **a, int n, int d)
0669 {
0670     string *pi, *pj, s, t; //unsigned char **pi, **pj, *s, *t;
0671     for (pi = a + 1; --n > 0; pi++)
0672         for (pj = pi; pj > a; pj--) {
0673             /* Inline strcmp: break if *(pj-1) <= *pj */
0674             for (s=*(pj-1)+d, t=*pj+d; *s==*t && *s!=0; s++, t++)
0675                 ;
0676             if (*s <= *t)
0677                 break;
0678             swapKAZE(pj, pj-1);
0679         }
0680
0681 //int cmpit(unsigned char **h1, unsigned char **h2)
0682 //{
0683 //    return( strcmp(*h1, *h2) );
0684 //}
0685
0686 //int scmp( unsigned char *s1, unsigned char *s2 )
0687 //{
0688 //    while( *s1 != '\0' && *s1 == *s2 )
0689 //        {
0690 //            s1++;
0691 //            s2++;
0692 //        }
0693 //    return( *s1-*s2 );
0694 //}
0695

```

```

0696 //static void simplesort(string a[], int n, int b)
0697 //{
0698 //    int i, j;
0699 //    string tmp;
0700 //
0701 //    for (i = 1; i < n; i++)
0702 //        for (j = i; j > 0 && strcmp(a[j-1]+b, a[j]+b) > 0; j--)
0703 //            { tmp = a[j]; a[j] = a[j-1]; a[j-1] = tmp; }
0704 //}
0705
0706 // SINHA fragment]
0707
0708 // mkqsort.c BEGIN *****
0709 /*
0710  Multikey quicksort, a radix sort algorithm for arrays of character
0711  strings by Bentley and Sedgewick.
0712
0713  J. Bentley and R. Sedgewick. Fast algorithms for sorting and
0714  searching strings. In Proceedings of 8th Annual ACM-SIAM Symposium
0715  on Discrete Algorithms, 1997.
0716
0717  http://www.CS.Princeton.EDU/~rs/strings/index.html
0718
0719  The code presented in this file has been tested with care but is
0720  not guaranteed for any purpose. The writer does not offer any
0721  warranties nor does he accept any liabilities with respect to
0722  the code.
0723
0724  Ranjan Sinha, 1 jan 2003.
0725
0726  School of Computer Science and Information Technology,
0727  RMIT University, Melbourne, Australia
0728  rsinha@cs.rmit.edu.au
0729
0730 */
0731
0732 //include "sortstring.h"
0733
0734 /* MULTIKEY QUICKSORT */
0735
0736 #ifndef min
0737 #define min(a, b) ((a)<=(b) ? (a) : (b))
0738 #endif
0739
0740
0741 // ----- BTREE [
0742 #define false -1
0743 #define true 0
0744
0745 struct nodeBTREE {
0746     int data;
0747     struct nodeBTREE* left;
0748     struct nodeBTREE* right;
0749 };
0750
0751 // ----- BTREE ]
0752
0753
0754 /* ssort2 -- Faster Version of Multikey Quicksort */
0755
0756 void vecswap2(unsigned char **a, unsigned char **b, int n)
0757 { while (n-- > 0) {
0758     unsigned char *t = *a;
0759     *a++ = *b;
0760     *b++ = t;
0761 }
0762 }
0763
0764 #define swap2(a, b) { t = *(a); *(a) = *(b); *(b) = t; }
0765 #define ptr2char(i) (*(i) + depth)
0766
0767 unsigned char **med3func(unsigned char **a, unsigned char **b, unsigned char **c, int depth)
0768 { int va, vb, vc;
0769     if ((va=ptr2char(a)) == (vb=ptr2char(b)))
0770         return a;
0771     if ((vc=ptr2char(c)) == va || vc == vb)
0772         return c;
0773     return va < vb ?
0774         (vb < vc ? b : (va < vc ? c : a)) :
0775         (vb > vc ? b : (va < vc ? a : c));
0776 }
0777 #define med3(a, b, c) med3func(a, b, c, depth)
0778
0779 void insort(unsigned char **a, int n, int d)
0780 { unsigned char **pi, **pj, *s, *t;
0781     for (pi = a + 1; --n > 0; pi++)
0782         for (pj = pi; pj > a; pj--) {
0783             /* Inline strcmp: break if *(pj-1) <= *pj */

```

```

0784         for (s=*(pj-1)+d, t=*pj+d; *s==*t && *s!=0; s++, t++)
0785             ;
0786         if (*s <= *t)
0787             break;
0788         swap2(pj, pj-1);
0789     }
0790 }
0791
0792 void mkqsort(unsigned char **a, int n, int depth)
0793 { int d, r, partval;
0794   unsigned char **pa, **pb, **pc, **pd, **pl, **pm, **pn, **t;
0795   if (n < 20) {
0796       insort(a, n, depth);
0797       return;
0798   }
0799   pl = a;
0800   pm = a + (n/2);
0801   pn = a + (n-1);
0802   if (n > 30) { /* On big arrays, pseudomedian of 9 */
0803       d = (n/8);
0804       pl = med3(pl, pl+d, pl+2*d);
0805       pm = med3(pm-d, pm, pm+d);
0806       pn = med3(pn-2*d, pn-d, pn);
0807   }
0808   pm = med3(pl, pm, pn);
0809   swap2(a, pm);
0810   partval = ptr2char(a);
0811   pa = pb = a + 1;
0812   pc = pd = a + n-1;
0813   for (;;) {
0814       while (pb <= pc && (r = ptr2char(pb)-partval) <= 0) {
0815           if (r == 0) { swap2(pa, pb); pa++; }
0816           pb++;
0817       }
0818       while (pb <= pc && (r = ptr2char(pc)-partval) >= 0) {
0819           if (r == 0) { swap2(pc, pd); pd--; }
0820           pc--;
0821       }
0822       if (pb > pc) break;
0823       swap2(pb, pc);
0824       pb++;
0825       pc--;
0826   }
0827   pn = a + n;
0828   r = min(pa-a, pb-pa); vecswap2(a, pb-r, r);
0829   r = min(pd-pc, pn-pd-1); vecswap2(pb, pn-r, r);
0830   if ((r = pb-pa) > 1)
0831       mkqsort(a, r, depth);
0832   if (ptr2char(a + r) != 0)
0833       mkqsort(a + r, pa-a + pn-pd-1, depth+1);
0834   if ((r = pd-pc) > 1)
0835       mkqsort(a + n-r, r, depth);
0836 }
0837
0838 void mkqsort_main(unsigned char **a, int n) { mkqsort(a, n, 0); }
0839 // mkqsort.c END *****
0840
0841 // why sinha uses int instead of long??!!
0842 static int readlines(char *file_name, string **lines)
0843 {
0844     int nlines = 0;
0845     size_t size;
0846     FILE *in_file;
0847     string basep, cur, next;
0848     string *ASbackup;
0849
0850     if (!(in_file = fopen(file_name, "rb"))) {
0851         printf("Leprechaun: Can't open file %s \n", file_name);
0852         exit(-1);
0853     }
0854     fseek(in_file, 0, SEEK_END);
0855     size = ftell(in_file);
0856     fseek(in_file, 0, SEEK_SET);
0857     if (!(basep = (string) malloc(size*sizeof(char_t)))) return -1;
0858     printf("Allocated memory for words in MB: %lu\n", ((size*sizeof(char_t))>>20)+1);
0859     if (fread(basep, 1, size, in_file) < size) {
0860         printf("Leprechaun: Can't read file %s \n", file_name);
0861         exit(-1);
0862     }
0863     fclose(in_file);
0864
0865     // GET nlines:
0866     cur = basep;
0867     while (cur < basep + size) {
0868         next = cur;
0869         while ((next < basep + size) && (*next != '\n')) {next++;}
0870         *--next = '\0'; // This is ala DOS i.e. windows
0871         // 1310 not 10(\n=10)

```

```

0872     cur = next + 2;
0873     nlines++;
0874 }
0875
0876 // printf("%lu\n", (unsigned long)*lines); -> backup = *lines = 0
0877 ASbackup = (string *)malloc( nlines*sizeof(string) ); // sizeof(string) is 4
0878 if( ASbackup == NULL )
0879 { puts( "Leprechaun: Needed memory allocation denied!\n" ); return( 1 ); }
0880 printf( "Allocated memory for pointers-to-words in MB: %lu\n", ((nlines*sizeof(string))>>20)+1 );
0881 *lines = ASbackup;
0882 //printf("%lu\n", (unsigned long)*lines); -> backup = *lines = ASbackup = 6946888
0883
0884 // Upload nlines times:
0885 nlines = 0;
0886 cur = basep;
0887 while (cur < basep + size) {
0888     next = cur;
0889     while ((next < basep + size) && (*next != '\n')) {next++;}
0890     *--next = '\0'; // This is ala DOS i.e. windows
0891                     // 1310 not 10(\n=10)
0892     ASbackup[nlines] = cur;
0893     cur = next + 2;
0894     nlines++;
0895 }
0896 return nlines;
0897 }
0898
0899 void x64toaKAZE ( // stdcall is faster and smaller... Might as well use it for the helper. */
0900     unsigned long long val,
0901     char *buf,
0902     unsigned radix,
0903     int is_neg
0904 )
0905 {
0906     char *p; // pointer to traverse string */
0907     char *firstdig; // pointer to first digit */
0908     char temp; // temp char */
0909     unsigned digval; // value of digit */
0910
0911     p = buf;
0912
0913     if ( is_neg )
0914     {
0915         *p++ = '-'; // negative, so output '-' and negate */
0916         val = (unsigned long long)(-(long long)val);
0917     }
0918
0919     firstdig = p; // save pointer to first digit */
0920
0921     do {
0922         digval = (unsigned) (val % radix);
0923         val /= radix; // get next digit */
0924
0925         /* convert to ascii and store */
0926         if (digval > 9)
0927             *p++ = (char) (digval - 10 + 'a'); /* a letter */
0928         else
0929             *p++ = (char) (digval + '0'); /* a digit */
0930     } while (val > 0);
0931
0932     /* We now have the digit of the number in the buffer, but in reverse
0933     order. Thus we reverse them now. */
0934
0935     *p-- = '\0'; // terminate string; p points to last digit */
0936
0937     do {
0938         temp = *p;
0939         *p = *firstdig;
0940         *firstdig = temp; // swap *p and *firstdig */
0941         --p;
0942         ++firstdig; // advance to next two digits */
0943     } while (firstdig < p); // repeat until halfway */
0944 }
0945
0946 /* Actual functions just call conversion helper with neg flag set correctly,
0947 and return pointer to buffer. */
0948
0949 char * _i64toaKAZE (
0950     long long val,
0951     char *buf,
0952     int radix
0953 )
0954 {
0955     x64toaKAZE((unsigned long long)val, buf, radix, (radix == 10 && val < 0));
0956     return buf;
0957 }
0958
0959 char * _ui64toaKAZE (

```

```

0960     unsigned long long val,
0961     char *buf,
0962     int radix
0963 )
0964 {
0965     x64toakAZE(val, buf, radix, 0);
0966     return buf;
0967 }
0968
0969 char * _ui64toakAZEzerocomma (
0970     unsigned long long val,
0971     char *buf,
0972     int radix
0973 )
0974 {
0975     char *p;
0976     char temp;
0977     int txpman;
0978     int pxnman;
0979     x64toakAZE(val, buf, radix, 0);
0980     p = buf;
0981     do {
0982     } while (*++p != '\0');
0983     p--; // p points to last digit
0984     // buf points to first digit
0985     buf[26] = 0;
0986     txpman = 1;
0987     pxnman = 0;
0988     do
0989     { if (buf <= p)
0990     { temp = *p;
0991       buf[26-txpman] = temp; pxnman++;
0992       p--;
0993       if (pxnman % 3 == 0)
0994       { txpman++;
0995         buf[26-txpman] = (char) (' ');
0996       }
0997     }
0998     else
0999     { buf[26-txpman] = (char) ('0'); pxnman++;
1000       if (pxnman % 3 == 0)
1001       { txpman++;
1002         buf[26-txpman] = (char) (' ');
1003       }
1004     }
1005     txpman++;
1006     } while (txpman <= 26);
1007     return buf;
1008 }
1009
1010 char * _ui64toakAZEcomma (
1011     unsigned long long val,
1012     char *buf,
1013     int radix
1014 )
1015 {
1016     char *p;
1017     char temp;
1018     int txpman;
1019     int pxnman;
1020     x64toakAZE(val, buf, radix, 0);
1021     p = buf;
1022     do {
1023     } while (*++p != '\0');
1024     p--; // p points to last digit
1025     // buf points to first digit
1026     buf[26] = 0;
1027     txpman = 1;
1028     pxnman = 0;
1029     while (buf <= p)
1030     { temp = *p;
1031       buf[26-txpman] = temp; pxnman++;
1032       p--;
1033       if (pxnman % 3 == 0 && buf <= p)
1034       { txpman++;
1035         buf[26-txpman] = (char) (' ');
1036       }
1037       txpman++;
1038     }
1039     return buf+26-(txpman-1);
1040 }
1041
1042 unsigned char KuxHash(char *str)
1043 { unsigned char h = 0;
1044   int max31 = 0;
1045   //while (*str)
1046   while (str[max31])
1047   { h = h ^ str[max31++];

```

```

1048 //h = h ^ *str++; // I am not sure 'str' is returned changed after return?!
1049 }
1050 return h; // 00..255 i.e. 2^8=256
1051 }
1052
1053 int KuxHash2(char *str)
1054 { int h = 0;
1055   unsigned long h2 = 0; // must be long: 31*'z'=31*122
1056   int max31 = 0;
1057   while (str[max31])
1058   { h = h ^ str[max31]; // 00..255 i.e. 2^8=256
1059     //h2 = h2 + str[max31++]; // [113s]
1060     h2 = h2 + max31 * str[max31++];
1061   }
1062   h=h<<4; // 00..15 i.e. 2^4=16
1063   //h = h|( str[0] ^ str[max31-1] ); // [111s] a..z: each XOR each gives 00..31
1064   h = h|( h2%((1<<4)-1) );
1065   return h; // 00..4095 i.e. 2^12=4096
1066 }
1067
1068 //OSHO test - Attempts to Find/Put a WORD into linked list count: 32,011,937
1069 int KuxHash3(char *str)
1070 { int h = 0;
1071   unsigned long h2 = 0; // must be long: 31*'z'=31*122
1072   int max31 = 0;
1073   while (str[max31])
1074   { h = h ^ str[max31]; // 00..255 i.e. 2^8=256
1075     //h2 = h2 + str[max31++]; // [113s]
1076     h2 = h2 + str[max31++] * (max31+1);
1077   }
1078   // Result is: 7bits in 'h' and 32bits in 'h2'.
1079
1080   //printf("%s:\n ",str);
1081   //printf("%d ",h);
1082   h=h<<6; // 00..15 i.e. 00-05+7bits=13bits
1083   //printf("%d ",h);
1084   //printf("%d ",h2);
1085   //h = h|( str[0] ^ str[max31-1] ); // [111s] a..z: each XOR each gives 00..31
1086   h = h|( h2%((1<<6)-1) ); // 64-1=63=9*7; 61 is prime
1087   //printf("%d\n",h);
1088   return h; // 00..8191 i.e. 2^13=8192
1089 }
1090
1091 //OSHO test - Attempts to Find/Put a WORD into a linked list count: 31,927,285
1092 int KuxHash3plus(char *str)
1093 { int h = 0;
1094   unsigned long h2 = 0; // must be long: 31*'z'=31*122
1095   int max31 = 0;
1096   while (str[max31])
1097   { h = h ^ str[max31]; // 00..255 i.e. 2^8=256
1098     //h2 = h2 + str[max31++]; // [113s]
1099     h2 = h2 + str[max31++] * (max31+1);
1100   }
1101   // Result is: 7bits in 'h' and 32bits in 'h2'.
1102
1103   //printf("%s:\n ",str);
1104   //printf("%d ",h);
1105   // a in ASCII is 097 = 0110 0001
1106   // z in ASCII is 122 = 0111 1010
1107   // Above two lines show that bits 8-7-6 are always 0-1-1 so need for low 5 bits.
1108   //h=h<<8; // 00..15 i.e. 5bits + 00-07bits=13bits
1109   //printf("%d ",h);
1110   //printf("%d ",h2);
1111   //h = h|( str[0] ^ str[max31-1] ); // [111s] a..z: each XOR each gives 00..31
1112   h = (( h<<8 )|( h2%(251) ))&8191; // 251 prime
1113   //printf("%d\n",h);
1114   return h; // 00..8191 i.e. 2^13=8192
1115 }
1116
1117 /*
1118 PUBLIC _KuxHash3plus
1119 ; Function compile flags: /Ogty
1120 _TEXT SEGMENT
1121 _str$ = 8 ; size = 4
1122 _KuxHash3plus PROC NEAR
1123 ; Line 511
1124 mov ecx, DWORD PTR _str$[esp-4]
1125 mov dl, BYTE PTR [ecx]
1126 push esi
1127 xor esi, esi
1128 xor eax, eax
1129 test dl, dl
1130 je SHORT $L1561
1131 push ebx
1132 push edi
1133 mov edi, 1
1134 sub edi, ecx
1135 npad 8

```

```

1136 $L1560:
1137 ; Line 512
1138 movsx     edx, BYTE PTR [ecx]
1139 ; Line 514
1140 lea ebx, DWORD PTR [edi+ecx]
1141 imul ebx, edx
1142 xor esi, edx
1143 mov dl, BYTE PTR [ecx+1]
1144 add eax, ebx
1145 inc ecx
1146 test dl, dl
1147 jne SHORT $L1560
1148 pop edi
1149 pop ebx
1150 $L1561:
1151 ; Line 527
1152 xor edx, edx
1153 mov ecx, 251 ; 000000fbh
1154 div ecx
1155 shl esi, 8
1156 mov eax, edx
1157 ; Line 529
1158 or  eax, esi
1159 and eax, 8191 ; 00001ffffh
1160 pop esi
1161 ; Line 530
1162 ret 0
1163 _KuxHash3plus ENDP
1164 _TEXT      ENDS
1165 */
1166
1167 //OSHO test - Attempts to Find/Put a WORD into a linked list count: 32,021,975
1168 int KuxHash4(char *str)
1169 {
1170     int h2 = 0;
1171     for (; *str != 0; str++) {
1172         //h2 = (127*h2 + *str) % (8192-1); // 2^13-1 = 8191 is Mersenne prime
1173         h2 = ((h2<<7) + *str) % (8192-1); // 2^13-1 = 8191 is Mersenne prime
1174     }
1175
1176     return h2; // 00..8191 i.e. 2^13=8192
1177 }
1178
1179 /*
1180 int hash(char *v, int M)
1181 { int h = 0, a = 127;
1182   for (; *v != 0; v++)
1183       h = (a*h + *v) % M;
1184   return h;
1185 }
1186
1187 int hashU(char *v, int M)
1188 { int h, a = 31415, b = 27183;
1189   for (h = 0; *v != 0; v++, a = a*b % (M-1))
1190       h = (a*h + *v) % M;
1191   return (h < 0) ? (h + M) : h;
1192 }
1193 */
1194
1195 // Kaze: My appreciation of FNV is far beyound C code optimization, it is alchemical, and why not, magical.
1196
1197 /*
1198 FNV hash history
1199 The basis of the FNV hash algorithm was taken from an idea sent as reviewer comments to the IEEE POSIX P1003.2 committee
1200 by Glenn Fowler and Phong Vo back in 1991. In a subsequent ballot round: Landon Curt Noll improved on their algorithm.
1201 Some people tried this hash and found that it worked rather well. In an EMail message to Landon, they named it
1202 the "Fowler/Noll/vo" or FNV hash.
1203 FNV hashes are designed to be fast while maintaining a low collision rate. The FNV speed allows one to quickly hash
1204 lots of data while maintaining a reasonable collision rate. The high dispersion of the FNV hashes makes them well suited
1205 for hashing nearly identical strings such as URLs, hostnames, filenames, text, IP addresses, etc.
1206 */
1207
1208 /* NOTE: u_int64_t is a 64 bit unsigned type */
1209 /* NOTE: u_int32_t is a 32 bit unsigned type */
1210 /* NOTE: u_int16_t is a 16 bit unsigned type */
1211 /* NOTE: u_int8_t is a 8 bit unsigned type */
1212
1213 //typedef unsigned char u_int8_t; //FNV only
1214 //typedef unsigned long u_int32_t; //FNV only
1215 //typedef unsigned long long u_int64_t; //FNV only
1216
1217 // 32 bit FNV_prime = 2^24 + 2^8 + 0x93 = 16777619
1218 // 64 bit FNV_prime = 2^40 + 2^8 + 0xb3 = 1099511628211
1219
1220 // 32 bit offset_basis = 2166136261
1221 // 64 bit offset_basis = 14695981039346656037
1222
1223 #define FNV1_64_INIT ((u_int64_t)14695981039346656037)

```

```

1224 #define FNV1_64_PRIME ((u_int64_t)1099511628211)
1225 #define FNV1_32_INIT ((u_int32_t)2166136261)
1226 #define FNV1_32_PRIME ((u_int32_t)602173697)
1227 // FNV1A_Hash_4_OCTETS gives dispersion as follows:
1228 //3549448: 1607
1229 //3549669: 171072511
1230 //3550710: 272961023
1231 //3550733: 172783361
1232 //3550734: 431562497
1233 //3550929: 204312319
1234 //3550984: 562853633
1235 //3550991: 551362303
1236 //3551359: 332820737
1237 //3551484: 354126079
1238 //3551514: 407138561
1239 //3551523: 442058753
1240 //3551701: 449230849
1241 //3551736: 107712257
1242 //3551961: 428904191
1243 //3552039: 602173697
1244 //3552103: 588411137
1245
1246 #define FNV_64A_OP(hash, octet) \
1247     (((u_int64_t)(hash) ^ (u_int8_t)(octet)) * FNV1_64_PRIME)
1248
1249 #define FNV_64A_OP64(hash, octet) \
1250     (((u_int64_t)(hash) ^ (u_int64_t)(octet)) * FNV1_64_PRIME)
1251
1252 #define FNV_32A_OP_GENERIC(hash, octet) \
1253     (((u_int32_t)(hash) ^ (u_int8_t)(octet)) * 16777619)
1254
1255 #define FNV_32A_OP(hash, octet) \
1256     (((u_int32_t)(hash) ^ (u_int8_t)(octet)) * FNV1_32_PRIME)
1257
1258 #define FNV_32A_OP_MULless_core(hash, octet) \
1259     ( (u_int32_t)(hash) ^ (u_int8_t)(octet) )
1260
1261 #define FNV_32A_OP_MULless(hash, octet) \
1262     ( (FNV_32A_OP_MULless_core(hash, octet)<<5) - FNV_32A_OP_MULless_core(hash, octet) )
1263
1264 #define FNV_32A_OP32(hash, octet) \
1265     (((u_int32_t)(hash) ^ (u_int32_t)(octet)) * FNV1_32_PRIME)
1266
1267 #define FNV_32A_OP64(hash, octet) \
1268     (((u_int64_t)(hash) ^ (u_int64_t)(octet)) * FNV1_32_PRIME)
1269
1270 #define FNV_32A_OP32_MULless_core(hash, octet) \
1271     ( (u_int32_t)(hash) ^ (u_int32_t)(octet) )
1272
1273 #define FNV_32A_OP32_MULless(hash, octet) \
1274     ( (FNV_32A_OP32_MULless_core(hash, octet)<<5) - FNV_32A_OP32_MULless_core(hash, octet) )
1275
1276
1277 // Invoking: FNV1A_Hash_4_OCTETS_31(wrd, wrdlen>>2) // = 0,1,2,3,4,5,6,7 [1..31]
1278 int FNV1A_Hash_4_OCTETS_31(char *str, int wrdlen_QUADRUPLETS)
1279 {
1280     u_int32_t hash;
1281     char *p;
1282
1283     hash = FNV1_32_INIT;
1284     p=str;
1285
1286     // The goal of stage #1: to reduce number of 'imul's in fact to reduce loops.
1287
1288     // Stage #1:
1289     for (; wrdlen_QUADRUPLETS != 0; --wrdlen_QUADRUPLETS) {
1290         hash = FNV_32A_OP32_MULless(hash, (unsigned long)*(long *)p); // mov edi, DWORD PTR [eax]
1291         p=p+4; // add eax, 4
1292     }
1293
1294     // Stage #2:
1295     for (; *p; ++p) {
1296         hash = FNV_32A_OP_MULless(hash, *p); // mov dl, BYTE PTR [ecx]
1297     }
1298
1299     //return ((hash>>13) ^ hash) & 8191; // (((u_int32_t)1<<(x))-1) where x=13
1300     return ((hash>>16) ^ hash) & 8191; // 00..8191 i.e. 2^13=8192
1301 }
1302
1303
1304 // Invoking: FNV1A_Hash_4_OCTETS(wrd, wrdlen>>2) // = 0,1,2,3,4,5,6,7 [1..31]
1305 int FNV1A_Hash_4_OCTETS(char *str, int wrdlen_QUADRUPLETS)
1306 {
1307     u_int32_t hash;
1308     char *p;
1309
1310     hash = FNV1_32_INIT;
1311     p=str;

```



```

1312
1313 // The goal of stage #1: to reduce number of 'imul's.
1314
1315 // Stage #1:
1316 for (; wrdlen_QUADRUPLTS != 0; --wrdlen_QUADRUPLTS) {
1317     hash = FNV_32A_OP32(hash, (unsigned long)*(long *)p); // mov edi, DWORD PTR [eax]
1318     p=p+4; // add eax, 4
1319 }
1320
1321 // Stage #2:
1322 for (; *p; ++p) {
1323     hash = FNV_32A_OP(hash, *p); // mov dl, BYTE PTR [ecx]
1324 }
1325
1326 //return ((hash>>13) ^ hash) & 8191; // (((u_int32_t)1<<(x))-1) where x=13
1327 return ((hash>>16) ^ hash) & 8191; // 00..8191 i.e. 2^13=8192
1328 }
1329
1330 /*
1331 Results for 'FNV1A_Hash_8_OCTETS':
1332 Bytes per second performance: 23,110,160B/s
1333 Words per second performance: 1,959,516W/s
1334 Input File with a list of TEXTual Files: Leprechaun_vs_wikipedia_LATIN-WORDS.lst
1335 Size of all TEXTual Files: 415,982,896
1336 Word count: 35,271,297 of them 22,202,980 distinct
1337 Number Of Files: 8
1338 Number Of Lines: 35271297
1339 Allocated memory in MB: 1950
1340 Number Of Trees(GREATER THE BETTER): 3419429
1341 Forest population(Hash Function Quality regarding Collisions i.e. Hash Table Utilization): 51%
1342 Number Of Hash Collisions(Distinct WORDs - Number Of Trees): 18783551
1343 Maximum Attempts to Find/Put a WORD into a Binary-Search-Tree: '1,119'
1344 Total Attempts to Find/Put WORDs into Binary-Search-Trees: 268,085,505
1345 Total Number of LEAFs in Binary-Search-Trees(GREATER THE BETTER): 7,690,615
1346 Perfectly-Balanced-Binary-Search-Tree for MaxNODEs = 2,622 must have PEAK = 12 = rounding down of integer (1+lb(2,622))
1347 Binary-Search-Tree(1st out of 1) with MaxNODEs = 2,622 has PEAK = 592 and LEAFs = 689
1348 Binary-Search-Tree(1st out of 1) with MaxPEAK = '1,119' has NODEs = 1,537 and LEAFs = 287
1349 Binary-Search-Tree(1st out of 1) with MaxLEAFs = 731 has NODEs = 2,517 and PEAK = 448
1350 */
1351 // Invoking: FNV1A_Hash_8_OCTETS(wrd, wrdlen>>3) // = 0,1,2,3 [1..31]
1352 int FNV1A_Hash_8_OCTETS(char *str, int wrdlen_OCTETS)
1353 {
1354     u_int32_t hash;
1355     char *p;
1356
1357     hash = FNV1_32_INIT;
1358     p=str;
1359
1360     // The goal of stage #1: to reduce number of 'imul's.
1361
1362     // Stage #1:
1363     for (; wrdlen_OCTETS != 0; --wrdlen_OCTETS) {
1364         hash = FNV_32A_OP64(hash, (unsigned long long)*(long *)p); // mov edi, DWORD PTR [eax]
1365         p=p+8; // add eax, 4
1366     }
1367
1368     // Stage #2:
1369     for (; *p; ++p) {
1370         hash = FNV_32A_OP(hash, *p); // mov dl, BYTE PTR [ecx]
1371     }
1372
1373     //return ((hash>>13) ^ hash) & 8191; // (((u_int32_t)1<<(x))-1) where x=13
1374     return ((hash>>16) ^ hash) & 8191; // 00..8191 i.e. 2^13=8192
1375 }
1376
1377
1378 // Invoking: FNV1A_Hash_Granularity(wrd, wrdlen>>0|2|3, 0|2|3)
1379 int FNV1A_Hash_Granularity(char *str, int wrdlen_granulated, int Granularity) // wrdlen>>0=wrdlen
1380 {
1381     u_int32_t hash;
1382     u_int64_t hash64;
1383     char *p;
1384
1385     hash = FNV1_32_INIT;
1386     p=str;
1387
1388     // The goal of stage #1: to reduce number of 'imul's and mainly: the number of loops.
1389
1390     // Stage #1:
1391     if (Granularity == 2) {
1392     for (; wrdlen_granulated != 0; --wrdlen_granulated) {
1393         hash = FNV_32A_OP32(hash, (u_int32_t)*(u_int32_t *)p);
1394         p=p+4; // (1<<Granularity): 1<<0=1, 1<<2=4, 1<<3=8
1395     }
1396     }
1397     if (Granularity == 3) {
1398     hash64 = FNV1_64_INIT;
1399     for (; wrdlen_granulated != 0; --wrdlen_granulated) {

```

```

1400     hash64 = FNV_64A_OP64(hash64, (u_int64_t)*(u_int64_t *)p);
1401     p=p+8; // (1<<Granularity): 1<<0=1, 1<<2=4, 1<<3=8
1402 }
1403 for (; *p; ++p) {
1404     hash64 = FNV_64A_OP(hash64, (u_int8_t)*(u_int8_t *)p);
1405 }
1406
1407 //return ((hash>>13) ^ hash) & 8191; // (((u_int32_t)1<<(x))-1) where x=13
1408 return ((hash64>>51) ^ hash64) & 8191; // 00..8191 i.e. 2^13=8192
1409 // probably better shifting is not by 16 bits but ...
1410 //hash64>>16: 3,544,160 just bad
1411 //hash64>>33: 3,547,854
1412 //hash64>>34: 3,547,266
1413 //hash64>>35: 3,547,453
1414 //hash64>>36: 3,547,242
1415 //hash64>>40: 3,548,263
1416 //hash64>>44: 3,548,242
1417 //hash64>>45: 3,549,056
1418 //hash64>>46: 3,549,207
1419 //hash64>>47: 3,549,094
1420 //hash64>>50: 3,549,392
1421 //hash64>>51: 3,549,395 i.e. maximum shift: the 13 most significant bits i.e. (64-13); closest to 3,549,448
1422
1423 // Above results are obtained for following set:
1424 //if (wrklen<=19) // 4x4+3=19 i.e. last contains 7 clashes
1425 //    slot = FNV1A_Hash_Granularity(wrd, wrklen>>2, 2)<<2; //13++++
1426 //else // 2x8+4=20 i.e. first contains 6 clashes
1427 //    slot = FNV1A_Hash_Granularity(wrd, wrklen>>3, 3)<<2; //13++++
1428 // }
1429
1430 //if (Granularity != 3) {
1431 // Stage #2:
1432 for (; *p; ++p) {
1433     hash = FNV_32A_OP(hash, (u_int8_t)*(u_int8_t *)p);
1434 }
1435
1436 //return ((hash>>13) ^ hash) & 8191; // (((u_int32_t)1<<(x))-1) where x=13
1437 return ((hash>>16) ^ hash) & 8191; // 00..8191 i.e. 2^13=8192
1438 //}
1439 }
1440
1441
1442 // char *string; /* the string to 64 bit FNV-1a hash */
1443 // u_int64_t hash; /* will hold the final value of the hash */
1444 // char *p;
1445 //
1446 // hash = FNV1_64_INIT;
1447 // for (p=string; *p; ++p) {
1448 //     hash = FNV_64A_OP(hash, *p);
1449 // }
1450
1451
1452 // If you need an x-bit hash where x is not a power of 2,
1453 // then we recommend that you compute the FNV hash that is just larger than x-bits and xor-fold the result down to x-bits.
1454 // By xor-folding we mean shift the excess high order bits down and xor them with the lower x-bits.
1455 // For tiny x < 16 bit values, we recommend using a 32 bit FNV-1 hash as follows:
1456
1457 // /* NOTE: for 0 < x < 16 ONLY!!! */
1458 // #define TINY_MASK(x) (((u_int32_t)1<<(x))-1)
1459 // #define FNV1_32_INIT ((u_int32_t)2166136261)
1460 // u_int32_t hash;
1461 // void *data;
1462 // size_t data_len;
1463 //
1464 // hash = fnv_32_buf(data, data_len, FNV1_32_INIT);
1465 // hash = (((hash>>x) ^ hash) & TINY_MASK(x));
1466
1467
1468 int FNV1A_Hash_SHIFTless_XORless(char *str)
1469 {
1470     u_int32_t hash; /* will hold the final value of the hash */
1471     char *p;
1472
1473     hash = FNV1_32_INIT;
1474     for (p=str; *p; ++p) {
1475         hash = FNV_32A_OP(hash, *p);
1476     }
1477     //hash = ((hash>>13) ^ hash) & 8191; // (((u_int32_t)1<<(x))-1) where x=13
1478
1479     return hash & 8191; // 00..8191 i.e. 2^13=8192
1480 }
1481
1482 /*
1483 _FNV1A_Hash_SHIFTless_XORless PROC NEAR
1484 ; Line 721
1485 mov edx, DWORD PTR _str$[esp-4]
1486 mov cl, BYTE PTR [edx]
1487 test cl, cl

```

```

1488 mov eax, -2128831035 ; 811c9dc5H
1489 je SHORT $L1582
1490 npad1
1491 $L1580:
1492 ; Line 722
1493 movzx ecx, cl
1494 xor ecx, eax
1495 imul ecx, 16777619 ; 01000193H
1496 inc edx
1497 mov eax, ecx
1498 mov cl, BYTE PTR [edx]
1499 test cl, cl
1500 jne SHORT $L1580
1501 $L1582:
1502 ; Line 726
1503 and eax, 8191 ; 00001ffffH
1504 ; Line 727
1505 ret 0
1506 _FNV1A_Hash_SHIFTless_XORless ENDP
1507 */
1508
1509
1510 int FNV1A_Hash(char *str)
1511 {
1512 u_int32_t hash; /* will hold the final value of the hash */
1513 char *p;
1514
1515 hash = FNV1_32_INIT;
1516 for (p=str; *p; ++p) {
1517 hash = FNV_32A_OP(hash, *p);
1518 }
1519 //hash = ((hash>>13) ^ hash) & 8191; // (((u_int32_t)1<<(x))-1) where x=13
1520
1521 return ((hash>>13) ^ hash) & 8191; // 00..8191 i.e. 2^13=8192
1522 }
1523
1524 /*
1525 _FNV1A_Hash PROC NEAR
1526 ; Line 722
1527 mov edx, DWORD PTR _str$[esp-4]
1528 mov al, BYTE PTR [edx]
1529 test al, al
1530 mov ecx, -2128831035 ; 811c9dc5H
1531 je SHORT $L1582
1532 npad1
1533 $L1580:
1534 ; Line 723
1535 movzx eax, al
1536 xor eax, ecx
1537 imul eax, 16777619 ; 01000193H
1538 inc edx
1539 mov ecx, eax
1540 mov al, BYTE PTR [edx]
1541 test al, al
1542 jne SHORT $L1580
1543 $L1582:
1544 ; Line 727
1545 mov ecx, ecx
1546 shr eax, 13 ; 0000000dH
1547 xor eax, ecx
1548 and eax, 8191 ; 00001ffffH
1549 ; Line 728
1550 ret 0
1551 _FNV1A_Hash ENDP
1552 */
1553
1554 /*
1555 Wayne Diamond implemented 32-bit FNV algorithm in PowerBASIC inline x86 assembly:
1556
1557
1558 FUNCTION FNV32(BYVAL dwOffset AS DWORD, BYVAL dwLen AS DWORD, BYVAL offset_basis AS DWORD) AS DWORD
1559 #REGISTER NONE
1560 ! mov esi, dwOffset ;esi = ptr to buffer
1561 ! mov ecx, dwLen ;ecx = length of buffer (counter)
1562 ! mov eax, offset_basis ;set to 2166136261 for FNV-1
1563 ! mov edi, &h01000193 ;FNV_32_PRIME = 16777619
1564 ! xor ebx, ebx ;ebx = 0
1565 nextbyte:
1566 ! mul edi ;eax = eax * FNV_32_PRIME
1567 ! mov bl, [esi] ;bl = byte from esi
1568 ! xor eax, ebx ;al = al xor bl
1569 ! inc esi ;esi = esi + 1 (buffer pos)
1570 ! dec ecx ;ecx = ecx - 1 (counter)
1571 ! jnz nextbyte ;if ecx is 0, jmp to NextByte
1572 ! mov FUNCTION, eax ;else, function = eax
1573 END FUNCTION
1574
1575 Wayne said:

```

```

1576
1577 ''Just thought I should let you know that I've ported the 32-bit FNV algorithm over to inline assembly.
1578 It's actually in PowerBASIC (www.powerbasic.com) format - a compiler I use, but the main function is all assembly.
1579 It could be optimized further in terms of saving a couple of clock cycles,
1580 but it's fairly optimized al ready - only 6 instructions in the main loop, plus 5 setup instructions,
1581 and compiles to just 33 bytes.''
1582
1583 M.S.Schulte sent us these 32-bit FNV-1 and FNV-1a x86 assembler implementations (written in flat assembler),
1584 half of which were optimized for speed, the other half were optimized for size:
1585
1586 small_fnv32: ;FNV1 32bit (size: 31 bytes)
1587 ; Intel Core 2 Duo E6600: 354.20 mb/s
1588     push    esi
1589     push    edi
1590     mov     esi, [esp + 0ch] ;buffer
1591     mov     ecx, [esp + 10h] ;length
1592     mov     eax, [esp + 14h] ;basis
1593     mov     edi, 01000193h ;fnv_32_prime
1594 next:
1595     mul     edi
1596     xor     al, [esi]
1597     inc     esi
1598     loop    snext
1599     pop     edi
1600     pop     esi
1601     retn    0ch
1602
1603 small_fnv32a: ;FNV1a 32bit (size: 31 bytes)
1604 ; Intel Core 2 Duo E6600: 327.68 mb/s
1605     push    esi
1606     push    edi
1607     mov     esi, [esp + 0ch] ;buffer
1608     mov     ecx, [esp + 10h] ;length
1609     mov     eax, [esp + 14h] ;basis
1610     mov     edi, 01000193h ;fnv_32_prime
1611 nexta:
1612     xor     al, [esi]
1613     mul     edi
1614     inc     esi
1615     loop    nexta
1616     pop     edi
1617     pop     esi
1618     retn    0ch
1619
1620 fast_fnv32: ;FNV1 32bit (size: 36 bytes)
1621 ; Intel Core 2 Duo E6600: 565.12 mb/s
1622     push    ebx
1623     push    esi
1624     push    edi
1625     mov     esi, [esp + 10h] ;buffer
1626     mov     ecx, [esp + 14h] ;length
1627     mov     eax, [esp + 18h] ;basis
1628     mov     edi, 01000193h ;fnv_32_prime
1629     xor     ebx, ebx
1630 next:
1631     mul     edi
1632     mov     bl, [esi]
1633     xor     eax, ebx
1634     inc     esi
1635     dec     ecx
1636     jnz     next
1637     pop     edi
1638     pop     esi
1639     pop     ebx
1640     retn    0ch
1641
1642 fast_fnv32a: ;FNV1a 32bit (size: 36 bytes)
1643 ; Intel Core 2 Duo E6600: 574.95 mb/s
1644     push    ebx
1645     push    esi
1646     push    edi
1647     mov     esi, [esp + 10h] ;buffer
1648     mov     ecx, [esp + 14h] ;length
1649     mov     eax, [esp + 18h] ;basis
1650     mov     edi, 01000193h ;fnv_32_prime
1651     xor     ebx, ebx
1652 nexta:
1653     mov     bl, [esi]
1654     xor     eax, ebx
1655     mul     edi
1656     inc     esi
1657     dec     ecx
1658     jnz     nexta
1659     pop     edi
1660     pop     esi
1661     pop     ebx
1662     retn    0ch
1663 */

```

```

1664
1665 //Number Of Trees(GREATER THE BETTER): 3525737
1666 //Forest population(Hash Function Quality regarding Collisions i.e. Hash Table Utilization): 53%
1667 //Number Of Hash Collisions(Distinct WORDS - Number Of Trees): 18677243
1668 int Hash17_unrolled(const char *key, int wrdlen)
1669 {
1670     int hash = 1;
1671     int i;
1672     for(i = 0; i < (wrdlen & -2); i += 2) {
1673         hash = (17) * hash + (key[i] - ' ');
1674         hash = (17) * hash + (key[i+1] - ' ');
1675     }
1676     if(wrdlen & 1)
1677         hash = (17) * hash + (key[wrdlen-1] - ' ');
1678     return ( hash ^ (hash >> 16) ) & 8191;
1679 }
1680
1681 //hash = 1:
1682 //Number Of Trees(GREATER THE BETTER): 3556516
1683 //Forest population(Hash Function Quality regarding Collisions i.e. Hash Table Utilization): 53%
1684 //Number Of Hash Collisions(Distinct WORDS - Number Of Trees): 18646464
1685 //hash = 13:
1686 //Number Of Trees(GREATER THE BETTER): 3556755
1687 //Forest population(Hash Function Quality regarding Collisions i.e. Hash Table Utilization): 53%
1688 //Number Of Hash Collisions(Distinct WORDS - Number Of Trees): 18646225
1689 //hash = 11:
1690 //Number Of Trees(GREATER THE BETTER): 3557011
1691 //Forest population(Hash Function Quality regarding Collisions i.e. Hash Table Utilization): 53%
1692 //Number Of Hash Collisions(Distinct WORDS - Number Of Trees): 18645969
1693 //hash = 7:
1694 //Number Of Trees(GREATER THE BETTER): 3557181
1695 //Forest population(Hash Function Quality regarding Collisions i.e. Hash Table Utilization): 53%
1696 //Number Of Hash Collisions(Distinct WORDS - Number Of Trees): 18645799
1697 int Alfalfa(const char *key, int wrdlen)
1698 {
1699     int hash = 7;
1700     int i;
1701     for(i = 0; i < (wrdlen & -2); i += 2) {
1702         hash = (17+9) * ((17+9) * hash + (key[i])) + (key[i+1]);
1703     }
1704     if(wrdlen & 1)
1705         hash = (17+9) * hash + (key[wrdlen-1]);
1706     return ( hash ^ (hash >> 16) ) & 8191;
1707 }
1708
1709 /*
1710 [FNV1A 'shift-less-&-xor-less' hash used in Leprechaun r.13+++:]
1711
1712 int FNV1A_Hash_SHIFTless_XORless(char *str)
1713 {
1714     u_int32_t hash;
1715     char *p;
1716
1717     hash = FNV1_32_INIT;
1718     for (p=str; *p; ++p) {
1719         hash = FNV_32A_OP(hash, *p);
1720     }
1721     //hash = ((hash>>13) ^ hash) & 8191; // (((u_int32_t)1<<(x))-1) where x=13
1722
1723     return hash & 8191; // 00..8191 i.e. 2^13=8192
1724 }
1725
1726 words per second performance: 837,458w/s
1727 Input File with a list of TEXTual Files: wikipedia-en-html.tar.wrd.lst
1728 word count: 12,561,874 of them 12,561,874 distinct
1729 Number Of Trees(GREATER THE BETTER): 2772875
1730 Forest population(Hash Function Quality regarding Collisions i.e. Hash Table Utilization): 41%
1731 Number Of Hash Collisions(Distinct WORDS - Number Of Trees): 9788999
1732
1733 words per second performance: 1,007,751w/s
1734 Input File with a list of TEXTual Files: Leprechaun_vs_wikipedia_LATIN-WORDS.lst
1735 word count: 35,271,297 of them 22,202,980 distinct
1736 Number Of Trees(GREATER THE BETTER): 3537061
1737 Forest population(Hash Function Quality regarding Collisions i.e. Hash Table Utilization): 53%
1738 Number Of Hash Collisions(Distinct WORDS - Number Of Trees): 18665919
1739
1740 [My '2in1' hash used in Leprechaun r.13++:]
1741
1742 int KuxHash3plus(char *str)
1743 { int h = 0;
1744   unsigned long h2 = 0; // must be long: 31*'z'=31*122
1745   int max31 = 0;
1746   while (str[max31])
1747       { h = h ^ str[max31]; // 00..255 i.e. 2^8=256
1748         //h2 = h2 + str[max31++]; // [113s]
1749         h2 = h2 + str[max31++] * (max31+1);
1750     }
1751     // Result is: 7bits in 'h' and 32bits in 'h2'.

```

```

1752
1753 //printf("%s:\n",str);
1754 //printf("%d",h);
1755 // a in ASCII is 097 = 0110 0001
1756 // z in ASCII is 122 = 0111 1010
1757 // Above two lines show that bits 8-7-6 are always 0-1-1 so need for low 5 bits.
1758 //h=h<<8; // 00..15 i.e. 5bits + 00-07bits=13bits
1759 //printf("%d",h);
1760 //printf("%d",h2);
1761 //h = h|( str[0] ^ str[max31-1] ); // [111s] a..z: each XOR each gives 00..31
1762 h = (( h<<8 )|( h2%(251) ))&8191; // 251 prime
1763 //printf("%d \n",h);
1764 return h; // 00..8191 i.e. 2^13=8192
1765 }
1766
1767 words per second performance: 785,117w/s
1768 Input File with a list of TEXTual Files: wikipedia-en-html.tar.wrd.lst
1769 word count: 12,561,874 of them 12,561,874 distinct
1770 Number Of Trees(GREATER THE BETTER): 2663566
1771 Forest population(Hash Function Quality regarding Collisions i.e. Hash Table Utilization): 40%
1772 Number Of Hash Collisions(Distinct WORDs - Number Of Trees): 9898308
1773
1774 words per second performance: 979,758w/s
1775 Input File with a list of TEXTual Files: Leprechaun_vs_Wikipedia_LATIN-WORDS.lst
1776 word count: 35,271,297 of them 22,202,980 distinct
1777 Number Of Trees(GREATER THE BETTER): 3410463
1778 Forest population(Hash Function Quality regarding Collisions i.e. Hash Table Utilization): 51%
1779 Number Of Hash Collisions(Distinct WORDs - Number Of Trees): 18792517
1780
1781 [Last standing for English(en)-Wikipedia's wordlist:]
1782 chongo's hash is faster(in total, not the function itself) than Kaze's hash by ((837,458w/s - 785,117w/s)/785,117w/s)*100% = 6.6%
1783 chongo's hash has better distribution than Kaze's hash by ((9898308 - 9788999)/9788999)*100% = 1.1%
1784
1785 [Last standing for LATIN(de,en,es,fr,it,nl,pt,ro)-Wikipedia's wordlist:]
1786 chongo's hash is faster(in total, not the function itself) than Kaze's hash by ((1,007,751w/s - 979,758w/s)/979,758w/s)*100% = 2.8%
1787 chongo's hash has better distribution than Kaze's hash by ((18792517 - 18665919)/18665919)*100% = 0.6%
1788
1789 Bottomline is:
1790 Your hash thrash, my hash for trash, he-he.
1791 Thanks a lot, again, Mr. Noll.
1792
1793 Yummy little file: http://www.isthe.com/chongo/src/fnv/fnv-5.0.2.tar.gz
1794 */
1795
1796 /*
1797 // Paul Larson (http://research.microsoft.com/~PALARSON/)
1798 UINT HashLarson(const CHAR *key, SIZE_T len) {
1799     UINT hash = 0;
1800     for(UINT i = 0; i < len; ++i)
1801         hash = 101 * hash + key[i];
1802     return hash ^ (hash >> 16);
1803 }
1804
1805 // Kernighan & Ritchie, "The C programming Language", 3rd edition.
1806 UINT HashKernighanRitchie(const CHAR *key, SIZE_T len) {
1807     UINT hash = 0;
1808     for(UINT i = 0; i < len; ++i)
1809         hash = 31 * hash + key[i];
1810     return hash;
1811 }
1812
1813 // A hash function with multiplier 65599 (from Red Dragon book)
1814 UINT Hash65599(const CHAR *key, SIZE_T len) {
1815     UINT hash = 0;
1816     for(UINT i = 0; i < len; ++i)
1817         hash = 65599 * hash + key[i];
1818     return hash ^ (hash >> 16);
1819 }
1820
1821 // FNV hash, http://isthe.com/chongo/tech/comp/fnv/
1822 UINT HashFNV1a(const CHAR *key, SIZE_T len) {
1823     UINT hash = 2166136261;
1824     for(UINT i = 0; i < len; ++i)
1825         hash = 16777619 * (hash ^ key[i]);
1826     return hash ^ (hash >> 16);
1827 }
1828
1829 // Ramakrishna hash
1830 UINT HashRamakrishna(const CHAR *key, SIZE_T len) {
1831     UINT h = 0;
1832     for(UINT i = 0; i < len; ++i) {
1833         h ^= (h << 5) + (h >> 2) + key[i];
1834     }
1835     return h;
1836 }
1837 */
1838
1839 /*

```

```

1840 Results for 'HashAlfalfa':
1841 Bytes per second performance: 19,808,709B/s
1842 Words per second performance: 1,679,585W/s
1843 Input File with a list of TEXTual Files: Leprechaun_vs_wikipedia_LATIN-WORDS.lst
1844 Size of all TEXTual Files: 415,982,896
1845 Word count: 35,271,297 of them 22,202,980 distinct
1846 Number Of Files: 8
1847 Number Of Lines: 35271297
1848 Allocated memory in MB: 1950
1849 Number Of Trees(GREATER THE BETTER): 3549079
1850 Forest population(Hash Function Quality regarding Collisions i.e. Hash Table Utilization): 53%
1851 Number of Hash Collisions(Distinct WORDs - Number Of Trees): 18653901
1852 Maximum Attempts to Find/Put a WORD into a Binary-Search-Tree: '37'
1853 Total Attempts to Find/Put WORDs into Binary-Search-Trees: 117,063,824
1854 Total Number of LEAFs in Binary-Search-Trees(GREATER THE BETTER): 8,072,279
1855 Perfectly-Balanced-Binary-Search-Tree for MaxNODEs = 84 must have PEAK = 7 = rounding down of integer (1+lb(84))
1856 Binary-Search-Tree(1st out of 2) with MaxNODEs = 84 has PEAK = 20 and LEAFs = 24
1857 Binary-Search-Tree(1st out of 3) with MaxPEAK = '37' has NODEs = 67 and LEAFs = 17
1858 Binary-Search-Tree(1st out of 1) with MaxLEAFs = 28 has NODEs = 78 and PEAK = 22
1859 */
1860 UINT HashAlfalfa(const char *key, unsigned int wrdlen)
1861 {
1862     UINT hash = 7;
1863     unsigned int i;
1864     for (i = 0; i < (wrdlen & -2); i += 2) {
1865         hash = (53) * ((53) * hash + (key[i])) + (key[i+1]);
1866     }
1867     if (wrdlen & 1)
1868         hash = (53) * hash + (key[wrdlen-1]);
1869     return ((hash>>16) ^ hash) & 8191;
1870 }
1871
1872 /*
1873 Results for 'HashAlfalfa_HALF':
1874 Bytes per second performance: 19,808,709B/s
1875 Words per second performance: 1,679,585W/s
1876 Input File with a list of TEXTual Files: Leprechaun_vs_wikipedia_LATIN-WORDS.lst
1877 Size of all TEXTual Files: 415,982,896
1878 Word count: 35,271,297 of them 22,202,980 distinct
1879 Number Of Files: 8
1880 Number Of Lines: 35271297
1881 Allocated memory in MB: 1950
1882 Number Of Trees(GREATER THE BETTER): 3550665
1883 Forest population(Hash Function Quality regarding Collisions i.e. Hash Table Utilization): 53%
1884 Number of Hash Collisions(Distinct WORDs - Number Of Trees): 18652315
1885 Maximum Attempts to Find/Put a WORD into a Binary-Search-Tree: '39'
1886 Total Attempts to Find/Put WORDs into Binary-Search-Trees: 117,053,918
1887 Total Number of LEAFs in Binary-Search-Trees(GREATER THE BETTER): 8,072,259
1888 Perfectly-Balanced-Binary-Search-Tree for MaxNODEs = 87 must have PEAK = 7 = rounding down of integer (1+lb(87))
1889 Binary-Search-Tree(1st out of 1) with MaxNODEs = 87 has PEAK = 21 and LEAFs = 27
1890 Binary-Search-Tree(1st out of 2) with MaxPEAK = '39' has NODEs = 65 and LEAFs = 18
1891 Binary-Search-Tree(1st out of 4) with MaxLEAFs = 27 has NODEs = 77 and PEAK = 23
1892 */
1893 UINT HashAlfalfa_HALF(const char *key, unsigned int wrdlen)
1894 {
1895     UINT hash = 12;
1896     UINT hashBUFFER;
1897     unsigned int i,j;
1898     for(i = 0; i < (wrdlen & -4); i += 4) {
1899         //hash = (( ((hash<<5)-hash) + key[i] )<<5) - ( ((hash<<5)-hash) + key[i] ) + (key[i+1]);
1900         hashBUFFER = ((hash<<5)-hash) + key[i];
1901         hash = (( hashBUFFER )<<5) - ( hashBUFFER ) + (key[i+1]);
1902         //hash = (( ((hash<<5)-hash) + key[i+2] )<<5) - ( ((hash<<5)-hash) + key[i+2] ) + (key[i+3]);
1903         hashBUFFER = ((hash<<5)-hash) + key[i+2];
1904         hash = (( hashBUFFER )<<5) - ( hashBUFFER ) + (key[i+3]);
1905     }
1906     for(j = 0; j < (wrdlen & 3); j += 1) {
1907         hash = ((hash<<5)-hash) + key[i+j];
1908     }
1909     return ((hash>>16) ^ hash) & 8191;
1910 }
1911
1912 /*
1913 Results for 'HashFNv1A_unrolled_Final':
1914 Bytes per second performance: 19,808,709B/s
1915 Words per second performance: 1,679,585W/s
1916 Input File with a list of TEXTual Files: Leprechaun_vs_wikipedia_LATIN-WORDS.lst
1917 Size of all TEXTual Files: 415,982,896
1918 Word count: 35,271,297 of them 22,202,980 distinct
1919 Number Of Files: 8
1920 Number Of Lines: 35271297
1921 Allocated memory in MB: 1950
1922 Number Of Trees(GREATER THE BETTER): 3445337
1923 Forest population(Hash Function Quality regarding Collisions i.e. Hash Table Utilization): 52%
1924 Number of Hash Collisions(Distinct WORDs - Number Of Trees): 18757643
1925 Maximum Attempts to Find/Put a WORD into a Binary-Search-Tree: '43'
1926 Total Attempts to Find/Put WORDs into Binary-Search-Trees: 118,349,998
1927 Total Number of LEAFs in Binary-Search-Trees(GREATER THE BETTER): 7,997,033

```

```

1928 Perfectly-Balanced-Binary-Search-Tree for MaxNODES = 89 must have PEAK = 7 = rounding down of integer (1+lb(89))
1929 Binary-Search-Tree(1st out of 1) with MaxNODES = 89 has PEAK = 28 and LEAFs = 28
1930 Binary-Search-Tree(1st out of 1) with MaxPEAK = '43' has NODES = 65 and LEAFs = 11
1931 Binary-Search-Tree(1st out of 2) with MaxLEAFs = 28 has NODES = 78 and PEAK = 24
1932 */
1933 UINT HashFNV1A_unrolled_Final(char *str, unsigned int wrdlen)
1934 {
1935     //const UINT PRIME = 31;
1936     unsigned int hash = 2166136261;
1937     char * p = str;
1938
1939     /*
1940     // Reduce the number of multiplications by unrolling the loop
1941     for (SIZE_T ndwords = wrdlen / sizeof(DWORD); ndwords; --ndwords) {
1942         //hash = (hash ^ *(DWORD*)p) * PRIME;
1943         hash = ((hash ^ *(DWORD*)p)<<5) - (hash ^ *(DWORD*)p);
1944
1945         p += sizeof(DWORD);
1946     }
1947     */
1948     for(; wrdlen >= 4; wrdlen -= 4, p += 4) {
1949         hash = ((hash ^ *(unsigned int*)p)<<5) - (hash ^ *(unsigned int*)p);
1950     }
1951
1952     // Process the remaining bytes
1953     /*
1954     for (SIZE_T i = 0; i < (wrdlen & (sizeof(DWORD) - 1)); i++) {
1955         //hash = (hash ^ *p++) * PRIME;
1956         hash = ((hash ^ *p)<<5) - (hash ^ *p);
1957         p++;
1958     }
1959     */
1960     if (wrdlen & -2) {
1961         hash = ((hash ^ *(unsigned int*)p&0xFFFF)<<5) - (hash ^ *(unsigned int*)p&0xFFFF);
1962         p++;p++;
1963     }
1964     if (wrdlen & 1)
1965         hash = ((hash ^ *p)<<5) - (hash ^ *p);
1966
1967     return ((hash>>16) ^ hash) & 8191;
1968 }
1969
1970 /*
1971 Results for 'Sixtinsensitive':
1972 Bytes per second performance: 19,808,709B/s
1973 Words per second performance: 1,679,585W/s
1974 Input File with a list of TEXTual Files: Leprechaun_vs_wikipedia_LATIN-WORDS.lst
1975 Size of all TEXTual Files: 415,982,896
1976 word count: 35,271,297 of them 22,202,980 distinct
1977 Number of Files: 8
1978 Number of Lines: 35271297
1979 Allocated memory in MB: 1950
1980 Number of Trees(GREATER THE BETTER): 3531949
1981 Forest population(Hash Function Quality regarding Collisions i.e. Hash Table Utilization): 53%
1982 Number of Hash Collisions(Distinct WORDs - Number of Trees): 18671031
1983 Maximum Attempts to Find/Put a WORD into a Binary-Search-Tree: '38'
1984 Total Attempts to Find/Put WORDs into Binary-Search-Trees: 118,959,016
1985 Total Number of LEAFs in Binary-Search-Trees(GREATER THE BETTER): 8,047,983
1986 Perfectly-Balanced-Binary-Search-Tree for MaxNODES = 98 must have PEAK = 7 = rounding down of integer (1+lb(98))
1987 Binary-Search-Tree(1st out of 1) with MaxNODES = 98 has PEAK = 36 and LEAFs = 30
1988 Binary-Search-Tree(1st out of 1) with MaxPEAK = '38' has NODES = 54 and LEAFs = 11
1989 Binary-Search-Tree(1st out of 2) with MaxLEAFs = 30 has NODES = 98 and PEAK = 36
1990 */
1991 // Tuned for lowercase-and-uppercase letters i.e. 26 ASCII symbols 65-90 and 97-122 decimal.
1992 UINT Sixtinsensitive(const char *str, unsigned int wrdlen)
1993 {
1994     UINT hash = 2166136261;
1995     UINT hashBUFFER_EAX, hashBUFFER_BH, hashBUFFER_BL;
1996     const char * p = str;
1997
1998     // 0x41 = 065 'A' 010 [0 0001]
1999     // 0x5A = 090 'Z' 010 [1 1010]
2000     // 0x61 = 097 'a' 011 [0 0001]
2001     // 0x7A = 122 'z' 011 [1 1010]
2002
2003     // Reduce the number of multiplications by unrolling the loop
2004     for(; wrdlen >= 6; wrdlen -= 6, p += 6) {
2005         //hashBUFFER_AX = (*(DWORD*)(p+0)&0xFFFF);
2006         hashBUFFER_EAX = (*(DWORD*)(p+0)&0x1F1F1F1F);
2007         hashBUFFER_BL = (*(p+4)&0x1F);
2008         hashBUFFER_BH = (*(p+5)&0x1F);
2009         //6bytes-in-4bytes or 48bits-to-30bits
2010         // Two times next:
2011         //3bytes-in-2bytes or 24bits-to-15bits
2012         //EAX BL BH
2013         //[5bit][3bit][5bit][3bit][5bit][3bit][5bit][3bit]
2014         // 5th[0..15] 13th[0..15]
2015         // BL lower 3 BL higher 2bits

```



```

2016 // OR or XOR no difference
2017 hashBUFFER_EAX = hashBUFFER_EAX ^ ((hashBUFFER_BL&0x07)<<5); // BL lower 3bits of 5bits
2018 hashBUFFER_EAX = hashBUFFER_EAX ^ ((hashBUFFER_BL&0x18)<<(2+8)); // BL higher 2bits of 5bits
2019 hashBUFFER_EAX = hashBUFFER_EAX ^ ((hashBUFFER_BH&0x07)<<(5+16)); // BH lower 3bits of 5bits
2020 hashBUFFER_EAX = hashBUFFER_EAX ^ ((hashBUFFER_BH&0x18)<<((2+8)+16)); // BH higher 2bits of 5bits
2021 //hash = (hash ^ hashBUFFER_EAX)*1607; //what a mess: <<7 becomes imul but <<5 not!?
2022 hash = ((hash ^ hashBUFFER_EAX)<<5) - (hash ^ hashBUFFER_EAX);
2023 //1607:[2118599]
2024 // 127:[2121081]
2025 // 31:[2139242]
2026 // 17:[2150803]
2027 // 7:[2166336]
2028 // 5:[2183044]
2029 //8191:[2200477]
2030 // 3:[2205095]
2031 // 257:[2206188]
2032 }
2033 // Post-Variant #1:
2034 for(; wrdlen; wrdlen--, p++) {
2035     hash = ((hash ^ (*p&0x1F))<<5) - (hash ^ (*p&0x1F));
2036 }
2037 /*
2038 // Post-Variant #2:
2039 for(; wrdlen >= 2; wrdlen -= 2, p += 2) {
2040     hash = ((hash ^ (*(DWORD*)p&0xFFFF))<<5) - (hash ^ (*(DWORD*)p&0xFFFF));
2041 }
2042 if (wrdlen & 1)
2043     hash = ((hash ^ *p)<<5) - (hash ^ *p);
2044 */
2045 /*
2046 // Post-Variant #3:
2047 for(; wrdlen >= 4; wrdlen -= 4, p += 4) {
2048     hash = ((hash ^ *(DWORD*)p)<<5) - (hash ^ *(DWORD*)p);
2049 }
2050 if (wrdlen & -2) {
2051     hash = ((hash ^ *(DWORD*)p&0xFFFF)<<5) - (hash ^ *(DWORD*)p&0xFFFF);
2052     p++;p++;
2053 }
2054 if (wrdlen & 1)
2055     hash = ((hash ^ *p)<<5) - (hash ^ *p);
2056 */
2057 return ((hash>>16) ^ hash) & 8191;
2058 }
2059
2060 /*
2061 #define FNV1_32_INIT ((UINT)2166136261)
2062 #define FNV1_32_PRIME ((UINT)1709)
2063
2064 #define FNV_32A_OP(hash, octet) \
2065     (((UINT)(hash) ^ (unsigned char)(octet)) * FNV1_32_PRIME)
2066
2067 #define FNV_32A_OP32(hash, octet) \
2068     (((UINT)(hash) ^ (UINT)(octet)) * FNV1_32_PRIME)
2069
2070 UINT FNV1A_Hash_WHIZ(const char *str, SIZE_T wrdlen)
2071 {
2072
2073     UINT hash32;
2074     const char *p;
2075
2076     hash32 = FNV1_32_INIT;
2077     p=str;
2078
2079     for(; wrdlen >= 4; wrdlen -= 4, p += 4) {
2080         hash32 = FNV_32A_OP32(hash32, (UINT)*(UINT *)p);
2081     }
2082     if (wrdlen & -2) {
2083         hash32 = FNV_32A_OP32(hash32, *(UINT*)p&0xFFFF);
2084         p++;p++;
2085     }
2086     if (wrdlen & 1)
2087         hash32 = FNV_32A_OP(hash32, *p);
2088
2089     return hash32 ^ (hash32 >> 16);
2090 }
2091 */
2092
2093 /*
2094 Results for 'FNV1A_Hash_Jester':
2095 Bytes per second performance: 19,808,709B/s
2096 Words per second performance: 1,679,585W/s
2097 Input File with a list of TEXTual Files: Leprechaun_vs_Wikipedia_LATIN-WORDS.lst
2098 Size of all TEXTual Files: 415,982,896
2099 word count: 35,271,297 of them 22,202,980 distinct
2100 Number Of Files: 8
2101 Number Of Lines: 35271297
2102 Allocated memory in MB: 1950
2103 Number Of Trees(GREATER THE BETTER): 3537352

```

```

2104 Forest population(Hash Function Quality regarding Collisions i.e. Hash Table Utilization): 53%
2105 Number Of Hash Collisions(Distinct WORDs - Number Of Trees): 18665628
2106 Maximum Attempts to Find/Put a WORD into a Binary-Search-Tree: '37'
2107 Total Attempts to Find/Put WORDs into Binary-Search-Trees: 117,243,563
2108 Total Number of LEAFs in Binary-Search-Trees(GREATER THE BETTER): 8,063,361
2109 Perfectly-Balanced-Binary-Search-Tree for MaxNODEs = 87 must have PEAK = 7 = rounding down of integer (1+lb(87))
2110 Binary-Search-Tree(1st out of 2) with MaxNODEs = 87 has PEAK = 27 and LEAFs = 23
2111 Binary-Search-Tree(1st out of 1) with MaxPEAK = '37' has NODEs = 66 and LEAFs = 18
2112 Binary-Search-Tree(1st out of 3) with MaxLEAFs = 27 has NODEs = 84 and PEAK = 27
2113 */
2114 UINT FNV1A_Hash_Jester(const char *str, unsigned int wrdlen)
2115 {
2116     const UINT PRIME = 709607;
2117     UINT hash32 = 2166136261;
2118     const char *p = str;
2119
2120     // Idea comes from Igor Pavlov's 7zCRC, thanks.
2121     /*
2122     for(; wrdlen && ((unsigned)(ptrdiff_t)p&3); wrdlen -= 1, p++) {
2123         hash32 = (hash32 ^ *p) * PRIME;
2124     }
2125     */
2126     for(; wrdlen >= 2*sizeof(DWORD); wrdlen -= 2*sizeof(DWORD), p += 2*sizeof(DWORD)) {
2127         hash32 = (hash32 ^ *(DWORD *)p) * PRIME;
2128         hash32 = (hash32 ^ *(DWORD *) (p+4)) * PRIME;
2129     }
2130     // Cases: 0,1,2,3,4,5,6,7
2131     if (wrdlen & sizeof(DWORD)) {
2132         hash32 = (hash32 ^ *(DWORD *)p) * PRIME;
2133         p += sizeof(DWORD);
2134     }
2135     if (wrdlen & sizeof(WORD)) {
2136         hash32 = (hash32 ^ *(WORD *)p) * PRIME;
2137         p += sizeof(WORD);
2138     }
2139     if (wrdlen & 1)
2140         hash32 = (hash32 ^ *p) * PRIME;
2141
2142     return (hash32 ^ (hash32 >> 16)) & 8191;
2143 }
2144
2145 /*
2146 Results for 'FNV1A_Hash_Jesteress':
2147 Bytes per second performance: 19,808,709B/s
2148 Words per second performance: 1,679,585W/s
2149 Input File with a list of TEXTual Files: Leprechaun_vs_Wikipedia_LATIN-WORDS.lst
2150 Size of all TEXTual Files: 415,982,896
2151 Word count: 35,271,297 of them 22,202,980 distinct
2152 Number Of Files: 8
2153 Number Of Lines: 35271297
2154 Allocated memory in MB: 1950
2155 Number Of Trees(GREATER THE BETTER): 3537293
2156 Forest population(Hash Function Quality regarding Collisions i.e. Hash Table Utilization): 53%
2157 Number Of Hash Collisions(Distinct WORDs - Number Of Trees): 18665687
2158 Maximum Attempts to Find/Put a WORD into a Binary-Search-Tree: '40'
2159 Total Attempts to Find/Put WORDs into Binary-Search-Trees: 117,526,680
2160 Total Number of LEAFs in Binary-Search-Trees(GREATER THE BETTER): 8,051,512
2161 Perfectly-Balanced-Binary-Search-Tree for MaxNODEs = 89 must have PEAK = 7 = rounding down of integer (1+lb(89))
2162 Binary-Search-Tree(1st out of 1) with MaxNODEs = 89 has PEAK = 25 and LEAFs = 23
2163 Binary-Search-Tree(1st out of 1) with MaxPEAK = '40' has NODEs = 49 and LEAFs = 8
2164 Binary-Search-Tree(1st out of 1) with MaxLEAFs = 28 has NODEs = 72 and PEAK = 21
2165 */
2166 #define ROL(x, n) (((x) << (n)) | ((x) >> (32-(n))))
2167 UINT FNV1A_Hash_Jesteress(const char *str, unsigned int wrdlen)
2168 {
2169     const UINT PRIME = 709607;
2170     UINT hash32 = 2166136261;
2171     const char *p = str;
2172
2173     // Idea comes from Igor Pavlov's 7zCRC, thanks.
2174     /*
2175     for(; wrdlen && ((unsigned)(ptrdiff_t)p&3); wrdlen -= 1, p++) {
2176         hash32 = (hash32 ^ *p) * PRIME;
2177     }
2178     */
2179     for(; wrdlen >= 2*sizeof(DWORD); wrdlen -= 2*sizeof(DWORD), p += 2*sizeof(DWORD)) {
2180         hash32 = (hash32 ^ (ROL(*(DWORD *)p, 5)^(DWORD *) (p+4))) * PRIME;
2181     }
2182     // Cases: 0,1,2,3,4,5,6,7
2183     if (wrdlen & sizeof(DWORD)) {
2184         hash32 = (hash32 ^ *(DWORD *)p) * PRIME;
2185         p += sizeof(DWORD);
2186     }
2187     if (wrdlen & sizeof(WORD)) {
2188         hash32 = (hash32 ^ *(WORD *)p) * PRIME;
2189         p += sizeof(WORD);
2190     }
2191     if (wrdlen & 1)

```

```

2192     hash32 = (hash32 ^ *p) * PRIME;
2193
2194 return (hash32 ^ (hash32 >> 16)) & 8191;
2195 }
2196
2197 UINT FNV1A_Hash_Jesteress_27bit(const char *str, unsigned int wrdlen)
2198 {
2199     const UINT PRIME = 709607;
2200     UINT hash32 = 2166136261;
2201     const char *p = str;
2202
2203     // Idea comes from Igor Pavlov's 7zCRC, thanks.
2204     /*
2205     for(; wrdlen && ((unsigned)(ptrdiff_t)p&3); wrdlen -= 1, p++) {
2206         hash32 = (hash32 ^ *p) * PRIME;
2207     }
2208     */
2209     for(; wrdlen >= 2*sizeof(DWORD); wrdlen -= 2*sizeof(DWORD), p += 2*sizeof(DWORD)) {
2210         hash32 = (hash32 ^ (ROL(*(DWORD *)p,5)^(DWORD *)p+4))) * PRIME;
2211     }
2212     // Cases: 0,1,2,3,4,5,6,7
2213     if (wrdlen & sizeof(DWORD)) {
2214         hash32 = (hash32 ^ *(DWORD*)p) * PRIME;
2215         p += sizeof(DWORD);
2216     }
2217     if (wrdlen & sizeof(WORD)) {
2218         hash32 = (hash32 ^ *(WORD*)p) * PRIME;
2219         p += sizeof(WORD);
2220     }
2221     if (wrdlen & 1)
2222         hash32 = (hash32 ^ *p) * PRIME;
2223
2224     return (hash32 ^ (hash32 >> 16)) & ((1<<27)-1);
2225 }
2226
2227 /*
2228 UINT NextPowerOfTwo(UINT x) {
2229     // Henry Warren, "Hacker's Delight", ch. 3.2
2230     x--;
2231     x |= (x >> 1);
2232     x |= (x >> 2);
2233     x |= (x >> 4);
2234     x |= (x >> 8);
2235     x |= (x >> 16);
2236     return x + 1;
2237 }
2238
2239 UINT NextLog2(UINT x) {
2240     // Henry Warren, "Hacker's Delight", ch. 5.3
2241     if(x <= 1) return x;
2242     x--;
2243     UINT n = 0;
2244     UINT y;
2245     y = x >> 16; if(y) {n += 16; x = y;}
2246     y = x >> 8; if(y) {n += 8; x = y;}
2247     y = x >> 4; if(y) {n += 4; x = y;}
2248     y = x >> 2; if(y) {n += 2; x = y;}
2249     y = x >> 1; if(y) return n + 2;
2250     return n + x;
2251 }
2252 */
2253
2254 // The following example code in the C language computes the binary logarithm (rounding down) of an integer, rounded down. [2] The operator
2255 // '>>' represents 'unsigned right shift'. The rounding down form of binary logarithm is identical to computing the position of the most
2256 // significant 1 bit.
2257 /**
2258  * Returns the floor form of binary logarithm for a 32 bit integer.
2259  * -1 is returned if n is 0.
2260  */
2261 int floorLog2(unsigned int n) {
2262     int pos = 0;
2263     if (n >= 1<<16) { n >>= 16; pos += 16; }
2264     if (n >= 1<<8) { n >>= 8; pos += 8; }
2265     if (n >= 1<<4) { n >>= 4; pos += 4; }
2266     if (n >= 1<<2) { n >>= 2; pos += 2; }
2267     if (n >= 1<<1) { pos += 1; }
2268     return ((n == 0) ? (-1) : pos);
2269 }
2270
2271 // QuickSortExternal_4+GB.c [
2272 int strcmpKAZE13 (
2273     const char * src,
2274     const char * dst
2275 )
2276 {
2277     int ret = 0 ;

```

```

2278 while( ! (ret = *(unsigned char *)src - *(unsigned char *)dst) && (*dst!=13-13))
2279     ++src, ++dst;
2280
2281 if ( ret < 0 )
2282     ret = -1 ;
2283 else if ( ret > 0 )
2284     ret = 1 ;
2285
2286 return( ret );
2287 }
2288
2289 #define LongestLineInclusive 31 //31 former, CAUTION: for command line options 'x' and 'y' it cannot be other than 31 [YET]!
2290 char FourGramL[LongestLineInclusive+2]; // 31 longest 4-gram + CR + LF
2291 char FourGramR[LongestLineInclusive+2]; // 31 longest 4-gram + CR + LF
2292 FILE *fp_outRG; // Global - not to burden the extract/compare function with one more parameter
2293 int CompareStringsEndingWith13_EXTERNAL(unsigned long long AtPosition64L, unsigned long long AtPosition64R) {
2294
2295     int i;
2296     unsigned long long *AtPosition64Lpointer=&AtPosition64L;
2297     unsigned long long *AtPosition64Rpointer=&AtPosition64R;
2298
2299     // Caramba: seek and tell report OK but in fact they lie, only setpos works?!?!
2300
2301     //if defined(_WIN32_ENVIRONMENT_)
2302     //    _lseeki64( fileno(fp_outRG), AtPosition64L, 0 );
2303     //else
2304     //    fseeko( fp_outRG, AtPosition64L, SEEK_SET );
2305     //endif /* defined(_WIN32_ENVIRONMENT_) */
2306
2307     // _CRTIMP __int64 __cdecl _telli64(int);
2308     // off64_t ftello64 (FILE *stream)
2309
2310     fsetpos(fp_outRG, AtPosition64Lpointer);
2311     for (i=0; i<(LongestLineInclusive+1); i++) {fread(&FourGramL[i], 1, 1, fp_outRG); if (FourGramL[i]==13-13) break;}
2312     //Commented line below is slower than the one above: 778156 clocks vs 756297 clocks.
2313     //fread(&FourGramL[0], 31+1, 1, fp_outRG);
2314
2315     fsetpos(fp_outRG, AtPosition64Rpointer);
2316     for (i=0; i<(LongestLineInclusive+1); i++) {fread(&FourGramR[i], 1, 1, fp_outRG); if (FourGramR[i]==13-13) break;}
2317     //Commented line below is slower than the one above: 778156 clocks vs 756297 clocks.
2318     //fread(&FourGramR[0], 31+1, 1, fp_outRG);
2319
2320     return(strcmpKAZE13(FourGramL, FourGramR));
2321 }
2322
2323 int CompareStringsEndingWith13_INTERNAL(unsigned long long AtPosition64L, unsigned long long AtPosition64R, char *POOLinternal) {
2324
2325     int i;
2326     //char FourGramL[LongestLineInclusive+2]; // 31 longest 4-gram + CR + LF
2327     //char FourGramR[LongestLineInclusive+2]; // 31 longest 4-gram + CR + LF
2328
2329     for (i=0; i<(LongestLineInclusive+1); i++) {
2330         //fread(&FourGramL[i], 1, 1, fp_in);
2331         FourGramL[i] = *(char *) (POOLinternal + AtPosition64L);
2332         if (FourGramL[i]==13-13) break;
2333     }
2334
2335     for (i=0; i<(LongestLineInclusive+1); i++) {
2336         //fread(&FourGramR[i], 1, 1, fp_in);
2337         FourGramR[i] = *(char *) (POOLinternal + AtPosition64R);
2338         if (FourGramR[i]==13-13) break;
2339     }
2340
2341     return(strcmpKAZE13(FourGramL, FourGramR));
2342 }
2343
2344 // QuickSortExternal_4+GB.c ]
2345
2346
2347 int main( argc, argv )
2348 int argc; char *argv[];
2349 {
2350     int nlines;
2351     string *backup = NULL;
2352
2353     FILE *fp_in, *fp_out, *fp_outLOG, *fp_inLINE;
2354     int LetterOffset;
2355     unsigned long long FilesLEN;
2356     unsigned long long WORDcount;
2357     unsigned long long WORDcountAttemptsToPut;
2358     int Thunderwith;
2359     unsigned long NumberOfFiles, WORDcountDistinct;
2360     unsigned long long NumberOfLines; // rev. 12+
2361     unsigned long WHOLEletter_BufferSize;
2362     unsigned long long WHOLEletter_BufferSize_L14;
2363     unsigned long memory_size, LetterBuffer, j, k, LINE10len, wrdlen;
2364     unsigned long k_FIX;

```

```

2366     unsigned long long i;           // rev. 12+
2367     //unsigned long size_in, size_out, size_inLINE;
2368     unsigned long size_in;          // rev. 12+
2369 #if defined(_WIN32_ENVIRONMENT_)
2370     unsigned long long size_inLINESIXFOUR;
2371 #else
2372     size_t size_inLINESIXFOUR;
2373 #endif /* defined(_WIN32_ENVIRONMENT_) */
2374
2375     //unsigned long t1, t2, t3;
2376     time_t t1, t2, t3;
2377
2378     const int NumberOfSLOTS = 4096*2; // Since r.12+ in rev.12 it was 4096
2379     unsigned long StackPtr;
2380     //unsigned long BSTstack [65536*3]; // BST in worst case could become a LL.
2381     unsigned long long BSTstack [8192*3]; // BST in worst case could become a LL.
2382     unsigned long NumberOfTrees=0, NumberOfHashCollisions=0;
2383     unsigned long iBSTwithMAXpeak, jBSTwithMAXpeak;
2384     unsigned int PEAKibBST;
2385     unsigned long BSTsTotalLEAFs=0; // ?! MADHOUSE: if BSTcurrent is not zeroed here then INSANE values of BSTcurrent occur below where 'break'
is ?!
2386     unsigned long BSTwithMAXnode=0, BSTcurrentNode=0; // ?! MADHOUSE: if BSTcurrent is not zeroed here then INSANE values of BSTcurrent occur
below where 'break' is ?!
2387     unsigned long BSTcurrentNodeMAXQUANTITY=0; // ?! MADHOUSE: if BSTcurrent is not zeroed here then INSANE values of
BSTcurrent occur below where 'break' is ?!
2388     unsigned long BSTwithMAXnodePEAK=1, BSTwithMAXnodeLEAF=1; // ?! MADHOUSE: if BSTcurrent is not zeroed here then INSANE values of BSTcurrent
occur below where 'break' is ?!
2389     unsigned long BSTwithMAXpeak=0, BSTcurrentPeak=0; // ?! MADHOUSE: if BSTcurrent is not zeroed here then INSANE values of BSTcurrent occur
below where 'break' is ?!
2390     unsigned long BSTcurrentPeakMAX=0; // ?! MADHOUSE: if BSTcurrent is not zeroed here then INSANE values of BSTcurrent occur
below where 'break' is ?!
2391     unsigned long BSTcurrentPeakMAXQUANTITY=0; // ?! MADHOUSE: if BSTcurrent is not zeroed here then INSANE values of
BSTcurrent occur below where 'break' is ?!
2392     unsigned long BSTwithMAXpeakNODE=1, BSTwithMAXpeakLEAF=1; // ?! MADHOUSE: if BSTcurrent is not zeroed here then INSANE values of BSTcurrent
occur below where 'break' is ?!
2393     unsigned long BSTwithMAXleaf=0, BSTcurrentLeaf=0; // ?! MADHOUSE: if BSTcurrent is not zeroed here then INSANE values of BSTcurrent occur
below where 'break' is ?!
2394     unsigned long BSTcurrentLeafMAXQUANTITY=0; // ?! MADHOUSE: if BSTcurrent is not zeroed here then INSANE values of
BSTcurrent occur below where 'break' is ?!
2395     unsigned long BSTwithMAXleafNODE=1, BSTwithMAXleafPEAK=1; // ?! MADHOUSE: if BSTcurrent is not zeroed here then INSANE values of BSTcurrent
occur below where 'break' is ?!
2396
2397     char *pointerflush, *pointerflushUNALIGN, *BufStart, *Flushing;
2398     unsigned long PseudoLinkedPointer, PseudoLinkedPointerNEW, PseudoLinkedPointerROOT, PseudoLinkedPointerNEWold;
2399     unsigned long PseudoLinkedPointerNEWleft, PseudoLinkedPointerNEWright;
2400     unsigned long PseudoLinkedPointerNEWmiddle;
2401     char *bufend[ 806 ]; // 'a'=0, ... 'z'=25 - 26 letters x 31 lengths
2402     long bufNumberOfwords[ 806 ]; // 'a'=0, ... 'z'=25 - 26 letters x 31 lengths
2403     // long bufNowps[ 806 ][ 8192 ]; // ?! crashes below when an attempt to use it occur
2404     char wrd[LongestLineInclusive+1]; // 0..30, 31 = 0
2405     char wrdUP[LongestLineInclusive+1]; // 0..30, 31 = 0
2406     char wrdUPold[LongestLineInclusive+1]; // 0..30, 31 = 0
2407     char LINE10[257]; // 000..255, 256 = 0
2408     char ZEROS[4]; // 0..3, 0 = 0, 1 = 0, 2 = 0, 3 = 0
2409     char CRdLFa[2]; // 0..1, 0 = 13, 1 = 10
2410     char workbyte;
2411     char workk[1024*128];
2412     long workkoffset = -1;
2413     int FoundInLinkedList, Slot;
2414     unsigned long OffsetsInBuffer[31]; // 00..30
2415     unsigned long MAXusedBuffer[32]; // 00 not used, only 01..31
2416     unsigned long GRMBLhill[32]; // 00..31
2417     unsigned long GRMBLFoolAgain[32]; // 00..31
2418     int Melnitchka;
2419     unsigned long MAXusedBufferABS = 0;
2420     unsigned long Utiliza1 = 0;
2421     unsigned long Utiliza2 = 0;
2422     unsigned long TotalWLchars = 0;
2423
2424     /* minimum signed 64 bit value */
2425     #define _I64_MIN    (-9223372036854775807i64 - 1)
2426     /* maximum signed 64 bit value */
2427     #define _I64_MAX    9223372036854775807i64
2428     /* maximum unsigned 64 bit value */
2429     #define _UI64_MAX    0xffffffffffffffffui64
2430
2431     /* minimum signed 128 bit value */
2432     #define _I128_MIN    (-170141183460469231731687303715884105727i128 - 1)
2433     /* maximum signed 128 bit value */
2434     #define _I128_MAX    170141183460469231731687303715884105727i128
2435     /* maximum unsigned 128 bit value */
2436     #define _UI128_MAX    0xffffffffffffffffffffffffffffffffui128
2437
2438     char llToaDigits[27]; // 9,223,372,036,854,775,807: 1(sign or carry)+19(digits)+1('\0')+6(,)
2439     // below duplicates are needed because of one_line_invoking need different buffers.
2440     char llToaDigits2[27]; // 9,223,372,036,854,775,807: 1(sign or carry)+19(digits)+1('\0')+6(,)
2441     char llToaDigits3[27]; // 9,223,372,036,854,775,807: 1(sign or carry)+19(digits)+1('\0')+6(,)
2442     char llToaDigits4[27]; // 9,223,372,036,854,775,807: 1(sign or carry)+19(digits)+1('\0')+6(,)

```

```

2443 unsigned long HEADOffsetFromStartBUKVA = 0;
2444 unsigned long TAILOffsetFromStartBUKVA = 0;
2445 int BStorBtree = 0;
2446 int SplitOccurred;
2447 int POffsetInLEAF;
2448 char *Auberge[4] = {"|\\0", "/\\0", "-\\0", "\\0\\0"};
2449 int hashAlfalfa, iAlfalfa;
2450 int PLE_words=0; // Quadruple!
2451 char wrd1st[LongestLineInclusive+1]; // 0..30, 31 = 0
2452 char wrd2nd[LongestLineInclusive+1]; // 0..30, 31 = 0
2453 char wrd3rd[LongestLineInclusive+1]; // 0..30, 31 = 0
2454 char wrd4th[LongestLineInclusive+1]; // 0..30, 31 = 0
2455 char *DelimiterUnderscore = "_\\0";
2456 int PLE_words_INITflag = 0;
2457
2458 // QuickSortExternal_4+GB [
2459 unsigned long long ThunderwithL64_L14;
2460 unsigned long long Strnglen64_L14;
2461 unsigned long long size_in64_L14, size_in2_L14;
2462 unsigned long long Over4billionLines, j_Over4billion;
2463 char OneChar_ieByte = '\\0';
2464 char CR_ieByte = '\\r';
2465 char SomeByte;
2466 unsigned long long BufEnd_64;
2467 unsigned long long SeekPosition;
2468 unsigned long long *PointerToSeekPosition;
2469 char FourGram[LongestLineInclusive+2]; // 31 longest 4-gram + CR + LF
2470 char *PoolPhysical;
2471 unsigned long long fsetpos_ZERO=0;
2472 char OneClusterZEROES[1024*4]; // Caution: must be ZEROed(NULLlified)!
2473 char *FileSwapTag = "LEPRECHAUNISH";
2474 char EOFcode = 0x1A;
2475 unsigned long long PseudoLinkedPointer_64, PseudoLinkedPointerNEW_64, PseudoLinkedPointerROOT_64, PseudoLinkedPointerNEWold_64;
2476 unsigned long long PseudoLinkedPointerNEWleft_64, PseudoLinkedPointerNEWright_64;
2477 unsigned long long PseudoLinkedPointerNEWmiddle_64;
2478 unsigned long long NULLs_64 = 0;
2479 unsigned long long PseudoLinkedPointerAUX_64;
2480 unsigned long long PseudoLinkedPointerAUXdumbo_64;
2481 char wrdAUX[LongestLineInclusive+1]; // 0..30, 31 = 0
2482 // QuickSortExternal_4+GB ]
2483
2484 // INIT INIT INIT INIT INIT INIT INIT INIT INIT INIT INIT INIT INIT INIT INIT
2485 puts( "Leprechaun(Fast Greedy Word-Ripper), rev. 14_minus_quadrupleton, written by Svalqyatchx." );
2486 puts( "Leprechaun: 'Oh, well, didn't you hear? Bigger is good, but jumbo is dear.'" );
2487 puts( "kaze: Let's see what a 3-way hash + 6,602,752 Binary-Search-Trees can give us," );
2488 puts( "    also the performance of a 3-way hash + 6,602,752 B-Trees of order 3," );
2489 puts( "    also the performance of a 1-way hash + 134,217,728 external B-Trees of order 3." );
2490 //puts( "Note1: Compiled with Microsoft C v. 13.10.3077: 'cl /Ox /TcLeprechaun.c'." );
2491 //puts( "Note2: This WORDLISTER makes as output pseudo(unsorted)_wordlist_CRLF_file." );
2492 if( argc != 3 && argc != 4 && argc != 5 && argc != 6 ) // +1 for program name
2493 {
2494     puts( "" );
2495     puts( "'The Little Monster' short notes:" );
2496     puts( "Note1: I wish to thank to R.N. Horspool, Ranjan Sinha, Dmitry Shkarin," );
2497     puts( "    Michael Abrash, J. Bentley, R. Sedgewick, Igor Pavlov, Lasse Reinhold," );
2498     puts( "    Landon Noll, Peter Kankowski for sharing their knowledge to public." );
2499     puts( "Note2: Run it without parameters to get usage and short notes." );
2500     puts( "Note3: This simple amateurish(more over I am not versed well neither in C nor" );
2501     puts( "    in mathematics nor in english language, but I am persistent in INDEXING" );
2502     puts( "    GBS of english TEXTS) tool is written in ANSI C(at least its source is" );
2503     puts( "    compileable for CL(Windows) and GCC(Linux)), and its purpose is to" );
2504     puts( "    create a wordList for a group of files(given via filelist)." );
2505     puts( "    Its name comes(according to Heritage Dictionary) from 'low corpus' or" );
2506     puts( "    'little body', in fact from amazing movie saga 'Leprechaun 1-2-3-4-5-6'" );
2507     puts( "    starring by warwick Davis." );
2508     puts( "Note4: Only words up to 31 chars are proceeded - the reason is 'DDT(the" );
2509     puts( "    longest word in Heritage Dictionary 3rd edition) or" );
2510     puts( "    'dichlorodiphenyltrichloroethane'." );
2511     puts( "Note5: Cursor hiding in C - mission impossible for me." );
2512     puts( "Note6: By default(third parameter is 1023) allocated memory is 393MB." );
2513     puts( "    Due to 'malloc()' limitation under WINDOWS, maximum value of third" );
2514     puts( "    parameter is 5174 which is 1988MB allocated block." );
2515     puts( "Note7: File Leprechaun.LOG is a log, where new statistics are appended." );
2516     puts( "Note8: Revision 12+ can handle files larger than 4GB." );
2517     puts( "Note9: Revision 12++ has a buffered 'fread()' - therefore I/O READ-BURST SPEED" );
2518     puts( "    is the first(worst) bottleneck, as a result r.12++ is much-much faster;" );
2519     puts( "    the second(worse) bottleneck: the linked lists - the b-trees" );
2520     puts( "    might be the answer; the third(bad) bottleneck: the amateurish author." );
2521     puts( "NoteA: Revision 12+++ has an improved(2 bits were used dolitshly) main hash" );
2522     puts( "    function - therefore less collisions, for example:" );
2523     puts( "    for file 'wikipedia-de-html.tar' 42,291,855,360 bytes with" );
2524     puts( "    5,750,179,678 words of them 7,375,373 distinct attempts to Find/Put" );
2525     puts( "    a WORD into a linked list are 6,117,675,470(r.12++) and 5,845,989,790" );
2526     puts( "    (r.12+++); also two 'if' sections were moved because they were executed" );
2527     puts( "    unnecessarily many times." );
2528     puts( "NoteB: Revision 13 uses BSTs instead of LLs, that is Linked-Lists were" );
2529     puts( "    replaced by Binary-Search-Trees, as a result for 22,202,980 distinct" );
2530     puts( "    words(out of 35,271,297) r.12+++ needs 225,548,268 total attempts to" );

```

```

2531 puts( " Find/Put WORDS into linked lists where r.13 needs 121,674,042 total" );
2532 puts( " attempts to Find/Put WORDS into Binary-Search-Trees. But this is a" );
2533 puts( " significant boost in performance only for wordlists of million words." );
2534 puts( "NoteC: Revision 13+ gives only more statistics. Future revisions could lessen" );
2535 puts( " number of attempts to Find/Put WORDS into Binary-Search-Trees" );
2536 puts( " furthermore by making them at some point Perfectly-Balanced. But" );
2537 puts( " for huge amount(multi-(m|b)illion) of distinct words the b-tree family" );
2538 puts( " must come in, until then this is the leprechaunish niche." );
2539 puts( "NoteD: Revision 13++ has a little fix(2 unnecessary ZEROings, when a new word" );
2540 puts( " is inserted, were deleted) and a fixed bug(13+ adds stupidly the" );
2541 puts( " highest BST to the wordlist). Also B-Tree of order 3 is added as a" );
2542 puts( " searching method. Main goal of B-Tree is to reduce number of" );
2543 puts( " comparisons but at nasty cost: a precious time wasted to construct it" );
2544 puts( " and twice more memory, i.e. one step forward two backward: this tree is" );
2545 puts( " more effective than BST in cases of 2++ billion/million" );
2546 puts( " different/distinct words." );
2547 puts( " The improvement which comes from using B-Tree of order 3 is about 200%" );
2548 puts( " much more pleasing than I expected, for wikipedia-en-html.tar.wrd with" );
2549 puts( " 12,561,874 distinct words Total Attempts to Find/Put WORDS into:" );
2550 puts( " Binary-Search-Trees was 61,895,043 while for" );
2551 puts( " B-trees order 3 was 19,295,791." );
2552 puts( "NoteE: Revision 13+++ has a faster(not heavily tested yet) and with" );
2553 puts( " better(0.6% to 1.1%) dispersion Fowler/Noll/Vo hash." );
2554 puts( " so called FNV1a hash. Revision 13++++ boosting: Leprechaun_Intel.exe" );
2555 puts( " gives 1,256,187w/s for wikipedia-en-html.tar.wrd with FNV1_32_PRIME" );
2556 puts( " 107712257 with 3,551,736 dispersion for 'FNV1A_Hash_Granularity'." );
2557 puts( "NoteF: For old r.12+ a USB connected HDD crippled test:" );
2558 puts( " for 'H:\>Leprechaun.exe static.wikipedia.org_downloads_2008-06_en.lst" );
2559 puts( " wikipedia-en-html.tar.wrd 5400" );
2560 puts( " where 223,674,511,360 wikipedia-en-html.tar" );
2561 puts( " on laptop Toshiba Pentium T3400 2166 MHz with" );
2562 puts( " Motherboard Name: Toshiba Satellite L305" );
2563 puts( " CPU Type: Mobile DualCore Intel Pentium, 2166 MHz (13 x 167)" );
2564 puts( " CPU Alias: Merom-1M" );
2565 puts( " L1 Code Cache: 32 KB per core" );
2566 puts( " L1 Data Cache: 32 KB per core" );
2567 puts( " L2 Cache: 1 MB (On-Die, ECC, ASC, Full-Speed)" );
2568 puts( " Bus Type: Dual DDR2 SDRAM" );
2569 puts( " Bus width: 128-bit" );
2570 puts( " Real Clock: 333 MHz (DDR)" );
2571 puts( " Effective Clock: 666 MHz" );
2572 puts( " EVEREST v5.00.1650 Memory Copy: 3725MB/s with timings 5-5-5-13" );
2573 puts( " result is logged to 'Leprechaun.LOG'" );
2574 puts( " Bytes per second performance: 20,658,955B/s" );
2575 puts( " Words per second performance: 2,860,880W/s" );
2576 puts( " Input File with a list of TEXTual Files:" );
2577 puts( " static.wikipedia.org_downloads_2008-06_en.lst" );
2578 puts( " Size of all TEXTual Files: 223,674,511,360" );
2579 puts( " Word count: 30,974,750,142 of them 12,561,874 distinct" );
2580 puts( " Number Of Files: 1" );
2581 puts( " Number Of Lines: 2088618575" );
2582 puts( " Allocated memory in MB: 1920" );
2583 puts( " Words with length 01 occupy 0,033KB of 0,349KB given i.e. 09% utilization" );
2584 puts( " Words with length 02 occupy 0,033KB of 0,349KB given i.e. 09% utilization" );
2585 puts( " Words with length 03 occupy 0,037KB of 0,697KB given i.e. 05% utilization" );
2586 puts( " Words with length 04 occupy 0,151KB of 0,871KB given i.e. 17% utilization" );
2587 puts( " Words with length 05 occupy 0,744KB of 1,568KB given i.e. 47% utilization" );
2588 puts( " Words with length 06 occupy 1,470KB of 3,136KB given i.e. 46% utilization" );
2589 puts( " Words with length 07 occupy 2,605KB of 5,923KB given i.e. 43% utilization" );
2590 puts( " Words with length 08 occupy 3,296KB of 6,968KB given i.e. 47% utilization" );
2591 puts( " Words with length 09 occupy 3,714KB of 6,968KB given i.e. 53% utilization" );
2592 puts( " Words with length 10 occupy 3,483KB of 6,968KB given i.e. 49% utilization" );
2593 puts( " Words with length 11 occupy 3,235KB of 5,923KB given i.e. 54% utilization" );
2594 puts( " Words with length 12 occupy 2,691KB of 4,181KB given i.e. 64% utilization" );
2595 puts( " Words with length 13 occupy 2,230KB of 3,484KB given i.e. 64% utilization" );
2596 puts( " Words with length 14 occupy 1,718KB of 3,484KB given i.e. 49% utilization" );
2597 puts( " Words with length 15 occupy 1,357KB of 2,613KB given i.e. 51% utilization" );
2598 puts( " Words with length 16 occupy 1,063KB of 2,613KB given i.e. 40% utilization" );
2599 puts( " Words with length 17 occupy 0,814KB of 1,742KB given i.e. 46% utilization" );
2600 puts( " Words with length 18 occupy 0,617KB of 1,742KB given i.e. 35% utilization" );
2601 puts( " Words with length 19 occupy 0,485KB of 1,742KB given i.e. 27% utilization" );
2602 puts( " Words with length 20 occupy 0,402KB of 1,742KB given i.e. 23% utilization" );
2603 puts( " Words with length 21 occupy 0,327KB of 1,742KB given i.e. 18% utilization" );
2604 puts( " Words with length 22 occupy 0,274KB of 1,742KB given i.e. 15% utilization" );
2605 puts( " Words with length 23 occupy 0,224KB of 1,394KB given i.e. 16% utilization" );
2606 puts( " Words with length 24 occupy 0,190KB of 1,394KB given i.e. 13% utilization" );
2607 puts( " Words with length 25 occupy 0,162KB of 1,394KB given i.e. 11% utilization" );
2608 puts( " Words with length 26 occupy 0,136KB of 1,220KB given i.e. 11% utilization" );
2609 puts( " Words with length 27 occupy 0,119KB of 1,046KB given i.e. 11% utilization" );
2610 puts( " Words with length 28 occupy 0,107KB of 0,871KB given i.e. 12% utilization" );
2611 puts( " Words with length 29 occupy 0,091KB of 0,697KB given i.e. 13% utilization" );
2612 puts( " Words with length 30 occupy 0,080KB of 0,523KB given i.e. 15% utilization" );
2613 puts( " Words with length 31 occupy 0,076KB of 0,523KB given i.e. 14% utilization" );
2614 puts( " Total pseudo(including hash table) memory utilization: 42%" );
2615 puts( " Total real(wordlist's words vs allocated block) memory utilization: 60/1000" );
2616 puts( " Used value for third parameter in KB: 5400" );
2617 puts( " Use next time as third parameter: 3475-" );
2618 puts( " Time for making unsorted wordlist: 10827 second(s)" );

```

```

2619 puts( " Time for sorting unsorted wordlist: 10 second(s)" );
2620 puts( "NoteG: 2011-Mar-07: Fixed a small command line parsing bug." );
2621 puts( "NoteH: A heavy blow for my illusions(regarding speed performance of external b-trees)," );
2622 puts( "    desperate results for ripping on HDD 7200rpm:" );
2623 puts( "    20,000,000 distinct 4-grams per 5 hours." );
2624 puts( "    D:\\Leprechaun_quadupleton_r14_minus>Leprechaun_quadupleton.exe GRAFFITH_2048.lst GRAFFITH_2048.wrd 48000000 z" );
2625 puts( "    Leprechaun(Fast Greedy Word-Ripper), rev. 14_minus_quadupleton, written by Svalqyatchx." );
2626 puts( "    Leprechaun: 'Oh, well, didn't you hear? Bigger is good, but jumbo is dear.'" );
2627 puts( "    Kaze: Let's see what a 3-way hash + 6,602,752 Binary-Search-Trees can give us," );
2628 puts( "        also the performance of a 3-way hash + 6,602,752 B-Trees of order 3," );
2629 puts( "        also the performance of a 1-way hash + 134,217,728 external B-Trees of order 3." );
2630 puts( "    Size of input file with files for Leprechauning: 42140" );
2631 puts( "    Allocating HASH memory 1,073,741,889 bytes ... OK" );
2632 puts( "    Allocating/ZEROing 49,152,000,014 bytes swap file ... OK" );
2633 puts( "    Size of Input TEXTual file: 33,470,581" );
2634 puts( "    |; Word count: 3,045,077 of them 2,597,942 distinct; Done: 64/64" );
2635 puts( "    Size of Input TEXTual file: 17,229,900" );
2636 puts( "    -; Word count: 4,235,032 of them 3,588,757 distinct; Done: 64/64" );
2637 puts( "    Size of Input TEXTual file: 19,191,256" );
2638 puts( "    |; Word count: 5,803,400 of them 4,866,213 distinct; Done: 64/64" );
2639 puts( "    Size of Input TEXTual file: 34,651,077" );
2640 puts( "    \\; Word count: 8,714,961 of them 6,941,108 distinct; Done: 64/64" );
2641 puts( "    Size of Input TEXTual file: 26,875,458" );
2642 puts( "    /; Word count: 11,022,830 of them 8,579,931 distinct; Done: 64/64" );
2643 puts( "    Size of Input TEXTual file: 19,605,129" );
2644 puts( "    -; Word count: 12,924,821 of them 10,078,191 distinct; Done: 64/64" );
2645 puts( "    Size of Input TEXTual file: 17,053,521" );
2646 puts( "    /; Word count: 14,577,010 of them 11,455,983 distinct; Done: 64/64" );
2647 puts( "    Size of Input TEXTual file: 44,087,709" );
2648 puts( "    -; Word count: 18,953,280 of them 15,010,569 distinct; Done: 64/64" );
2649 puts( "    Size of Input TEXTual file: 32,796,705" );
2650 puts( "    |; Word count: 22,412,912 of them 17,621,649 distinct; Done: 64/64" );
2651 puts( "    Size of Input TEXTual file: 19,538,360" );
2652 puts( "    /; Word count: 24,381,005 of them 19,137,701 distinct; Done: 64/64" );
2653 puts( "    Size of Input TEXTual file: 29,565,366" );
2654 puts( "    \\; Word count: 26,214,400 of them 20,528,357 distinct; Done: 40/64" );
2655 puts( "    ..." );
2656 puts( "NoteI: In revision 14- the resultant wordlist is NOT sorted when 'Z' is used." );
2657 puts( "" );
2658 puts( "Usage: Leprechaun InFile OutFile [BufferSize] [SortMethod] [TreeMethod]" );
2659 puts( "    <InFile>: Input file with files for Leprechauning, in WINDOWS console" );
2660 puts( "        you can create it by 'E:\\KAZEHOME>dir *.txt/s/b>Leprechaun.lst'" );
2661 puts( "    <OutFile>: Output WORDLIST(sorted since r.9, CRLF) file" );
2662 puts( "    <BufferSize>: Optional Dynamic RAM buffer in KB, default(and minimum" );
2663 puts( "        in the same time) is 1023, i.e. omit or specify greater one" );
2664 puts( "    <SortMethod>: Optional Sort Method, default is 'D'," );
2665 puts( "        A - InsertionSort" );
2666 puts( "        B - InsertionX26Sort" );
2667 puts( "        C - MultiKeyQuickSortSort by J. Bentley, R. Sedgewick" );
2668 puts( "        D - MultiKeyQuickSortX26Sort' by J. Bentley, R. Sedgewick" );
2669 puts( "    <TreeMethod>: Optional Tree Method, default is 'X'," );
2670 puts( "        X - Binary-Search-Trees" );
2671 puts( "        Y - B-Trees of order 3, INTERNAL/fast memory" );
2672 puts( "        Z - B-Trees of order 3, EXTERNAL/slow memory, 64bit addressing!" );
2673 puts( "" );
2674 puts( "Have a nice Leprechauning." );
2675 puts( "For contacts: sanmayce@sanmayce.com" );
2676 puts( "Sanmayce Svalqyatchx 'Kaze', 2005 Feb 07(rev. 14- quadrupleton: 2011 Jun 01)." );
2677 return( 1 );
2678 }
2679
2680 GRMBLhi11[0]=0;
2681 GRMBLhi11[1]=1;
2682 GRMBLhi11[2]=1;
2683 GRMBLhi11[3]=1;
2684 GRMBLhi11[4]=1;
2685 GRMBLhi11[5]=1;
2686 GRMBLhi11[6]=1;
2687 GRMBLhi11[7]=1;
2688 GRMBLhi11[8]=1;
2689 GRMBLhi11[9]=1;
2690 GRMBLhi11[10]=1;
2691 GRMBLhi11[11]=1;
2692 GRMBLhi11[12]=15;
2693 GRMBLhi11[13]=15;
2694 GRMBLhi11[14]=15;
2695 GRMBLhi11[15]=20;
2696 GRMBLhi11[16]=30;
2697 GRMBLhi11[17]=40;
2698 GRMBLhi11[18]=50;
2699 GRMBLhi11[19]=50;
2700 GRMBLhi11[20]=50;
2701 GRMBLhi11[21]=40;
2702 GRMBLhi11[22]=40;
2703 GRMBLhi11[23]=40;
2704 GRMBLhi11[24]=30;
2705 GRMBLhi11[25]=20;
2706 GRMBLhi11[26]=20;

```



```

2707 GRMBLhill[27]=20;
2708 GRMBLhill[28]=20;
2709 GRMBLhill[29]=20;
2710 GRMBLhill[30]=10;
2711 GRMBLhill[31]=10;
2712
2713 if( ( fp_in = fopen( argv[1], "rb" ) ) == NULL )
2714 { printf( "Leprechaun: Can't open file %s \n", argv[1] ); return( 1 ); }
2715
2716 fseek( fp_in, 0L, SEEK_END );
2717 size_in = ftell( fp_in );
2718 fseek( fp_in, 0L, SEEK_SET );
2719 printf( "Size of input file with files for Leprechauning: %lu\n", size_in );
2720
2721 if( ( fp_outLOG = fopen( "Leprechaun.LOG", "a+" ) ) == NULL )
2722 { printf( "Leprechaun: Can't open file Leprechaun.LOG.\n" ); return( 1 ); }
2723
2724 // argc is 4|5|6 due to eventual missing BufferSize
2725 if( argc == 4 ) // not 6 due to eventual missing BufferSize and SortMethod
2726     k_FIX = 3;
2727 if( argc == 5 ) // not 6 due to eventual missing BufferSize or SortMethod
2728     k_FIX = 4;
2729 if( argc == 6 )
2730     k_FIX = 5;
2731 if ( *argv[k_FIX] == 'y' || *argv[k_FIX] == 'y') BStorBtree = 1;
2732 if ( *argv[k_FIX] == 'z' || *argv[k_FIX] == 'z') BStorBtree = 2;
2733
2734 if( argc == 4 || argc == 5 || argc == 6 ) Thunderwith = atoi( argv[3] );
2735 else Thunderwith = 527; // for r.12: 527=17*31 this is minimum because of 4096*1*4=16KB+ needed for each buffer!
2736 // for r.12: 1023=33*31 this is minimum because of 4096*2*4=32KB+ needed for each buffer!
2737 if (Thunderwith < 1023) {Thunderwith = 1023;}
2738
2739 //printf( "Use next time as third parameter: %s\n", _ui64toaKAZEcomma((25123456789>>10)+1, 11ToaDigits, 10) );
2740 //printf( "Use next time as third parameter: %s\n", _ui64toaKAZEcomma((25123456789/1024)+1, 11ToaDigits, 10) );
2741
2742 if (BStorBtree != 2) {
2743     LetterBuffer = Thunderwith * 1024;
2744     WHOLEletter_BufferSize = 0;
2745     for( i = 1; i <= 31; i++ )
2746     { OffsetsInBuffer[i-1] = 0;
2747       for( j = 1; j <= i; j++ )
2748       { OffsetsInBuffer[i-1] = OffsetsInBuffer[i-1] + (GRMBLhill[(int)(j-1)] * LetterBuffer)/31;
2749       }
2750       WHOLEletter_BufferSize = WHOLEletter_BufferSize + (GRMBLhill[(int)i] * LetterBuffer)/31;
2751       GRMBLFoolAgain[(int)i] = (GRMBLhill[(int)i] * LetterBuffer)/31;
2752     }
2753     memory_size = 26 * WHOLEletter_BufferSize + 1 + 64;
2754     printf( "Allocating memory %luMB ... ", (memory_size>>20)+1 );
2755     pointerflushUNALIGN = (char *)malloc( memory_size );
2756     if( pointerflushUNALIGN == NULL )
2757     { puts( "\nLeprechaun: Needed memory allocation denied!\n" ); return( 1 ); }
2758     pointerflush = pointerflushUNALIGN + 64 - (((size_t)pointerflushUNALIGN) % 64); // 13_6+
2759     //offset=64-int((long)data&63);
2760
2761     printf( "OK\n" );
2762     fprintf( fp_outLOG, "Leprechaun report:\n" );
2763
2764     // Check once for ever whether allocated memory is ZEROed!? Answer: YES
2765     //for( i = 0; i < memory_size; i++ )
2766     // if (*(char *) (pointerflush+i)!=0) printf("NON-ZERO encountered, so 'NO'.");
2767
2768     for( i = 0; i < 26; i++ )
2769     { for( k = 1; k <= 31; k++ )
2770       { bufend[i*31+k-1] = pointerflush + i * WHOLEletter_BufferSize + OffsetsInBuffer[k-1]; // i*31+k-1 must be 0..805
2771         if (i==25) { MAXusedBuffer[k] = (unsigned long)bufend[i*31+k-1]; }
2772         for( j = 0; j < (NumberOfSLOTS+1)*4; j++ ) // ? memset(bufend[i],0,(NumberOfSLOTS+1)*4);
2773         { *bufend[i*31+k-1]++ = 0;
2774           //++bufend[i*31+k-1];
2775         }
2776         if (i==25) { MAXusedBuffer[k] = (unsigned long)bufend[i*31+k-1]-MAXusedBuffer[k]; }
2777         bufNumberOfWords[i*31+k-1]=0;
2778         //for( j = 0; j < NumberOfSLOTS; j++ )
2779         //bufNowps[i*31+k-1][j]=0;
2780       }
2781     }
2782
2783 } else { //if (BStorBtree != 2) {
2784 // _ ASCII code 095
2785 // ~ ASCII code 096 \
2786 // a ASCII code 097 / In total 26+1+1 radix instead of 27 to avoid +1 for each '_', code 096 not used.
2787 // z ASCII code 122
2788 // The hash for 'a_quadruplet_for_example' will be calculated for first 5 chars:
2789 // = (byte1-'_')*28*28*28*28 + (byte2-'_')*28*28*28 + (byte3-'_')*28*28 + (byte4-'_')*28 + (byte5-'_')
2790 // Hash slots are 28*28*28*28 = 17,210,368 each containing one 64bit pointer i.e. 8bytes in length.
2791 // Hash size = 17,210,368*8 = 137,682,944 bytes
2792 // when at end all these slots(17,210,368- Btrees) are traversed the outcome is a sorted wordlist - no need of sorting.
2793 //unsigned long long SeekPosition;
2794 //unsigned long long *PointerToSeekPosition;

```

```

2795 // The 64bit external pool will be addressed via fsetpos(fp_outRG, PointerToSeekPosition); similarly to bufend approach from r.13 - that is
    bufend points to first(always following the last used btree leaf) free position in the pool.
2796 // For final stats all non-zero slots point to one btree.
2797 // printf( "Allocating HASH memory %s bytes ... ", _ui64toaKAZEcomma( (17210368*8) + 1 + 64 , llToaDigits, 10) );
2798 // pointerflushUNALIGN = (char *)malloc( (17210368*8) + 1 + 64 );
2799 // Hash slots are 27bit = 2^27 = 134,217,728 each containing one 64bit pointer i.e. 8bytes in length.
2800 printf( "Allocating HASH memory %s bytes ... ", _ui64toaKAZEcomma( (134217728*8) + 1 + 64 , llToaDigits, 10) );
2801 pointerflushUNALIGN = (char *)malloc( (134217728*8) + 1 + 64 );
2802 if( pointerflushUNALIGN == NULL )
2803 { puts( "\nLeprechaun: Needed memory allocation denied!\n" ); return( 1 ); }
2804 pointerflush = pointerflushUNALIGN + 64 - (((size_t)pointerflushUNALIGN) % 64); // 13_6+
2805 //offset=64-int((long)data&63);
2806 printf( "OK\n" );
2807 // memset(pointerflush,0,17210368*8);
2808 memset(pointerflush,0,134217728*8);
2809 if( ( fp_outRG = fopen( "Leprechaun_64bit.swp", "wb+" ) ) == NULL )
2810 { printf( "Leprechaun: Can't create file 'Leprechaun_64bit.swp'.\n" ); return( 1 ); }
2811 // Tag for the swap file is: LEPRECHAUNISH{ASCIIcode26}
2812 // or 14bytes, then when type of the swap is requested:
2813 // D:\_KAZE~1\LEPREC~1>type Leprechaun_64bit.swp
2814 // LEPRECHAUNISH
2815 // D:\_KAZE~1\LEPREC~1>
2816 size_in64_L14 = 1024 * (unsigned long long)Thunderwith + 14;
2817 BufEnd_64 = 0+14;
2818 // The tag plays two roles, the second to avoid existence of SeekPosition equal to 0. The 0 cannot be used as a free slot FLAG without the
    TAG.
2819 printf( "Allocating/ZEROing %s bytes swap file ... ", _ui64toaKAZEcomma(size_in64_L14, llToaDigits, 10) );
2820 fsetpos(fp_outRG, &fsetpos_ZERO); // SOMETHING ROTTEN with lseeki64/fseeko and fsetpos ???! So DO-IT-OVER.
2821 memset(OneKusterZEROES,0,1024*4);
2822 for (ThunderwithL64_L14=0; ThunderwithL64_L14 < size_in64_L14/(1024*4); ThunderwithL64_L14++)
2823     fwrite(OneKusterZEROES, 1024*4, 1, fp_outRG);
2824 for (ThunderwithL64_L14=0; ThunderwithL64_L14 < size_in64_L14%(1024*4); ThunderwithL64_L14++)
2825     fwrite(&OneChar_ieByte, 1, 1, fp_outRG);
2826 fsetpos(fp_outRG, &fsetpos_ZERO); // SOMETHING ROTTEN with lseeki64/fseeko and fsetpos ???! So DO-IT-OVER.
2827 fwrite(FileSwapTag, 13, 1, fp_outRG);
2828 fwrite(&EOFCODE, 1, 1, fp_outRG);
2829 fsetpos(fp_outRG, &BufEnd_64); // SOMETHING ROTTEN with lseeki64/fseeko and fsetpos ???! So DO-IT-OVER.
2830 printf( "OK\n" );
2831 fprintf( fp_outLOG, "Leprechaun report:\n" );
2832 } //if (BStorBtree != 2) {
2833
2834 // PROGRAM PROGRAM PROGRAM PROGRAM PROGRAM PROGRAM PROGRAM PROGRAM PROGRAM
2835 (void) time(&t1);
2836     Melnitchka = 0;
2837     WORDcount = 0; // Total word count i.e. for all files!
2838     WORDcountDistinct = 0;
2839     NumberOfFiles = 0;
2840     NumberOfLines = 0;
2841     FilesLEN = 0;
2842     LINE10len = 0;
2843
2844     for( k = 0; k < size_in; k++ )
2845     {
2846         fread( &workbyte, 1, 1, fp_in );
2847         if( workbyte != 10 )
2848         { if( workbyte != 13 ) // NON UNIX
2849             { if( LINE10len < 255 ) { LINE10[ LINE10len ] = workbyte; }
2850                 LINE10len++;
2851             }
2852             else
2853             {
2854             }
2855         }
2856     }
2857     else
2858     { if( 1 <= LINE10len && LINE10len <= 255 )
2859         { LINE10[ LINE10len ] = 0;
2860         }
2861     }
2862     if( ( fp_inLINE = fopen( LINE10, "rb" ) ) == NULL )
2863     { printf( "Leprechaun: Can't open file %s\n", LINE10 ); return( 1 ); }
2864
2865     //fseek( fp_inLINE, 0L, SEEK_END ); //Rev. 12
2866     //size_inLINE = ftell( fp_inLINE ); //Rev. 12
2867     //fseek( fp_inLINE, 0L, SEEK_SET ); //Rev. 12
2868
2869     #if defined(_WIN32_ENVIRONMENT_)
2870     // 64bit:
2871     _lseeki64( fileno(fp_inLINE), 0L, SEEK_END );
2872     size_inLINESIXFOUR = _telli64( fileno(fp_inLINE) );
2873     _lseeki64( fileno(fp_inLINE), 0L, SEEK_SET );
2874     #else
2875     // 64bit:
2876     fseeko( fp_inLINE, 0L, SEEK_END );
2877     size_inLINESIXFOUR = ftello( fp_inLINE );
2878     fseeko( fp_inLINE, 0L, SEEK_SET );
2879     #endif /* defined(_WIN32_ENVIRONMENT_) */
2880
2881     printf( "Size of Input TEXTual file: %s\n", _ui64toaKAZEcomma(size_inLINESIXFOUR, llToaDigits, 10) );
2882     FilesLEN = FilesLEN + size_inLINESIXFOUR;
2883     NumberOfFiles++;

```

```

2881
2882 //~~~~~
2883 wrdlen = 0;
2884 for( i = 0; i < size_inLINESIXFOUR; i++ )
2885 {
2886     // ~~~~~ Buffering fread [
2887     if (workKoffset == -1) {
2888         if (i + 1024*128 < size_inLINESIXFOUR) {
2889             fread( &workK[0], 1, 1024*128, fp_inLINE );
2890             workKoffset = 0;
2891             workbyte = workK[workKoffset];
2892         } else
2893             fread( &workbyte, 1, 1, fp_inLINE );
2894     } else {
2895         workKoffset++;
2896         workbyte = workK[workKoffset];
2897         if (workKoffset == 1024*128 - 1) workKoffset = -1;
2898     }
2899     // ~~~~~ Buffering fread ]
2900
2901     if( isalpha( workbyte ) )
2902     {
2903         if( wrdlen < 31 )
2904             { wrd[ wrdlen ] = tolower( workbyte ); }
2905         wrdlen++;
2906     }
2907
2908     if ( workbyte < 'A' ) // Most characters are under alphabet - only one if
2909     {
2910         Elstupido:
2911             // This fragment is MIRRORed: #1 copy [
2912             if (workbyte == 10) {NumberOfLines++;}
2913
2914         // Quadruple! [
2915         // Sliding window for ' wrd ': The incoming string 'a lot of things must' becomes 'a_lot_of_things' and 'lot_of_things_must':
2916         // ain_t_that_a
2917         // didn_t_feel_a
2918         // i_didn_t_feel
2919         // t_feel_a_thing
2920         // t_that_a_cake
2921
2922         // 316
2923         // 00:17:55,859 --> 00:17:58,447
2924         // Ain't that a cake ? I didn't feel a thing !
2925
2926         if ( PLE_words_INITflag == 0 && ( (PLE_words != 0) || (PLE_words == 0 && wrdlen != 0) ) )
2927             if ( workbyte == '.' || workbyte == '!' || workbyte == '?' || workbyte == ':' || workbyte == ';' || workbyte == ','
2928                 || workbyte == '\t' ) {
2929                 PLE_words_INITflag = 1;
2930             }
2931         // Quadruple! ]
2932
2933         //if ( 1 <= wrdlen && wrdlen <= 31 )
2934         if ( 1 <= wrdlen && wrdlen <= LongestLineInclusive )
2935         {
2936             wrd[ wrdlen ] = 0;
2937             // OTKACHAM: 1<<17-1 gives 65536 i.e. '-' have had high priority than '<<'
2938             //Next line gives error due to mix of '&' and 'double'
2939             if ((WORDcount & ((1<<18)-1)) == 0)
2940             { // _ui64toaKAZEzerocomma(WORDcount, llToaDigits, 10);
2941                 //printf( "word count: %s(%lu/128 done)\r", llToaDigits, ((long long)i*100) / size_inLINESIXFOUR );
2942             }
2943             //++MeInitchka;
2944             //MeInitchka = MeInitchka % 4;
2945             //if (MeInitchka == 0){ printf( "|; word count: %s of them %s distinct; Done: %lu/64\r", _ui64toaKAZEcomma(WORDcount, llToaDigits, 10),
2946                 _ui64toaKAZEcomma((unsigned long long)WORDcountDistinct, llToaDigits2, 10), ((long long)i<<6) / size_inLINESIXFOUR ); }
2947             //if (MeInitchka == 1){ printf( "/; word count: %s of them %s distinct; Done: %lu/64\r", _ui64toaKAZEcomma(WORDcount, llToaDigits, 10),
2948                 _ui64toaKAZEcomma((unsigned long long)WORDcountDistinct, llToaDigits2, 10), ((long long)i<<6) / size_inLINESIXFOUR ); }
2949             //if (MeInitchka == 2){ printf( "-; word count: %s of them %s distinct; Done: %lu/64\r", _ui64toaKAZEcomma(WORDcount, llToaDigits, 10),
2950                 _ui64toaKAZEcomma((unsigned long long)WORDcountDistinct, llToaDigits2, 10), ((long long)i<<6) / size_inLINESIXFOUR ); }
2951             //if (MeInitchka == 3){ printf( "\\; word count: %s of them %s distinct; Done: %lu/64\r", _ui64toaKAZEcomma(WORDcount, llToaDigits, 10),
2952                 _ui64toaKAZEcomma((unsigned long long)WORDcountDistinct, llToaDigits2, 10), ((long long)i<<6) / size_inLINESIXFOUR ); }
2953             MeInitchka = MeInitchka & 3; // 0 1 2 3: 00 01 10 11
2954             printf( "%s; word count: %s of them %s distinct; Done: %lu/64\r", Auberger[MeInitchka++], _ui64toaKAZEcomma(WORDcount, llToaDigits, 10),
2955                 _ui64toaKAZEcomma((unsigned long long)WORDcountDistinct, llToaDigits2, 10), ((long long)i<<6) / size_inLINESIXFOUR );
2956         }
2957     }
2958     // Quadruple! [
2959     PLE_words++;
2960     if (PLE_words == 1)
2961         strcpy( wrd1st, wrd );
2962     else if (PLE_words == 2)
2963         strcpy( wrd2nd, wrd );
2964     else if (PLE_words == 3)
2965         strcpy( wrd3rd, wrd );
2966     else if (PLE_words == 4) {
2967         strcpy( wrd4th, wrd );

```

```

2963 wrdlen = strlen(wrd1st)+strlen(wrd2nd)+strlen(wrd3rd)+strlen(wrd4th)+1+1+1; // '_' '_' '_'
2964 //wrdlen = strlen(wrd);
2965 //if ( wrdlen <= 31 ) {
2966 if ( wrdlen <= LongestLineInclusive ) {
2967 strcpy(wrd, wrd1st);
2968 strcat(wrd, DelimiterUnderscore);
2969 strcat(wrd, wrd2nd);
2970 strcat(wrd, DelimiterUnderscore);
2971 strcat(wrd, wrd3rd);
2972 strcat(wrd, DelimiterUnderscore);
2973 strcat(wrd, wrd4th);
2974 }
2975 }
2976 else {
2977 PLE_words = 4;
2978 strcpy( wrd1st, wrd2nd );
2979 strcpy( wrd2nd, wrd3rd );
2980 strcpy( wrd3rd, wrd4th );
2981 strcpy( wrd4th, wrd );
2982 wrdlen = strlen(wrd1st)+strlen(wrd2nd)+strlen(wrd3rd)+strlen(wrd4th)+1+1+1; // '_' '_' '_'
2983 //wrdlen = strlen(wrd);
2984 //if ( wrdlen <= 31 ) {
2985 if ( wrdlen <= LongestLineInclusive ) {
2986 strcpy(wrd, wrd1st);
2987 strcat(wrd, DelimiterUnderscore);
2988 strcat(wrd, wrd2nd);
2989 strcat(wrd, DelimiterUnderscore);
2990 strcat(wrd, wrd3rd);
2991 strcat(wrd, DelimiterUnderscore);
2992 strcat(wrd, wrd4th);
2993 }
2994 }
2995 // Quadruple! ]
2996
2997 //if ( ( PLE_words == 4 ) && ( 12 <= wrdlen ) && ( wrdlen <= 31 ) ) {
2998 if ( ( PLE_words == 4 ) && ( 12 <= wrdlen ) && ( wrdlen <= LongestLineInclusive ) ) {
2999 WORDcount++;
3000 if (BSTorBtree != 2) {
3001
3002 LetterOffset = (int)( wrd[0] - 'a' ) * 31 + (wrdlen-1); // 0..805
3003 //BufStart = pointerflush + LetterOffset * LetterBuffer; // OLD
3004
3005 BufStart = pointerflush + (int)( wrd[0] - 'a' ) * WHOLEletter_Buffersize + OffsetsInBuffer[wrdlen-1];
3006 // Above line and Below line are equal
3007 //BufStart = pointerflush + (LetterOffset / 31) * WHOLEletter_Buffersize + OffsetsInBuffer[LetterOffset % 31];
3008
3009 //Slot = KuxHash3plus(wrd)<<2; //13++
3010 //Slot = FNV1A_Hash_SHIFTless_XORless(wrd)<<2; //13+++
3011 //Slot = FNV1A_Hash_4_OCTETS(wrd, wrdlen>>2)<<2; //13++++
3012 //Slot = FNV1A_Hash_4_OCTETS_31(wrd, wrdlen>>2)<<2; //13+++++
3013 /*
3014 if (wrdlen<=19) // 4x4+3=19
3015 Slot = FNV1A_Hash_4_OCTETS(wrd, wrdlen>>2)<<2; //13++++
3016 else // 2x8+4=20 i.e. first contains 5 clashes
3017 Slot = FNV1A_Hash_8_OCTETS(wrd, wrdlen>>3)<<2; //13+++++
3018 */
3019 //if (wrdlen<=19) // 4x4+3=19 i.e. last contains 7 clashes
3020 // Slot = FNV1A_Hash_Granularity(wrd, wrdlen>>2, 2)<<2; //13+++++
3021 //else // 2x8+4=20 i.e. first contains 6 clashes
3022 // Slot = FNV1A_Hash_Granularity(wrd, wrdlen>>3, 3)<<2; //13+++++
3023
3024 //Slot = FNV1A_Hash_8_OCTETS(wrd, wrdlen>>3)<<2; //13_7p
3025 //Slot = HashFNV1A_unrolled_Final(wrd, wrdlen)<<2; //13_7p
3026 //Slot = HashAlfalfa_HALF(wrd, wrdlen)<<2; //13_7p
3027 //Slot = HashAlfalfa(wrd, wrdlen)<<2; //13_7p
3028 // Slot = Sixtinsensitive(wrd, wrdlen)<<2; //13_7p
3029 Slot = FNV1A_Hash_Jesteress(wrd, wrdlen)<<2; //13_7p
3030 // Slot = FNV1A_Hash_Jester(wrd, wrdlen)<<2; //13_7p
3031
3032 /*
3033 hashAlfalfa = 7;
3034 for(iAlfalfa = 0; iAlfalfa < (wrdlen & -2); iAlfalfa += 2) {
3035 hashAlfalfa = (17+9) * ((17+9) * hashAlfalfa + (wrd[iAlfalfa])) + (wrd[iAlfalfa+1]);
3036 }
3037 if(wrdlen & 1)
3038 hashAlfalfa = (17+9) * hashAlfalfa + (wrd[wrdlen-1]);
3039
3040 Slot = (( hashAlfalfa ^ (hashAlfalfa >> 16) ) & 8191)<<2; //13_7p
3041 */
3042
3043 memcpy( &PseudoLinkedPointer, BufStart+Slot, 4 );
3044 //; Line 917
3045 // mov edx, DWORD PTR [eax+ebp]
3046 // add esp, 4
3047 // ?! DANGEROUS: above and below lines are(must:long must be 4bytes) identical
3048 //PseudoLinkedPointer = (unsigned long)*(long *) (BufStart+Slot);
3049 //; Line 919
3050 // mov edx, DWORD PTR [eax+ebp]

```

```

3051 // add     esp, 4
3052 //         while (count--) {
3053 //             *(char *)dst = *(char *)src;
3054 //             dst = (char *)dst + 1;
3055 //             src = (char *)src + 1;
3056 //         }
3057
3058 if (BStorBtree == 0)
3059 {
3060 // @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ BST fragment [
3061     if (PseudoLinkedPointer == 0) // means EMPTY-SLOT
3062     {
3063         //if( (unsigned long)(bufend[LetterOffset] - BufStart) + wrdlen + 4 < (GRMBLhi11[(int)wrdlen] * LetterBuffer)/31 ) // OLD slower
3064         if( (unsigned long)(bufend[LetterOffset] - BufStart) + wrdlen + 4 + 4 < GRMBLFoolAgain[(int)wrdlen] ) // +4 more for BST instead
3065         of LL
3066         {
3067             memcpy( BufStart+Slot, &bufend[LetterOffset], 4 );
3068
3069             //; Line 932
3070             // mov     DWORD PTR [eax+ebp], esi
3071             //      // ?! DANGEROUS: above and below lines are(must:long must be 4bytes) identical
3072             //      /*(long *) (BufStart+Slot) = *(long *)&bufend[LetterOffset];
3073
3074             //; Line 936
3075             // mov     DWORD PTR [eax+ebp], esi
3076
3077             // Below 3 lines are commented due to experiment below malloc which shows that allocated memory is ZEROed.
3078             //memcpy( bufend[LetterOffset], &BufStart[NumberOfSlots*4], 4 ); // means next exists not: Means PseudoLinkedPointerL =
3079             0
3080             //; Line 940
3081             // mov     ecx, DWORD PTR [ebp+32768]
3082             //      // ?! DANGEROUS: above and below lines are(must:long must be 4bytes) identical
3083             //      /*(long *)bufend[LetterOffset] = *(long *)&BufStart[NumberOfSlots*4];
3084
3085             //; Line 944
3086             // mov     ecx, DWORD PTR [ebp+32768]
3087             //      //bufend[LetterOffset] = bufend[LetterOffset] + 4;
3088             //      memcpy( bufend[LetterOffset], &BufStart[NumberOfSlots*4], 4 ); // means next exists not: Means PseudoLinkedPointerR =
3089             0
3090             bufend[LetterOffset] = bufend[LetterOffset] + 4 + 4; // + 4 due to above commenting
3091             memcpy( bufend[LetterOffset], wrd, wrdlen ); WORDcountDistinct++; bufNumberOfWords[LetterOffset]++;
3092             //bufNowPS[LetterOffset][Slot]++; // ?! crashes
3093             bufend[LetterOffset] = bufend[LetterOffset] + wrdlen;
3094             if (MAXusedBuffer[wrdlen] < (unsigned long)(bufend[LetterOffset] - BufStart)) {MAXusedBuffer[wrdlen] = (unsigned
3095             long)(bufend[LetterOffset] - BufStart);}
3096         }
3097     }
3098     else
3099     { printf( "\nLeprechaun: Failure! Increment 'Memory for each Letter' parameter(third one)!\n" );
3100       fprintf( fp_outLOG, "Input File with a list of TEXTual Files: %s\n", argv[1] );
3101       fprintf( fp_outLOG, "Size of all TEXTual Files: %s\n", _ui64toaKAZEcomma(FilesLEN, 11ToaDigits, 10) );
3102       fprintf( fp_outLOG, "Word count: %s of them %s distinct\n", _ui64toaKAZEcomma(WORDcount, 11ToaDigits, 10), _ui64toaKAZEcomma((unsigned long
3103       long)WORDcountDistinct, 11ToaDigits2, 10) );
3104       fprintf( fp_outLOG, "Number Of Files: %lu\n", NumberOfFiles );
3105       fprintf( fp_outLOG, "Number Of Lines: %lu\n", NumberOfLines );
3106       fprintf( fp_outLOG, "Allocated memory in MB: %lu\n", (unsigned long)(memory_size>>20)+1 );
3107       fprintf( fp_outLOG, "Total Attempts to Find/Put WORDs into Binary-Search-Trees: %s\n", _ui64toaKAZEcomma(WORDcountAttemptsToPut, 11ToaDigits,
3108       10) );
3109       for( k = 1; k < 32; k++ )
3110       { fprintf( fp_outLOG, "words with length %s occupy %sKB of %sKB given i.e. %s%s utilization\n", _ui64toaKAZEzerocomma(k, 11ToaDigits, 10)+(26-
3111       2), _ui64toaKAZEzerocomma((MAXusedBuffer[k]>>10)+1, 11ToaDigits2, 10)+(26-5), _ui64toaKAZEzerocomma(((GRMBLhi11[(int)k] *
3112       LetterBuffer)/31)>>10)+1, 11ToaDigits3, 10)+(26-5), _ui64toaKAZEzerocomma((unsigned long long)(MAXusedBuffer[k]*100)/((GRMBLhi11[(int)k] *
3113       LetterBuffer)/31), 11ToaDigits4, 10)+(26-2), "%0" ); // 26 are all 26-DESIRED=24
3114     }
3115     fprintf( fp_outLOG, "Used value for third parameter in KB: %lu\n", (unsigned long)Thunderwith );
3116     fprintf( fp_outLOG, "Leprechaun: Failure! Increment 'Memory for each Letter' parameter(third one)!\n\n");
3117     return( 1 );
3118 }
3119
3120 }
3121
3122 else // means USED-SLOT
3123 { FoundInLinkedList = 0;
3124   while (PseudoLinkedPointer != 0 && FoundInLinkedList == 0)
3125   {
3126     if (memcmp(PseudoLinkedPointer+4+4, wrd, wrdlen) == 0)
3127     {
3128       while ( --count && *(char *)buf1 == *(char *)buf2 ) {
3129         buf1 = (char *)buf1 + 1;
3130         buf2 = (char *)buf2 + 1;
3131       }
3132       return( *((unsigned char *)buf1) - *((unsigned char *)buf2) );
3133     }
3134     FoundInLinkedList = 1;
3135   }
3136   else // i.e < or >
3137   {
3138     if (memcmp(PseudoLinkedPointer+4+4, wrd, wrdlen) > 0)
3139     {
3140       memcpy( &PseudoLinkedPointerNEW, PseudoLinkedPointer, 4 );
3141     }
3142     else
3143     {
3144       PseudoLinkedPointer = PseudoLinkedPointer + 4;
3145       memcpy( &PseudoLinkedPointerNEW, PseudoLinkedPointer, 4 );
3146     }
3147   }
3148
3149   if (PseudoLinkedPointerNEW == 0)

```

```

3130     {
3131         //if( (unsigned long)(bufend[LetterOffset] - BufStart) + wrdlen + 4 < (GRMBLhll[(int)wrdlen] * LetterBuffer)/31 ) // OLD slower
3132         if( (unsigned long)(bufend[LetterOffset] - BufStart) + wrdlen + 4 + 4 < GRMBLFoolAgain[(int)wrdlen] )
3133             { memcpy( PseudoLinkedPointer, &bufend[LetterOffset], 4 );
3134               // Below 3 lines are commented due to experiment below malloc which shows that allocated memory is ZEROed.
3135               //memcpy( bufend[LetterOffset], &BufStart[NumberOfSLOTS*4], 4 ); // means next exists not
3136               //bufend[LetterOffset] = bufend[LetterOffset] + 4;
3137               //memcpy( bufend[LetterOffset], &BufStart[NumberOfSLOTS*4], 4 ); // means next exists not
3138               bufend[LetterOffset] = bufend[LetterOffset] + 4 + 4; // + 4 due to above commenting
3139               memcpy( bufend[LetterOffset], wrd, wrdlen ); WORDcountDistinct++; bufNumberOfWords[LetterOffset]++;
3140               //bufNowps[LetterOffset][slot]++; // ?! crashes
3141               bufend[LetterOffset] = bufend[LetterOffset] + wrdlen;
3142               if (MAXusedBuffer[wrdlen] < (unsigned long)(bufend[LetterOffset] - BufStart)) {MAXusedBuffer[wrdlen] = (unsigned
long)(bufend[LetterOffset] - BufStart);}
3143             }
3144         else
3145             { printf( "\nLeprechaun: Failure! Increment 'Memory for each Letter' parameter(third one)!\n" );
3146               fprintf( fp_outLOG, "Input File with a list of TEXTual Files: %s\n", argv[1] );
3147               fprintf( fp_outLOG, "Size of all TEXTual Files: %s\n", _ui64toaKAZEcomma(FilesLEN, llToaDigits, 10) );
3148               fprintf( fp_outLOG, "word count: %s of them %s distinct\n", _ui64toaKAZEcomma(WORDcount, llToaDigits, 10), _ui64toaKAZEcomma((unsigned long
long)WORDcountDistinct, llToaDigits2, 10) );
3149               fprintf( fp_outLOG, "Number Of Files: %lu\n", NumberOfFiles );
3150               fprintf( fp_outLOG, "Number Of Lines: %lu\n", NumberOfLines );
3151               fprintf( fp_outLOG, "Allocated memory in MB: %lu\n", (unsigned long)(memory_size>>20)+1 );
3152               fprintf( fp_outLOG, "Total Attempts to Find/Put WORDS into Binary-Search-Trees: %s\n", _ui64toaKAZEcomma(WORDcountAttemptsToPut, llToaDigits,
10) );
3153               for( k = 1; k < 32; k++ )
3154               { fprintf( fp_outLOG, "words with length %s occupy %sKB of %sKB given i.e. %s%s utilization\n", _ui64toaKAZEcomma(k, llToaDigits, 10)+(26-
2), _ui64toaKAZEcomma((MAXusedBuffer[k]>>10)+1, llToaDigits2, 10)+(26-5), _ui64toaKAZEcomma((((GRMBLhll[(int)k] *
LetterBuffer)/31)>>10)+1, llToaDigits3, 10)+(26-5), _ui64toaKAZEcomma((unsigned long long)(MAXusedBuffer[k]*100)/((GRMBLhll[(int)k] *
LetterBuffer)/31), llToaDigits4, 10)+(26-2), "%0" ); // 26 are all 26-DESIRED=24
3155             }
3156             fprintf( fp_outLOG, "Used value for third parameter in KB: %lu\n", (unsigned long)Thunderwith );
3157             fprintf( fp_outLOG, "Leprechaun: Failure! Increment 'Memory for each Letter' parameter(third one)!\n\n");
3158             return( 1 );
3159         }
3160     }
3161     PseudoLinkedPointer = PseudoLinkedPointerNEW;
3162 }
3163 WORDcountAttemptsToPut++;
3164 } // while
3165 }
3166 // @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ BST fragment ]
3167 } else
3168 {
3169 // ##### B-tree order 3 fragment [
3170 //
3171 // LEAF structure: [LeftPointer][MiddlePointer][RightPointer][LeftWord][RightWord]
3172 //                  4bytes      4bytes      4bytes      wrdlen      wrdlen
3173 //
3174 //                  <- if *(char *)==0 means the word cell is empty
3175 // ALL B-tree order 3 fragment consists of 3 sub-fragments:
3176 // 1] Search 2] if Search failed Trasirascht(pushing in stack PseudoLinkedPointer(visited LEAFs)) Search 3] Insert Iterative
3177 // 1] Search [ _____1407 line in C - see below: whole Search in assembler_____
3178 //             if (PseudoLinkedPointer == 0) // means EMPTY-SLOT
3179 //             {
3180 //             if( (unsigned long)(bufend[LetterOffset] - BufStart) + 2*wrdlen + 4 + 4 + 4 < GRMBLFoolAgain[(int)wrdlen] ) // +4 more for BST
instead of LL; + more(see LEAF)
3181 //             {
3182 //                 memcpy( BufStart+Slot, &bufend[LetterOffset], 4 );
3183 //                 bufend[LetterOffset] = bufend[LetterOffset] + 4 + 4 + 4; // + 4 due to above commenting
3184 //                 memcpy( bufend[LetterOffset], wrd, wrdlen ); WORDcountDistinct++; bufNumberOfWords[LetterOffset]++;
3185 //                 bufend[LetterOffset] = bufend[LetterOffset] + 2*wrdlen;
3186 //                 if (MAXusedBuffer[wrdlen] < (unsigned long)(bufend[LetterOffset] - BufStart)) {MAXusedBuffer[wrdlen] = (unsigned
long)(bufend[LetterOffset] - BufStart);}
3187 //             }
3188 //             else
3189 //             { printf( "\nLeprechaun: Failure! Increment 'Memory for each Letter' parameter(third one)!\n" );
3190 //               fprintf( fp_outLOG, "Input File with a list of TEXTual Files: %s\n", argv[1] );
3191 //               fprintf( fp_outLOG, "Size of all TEXTual Files: %s\n", _ui64toaKAZEcomma(FilesLEN, llToaDigits, 10) );
3192 //               fprintf( fp_outLOG, "word count: %s of them %s distinct\n", _ui64toaKAZEcomma(WORDcount, llToaDigits, 10), _ui64toaKAZEcomma((unsigned long
long)WORDcountDistinct, llToaDigits2, 10) );
3193 //               fprintf( fp_outLOG, "Number Of Files: %lu\n", NumberOfFiles );
3194 //               fprintf( fp_outLOG, "Number Of Lines: %lu\n", NumberOfLines );
3195 //               fprintf( fp_outLOG, "Allocated memory in MB: %lu\n", (unsigned long)(memory_size>>20)+1 );
3196 //               fprintf( fp_outLOG, "Total Attempts to Find/Put WORDS into B-trees order 3: %s\n", _ui64toaKAZEcomma(WORDcountAttemptsToPut, llToaDigits, 10)
);
3197 //               for( k = 1; k < 32; k++ )
3198 //               { fprintf( fp_outLOG, "words with length %s occupy %sKB of %sKB given i.e. %s%s utilization\n", _ui64toaKAZEcomma(k, llToaDigits, 10)+(26-
2), _ui64toaKAZEcomma((MAXusedBuffer[k]>>10)+1, llToaDigits2, 10)+(26-5), _ui64toaKAZEcomma((((GRMBLhll[(int)k] *
LetterBuffer)/31)>>10)+1, llToaDigits3, 10)+(26-5), _ui64toaKAZEcomma((unsigned long long)(MAXusedBuffer[k]*100)/((GRMBLhll[(int)k] *
LetterBuffer)/31), llToaDigits4, 10)+(26-2), "%0" ); // 26 are all 26-DESIRED=24
3199 //               }
3200 //               fprintf( fp_outLOG, "Used value for third parameter in KB: %lu\n", (unsigned long)Thunderwith );
3201 //               fprintf( fp_outLOG, "Leprechaun: Failure! Increment 'Memory for each Letter' parameter(third one)!\n\n");
3202 //               return( 1 );
3203 //             }
3204             FoundInLinkedList = 1;

```

```

3205     }
3206     else // means USED-SLOT
3207     { FoundInLinkedList = 0;
3208       while (PseudoLinkedPointer != 0 && FoundInLinkedList == 0)
3209       {
3210         // ***** 'P W P' section [
3211         // LW: existence check if ( *(char *) (PseudoLinkedPointer+4+4+4) != 0 )
3212         // RW: existence check if ( *(char *) (PseudoLinkedPointer+4+4+4+wordlen) != 0 )
3213         // here ALWAYS LW exists: no need for existence check - line below
3214         // if ( *(char *) (PseudoLinkedPointer+4+4+4) != 0 )
3215         if (memcmp(PseudoLinkedPointer+4+4+4, wrd, wrdlen) > 0) // go LP
3216         { memcpy( &PseudoLinkedPointerNEW, PseudoLinkedPointer + 0, 4 ); //LP
3217           PseudoLinkedPointer = PseudoLinkedPointerNEW;
3218         }
3219         else if (memcmp(PseudoLinkedPointer+4+4+4, wrd, wrdlen) < 0) // go RP or MP
3220         { // RW existence check - line below:
3221           if ( *(char *) (PseudoLinkedPointer+4+4+4+wordlen) != 0 ) // RW exists
3222             { // Here all 'P W P' section is repeated; the way of handling case when dynamic number of words in leaf
3223               // ++++++
3224               // ***** 'P W P' section 2 [
3225               // LW: existence check if ( *(char *) (PseudoLinkedPointer+4+4+4) != 0 )
3226               // RW: existence check if ( *(char *) (PseudoLinkedPointer+4+4+4+wordlen) != 0 )
3227               // here ALWAYS RW exists: no need for existence check - line below
3228               // if ( *(char *) (PseudoLinkedPointer+4+4+4+wordlen) != 0 )
3229               if (memcmp(PseudoLinkedPointer+4+4+4+wordlen, wrd, wrdlen) > 0) // go MP
3230               { memcpy( &PseudoLinkedPointerNEW, PseudoLinkedPointer + 4, 4 ); //MP
3231                 PseudoLinkedPointer = PseudoLinkedPointerNEW;
3232               }
3233               else if (memcmp(PseudoLinkedPointer+4+4+4+wordlen, wrd, wrdlen) < 0) // go RP
3234               { // No ?w after RW - go RP
3235                 memcpy( &PseudoLinkedPointerNEW, PseudoLinkedPointer + 4 + 4, 4 ); //RP
3236                 PseudoLinkedPointer = PseudoLinkedPointerNEW;
3237               }
3238               else FoundInLinkedList = 1; // wrd is RW
3239               WORDcountAttemptsToPut++;
3240             } // ***** 'P W P' section 2 ]
3241             // ++++++
3242             }
3243             else // RW empty - go MP
3244             { memcpy( &PseudoLinkedPointerNEW, PseudoLinkedPointer + 4, 4 ); //MP
3245               PseudoLinkedPointer = PseudoLinkedPointerNEW;
3246             }
3247             }
3248             else FoundInLinkedList = 1; // wrd is LW
3249             WORDcountAttemptsToPut++;
3250           } // ***** 'P W P' section ]
3251           } // while
3252           WORDcountAttemptsToPut--; // - 1 due to BST way of counting i.e. direct hash hit is not counted only successors
3253         }
3254       // 1] Search ] _____1484 line in C - see below: whole Search in assembler_____
3255
3256       /*
3257       ; Line 1397
3258       jmp $L2139
3259       $L2042:
3260       ; Line 1408
3261       test edx, edx
3262       jne SHORT $L2110
3263       ; Line 1410
3264       mov ecx, DWORD PTR _bufend$[esp+esi*4+892340]
3265       mov edi, DWORD PTR _GRMBLFoolAgain$[esp+ebx*4+892340]
3266       lea edx, DWORD PTR _bufend$[esp+esi*4+892340]
3267       lea esi, DWORD PTR [ebx+ebx+12]
3268       sub esi, ebp
3269       add esi, ecx
3270       cmp esi, edi
3271       mov DWORD PTR tv4122[esp+892340], edx
3272       jae $L2113
3273       ; Line 1412
3274       mov DWORD PTR [eax+ebp], ecx
3275       ; Line 1413
3276       lea eax, DWORD PTR [ecx+12]
3277       ; Line 1436
3278       jmp $L2749
3279       $L2110:
3280       ; Line 1437
3281       mov DWORD PTR _FoundInLinkedList$[esp+892340], 0
3282       npad 11
3283       $L2141:
3284       ; Line 1438
3285       mov eax, DWORD PTR _FoundInLinkedList$[esp+892340]
3286       test eax, eax
3287       jne $L2142
3288       ; Line 1445
3289       lea ebp, DWORD PTR [edx+12]
3290       mov ecx, ebx
3291       lea edi, DWORD PTR _wrd$[esp+892340]
3292       mov esi, ebp

```

```

3293 xor eax, eax
3294 repe cmpsb
3295 je SHORT $L2682
3296 sbb eax, eax
3297 sbb eax, -1
3298 $L2682:
3299 test eax, eax
3300 jle SHORT $L2143
3301 ; Line 1447
3302 mov edx, DWORD PTR [edx]
3303 ; Line 1449
3304 jmp $L2153
3305 $L2143:
3306 mov ecx, ebx
3307 lea edi, DWORD PTR _ wrd$[esp+892340]
3308 mov esi, ebp
3309 xor eax, eax
3310 repe cmpsb
3311 je SHORT $L2640
3312 sbb eax, eax
3313 sbb eax, -1
3314 $L2640:
3315 test eax, eax
3316 jge SHORT $L2145
3317 ; Line 1451
3318 mov cl, BYTE PTR [edx+ebx+12]
3319 test cl, cl
3320 lea eax, DWORD PTR [edx+ebx+12]
3321 je SHORT $L2147
3322 ; Line 1459
3323 mov ecx, ebx
3324 lea edi, DWORD PTR _ wrd$[esp+892340]
3325 mov esi, eax
3326 xor ebp, ebp
3327 repe cmpsb
3328 je SHORT $L2695
3329 sbb ebp, ebp
3330 sbb ebp, -1
3331 $L2695:
3332 test ebp, ebp
3333 jle SHORT $L2148
3334 ; Line 1461
3335 mov edx, DWORD PTR [edx+4]
3336 ; Line 1463
3337 jmp SHORT $L2151
3338 $L2148:
3339 mov esi, eax
3340 mov ecx, ebx
3341 lea edi, DWORD PTR _ wrd$[esp+892340]
3342 xor eax, eax
3343 repe cmpsb
3344 je SHORT $L2642
3345 sbb eax, eax
3346 sbb eax, -1
3347 $L2642:
3348 test eax, eax
3349 jge SHORT $L2150
3350 ; Line 1466
3351 mov edx, DWORD PTR [edx+8]
3352 ; Line 1468
3353 jmp SHORT $L2151
3354 $L2150:
3355 mov DWORD PTR _FoundInLinkedList$[esp+892340], 1
3356 $L2151:
3357 ; Line 1469
3358 mov ecx, DWORD PTR _WORDcountAttemptsToPut$[esp+892340]
3359 mov eax, DWORD PTR _WORDcountAttemptsToPut$[esp+892344]
3360 add ecx, 1
3361 adc eax, 0
3362 mov DWORD PTR _WORDcountAttemptsToPut$[esp+892340], ecx
3363 mov DWORD PTR _WORDcountAttemptsToPut$[esp+892344], eax
3364 ; Line 1473
3365 jmp SHORT $L2153
3366 $L2147:
3367 ; Line 1475
3368 mov edx, DWORD PTR [edx+4]
3369 ; Line 1478
3370 jmp SHORT $L2153
3371 $L2145:
3372 mov DWORD PTR _FoundInLinkedList$[esp+892340], 1
3373 $L2153:
3374 ; Line 1479
3375 mov esi, DWORD PTR _WORDcountAttemptsToPut$[esp+892340]
3376 mov ecx, DWORD PTR _WORDcountAttemptsToPut$[esp+892344]
3377 add esi, 1
3378 adc ecx, 0
3379 test edx, edx
3380 mov DWORD PTR _WORDcountAttemptsToPut$[esp+892340], esi

```



```

3381 mov DWORD PTR _WORDcountAttemptsToPut$[esp+892344], ecx
3382 jne $L2141
3383 $L2142:
3384 ; Line 1482
3385 mov edx, DWORD PTR _WORDcountAttemptsToPut$[esp+892340]
3386 mov ecx, DWORD PTR _WORDcountAttemptsToPut$[esp+892344]
3387 or  eax, -1
3388 add  edx, eax
3389 adc  ecx, eax
3390 mov DWORD PTR _WORDcountAttemptsToPut$[esp+892344], ecx
3391 mov DWORD PTR _WORDcountAttemptsToPut$[esp+892340], edx
3392 $L2139:
3393 */
3394
3395 if (FoundInLinkedList == 0)
3396 {
3397 // 2] if Search failed Trasirascht(pushing in stack PseudoLinkedPointer(visited LEAFs)) Search [
3398 // 'TracingSearch' is the same as 'Search' except that adds the trail in my simulated stack,
3399 // the goal is not to waste time in 'Search' by dealing with no needed trail in case of not 'Insert'.
3400 // Simulated stack contains pairs of 'Address of ParentLEAF' + 'Offset of ParentPointer in ParentLEAF i.e. 0 for LP, 4 for MP, 8 for RP'.
3401 // 'Offset ...' saves unnecessary comparisons of NEWword which after splitting goes up.
3402     memcpy( &PseudoLinkedPointer, BufStart+Slot, 4 );
3403     stackPtr = 0;
3404     while (PseudoLinkedPointer != 0)
3405     {
3406 // ***** 'P W P' section [
3407 // LW: existence check if ( *(char *) (PseudoLinkedPointer+4+4+4) != 0 )
3408 // RW: existence check if ( *(char *) (PseudoLinkedPointer+4+4+4+wordlen) != 0 )
3409 // here ALWAYS LW exists: no need for existence check - line below
3410 // if ( *(char *) (PseudoLinkedPointer+4+4+4) != 0 )
3411 // if (memcmp(PseudoLinkedPointer+4+4+4, wrd, wrdlen) > 0) // go LP
3412 // { memcpy( &PseudoLinkedPointerNEW, PseudoLinkedPointer + 0, 4 ); //LP
3413 // if (StackPtr > 8192*3-1-1) { printf( "\nLeprechaun: Failure! 'B-tree order 3' simulated stack overflow!\n" ); return( 13 );}
3414 // BSTstack[StackPtr] = PseudoLinkedPointer; ++StackPtr; //pt to visited leaf
3415 // BSTstack[StackPtr] = 0; ++StackPtr; //LPoffset=0;MPoffset=4;RPOffset=8;
3416 // PseudoLinkedPointer = PseudoLinkedPointerNEW;
3417 // }
3418 // else if (memcmp(PseudoLinkedPointer+4+4+4, wrd, wrdlen) < 0) // go RP or MP
3419 // { // RW existence check - line below:
3420 // if ( *(char *) (PseudoLinkedPointer+4+4+4+wordlen) != 0 ) // RW exists
3421 // { // Here all 'P W P' section is repeated; the way of handling case when dynamic number of words in leaf
3422 // ++++++
3423 // ***** 'P W P' section 2 [
3424 // LW: existence check if ( *(char *) (PseudoLinkedPointer+4+4+4) != 0 )
3425 // RW: existence check if ( *(char *) (PseudoLinkedPointer+4+4+4+wordlen) != 0 )
3426 // here ALWAYS RW exists: no need for existence check - line below
3427 // if ( *(char *) (PseudoLinkedPointer+4+4+4+wordlen) != 0 )
3428 // if (memcmp(PseudoLinkedPointer+4+4+4+wordlen, wrd, wrdlen) > 0) // go MP
3429 // { memcpy( &PseudoLinkedPointerNEW, PseudoLinkedPointer + 4, 4 ); //MP
3430 // if (StackPtr > 8192*3-1-1) { printf( "\nLeprechaun: Failure! 'B-tree order 3' simulated stack overflow!\n" ); return( 13 );}
3431 // BSTstack[StackPtr] = PseudoLinkedPointer; ++StackPtr; //pt to visited leaf
3432 // BSTstack[StackPtr] = 4; ++StackPtr; //LPoffset=0;MPoffset=4;RPOffset=8;
3433 // PseudoLinkedPointer = PseudoLinkedPointerNEW;
3434 // }
3435 // else if (memcmp(PseudoLinkedPointer+4+4+4+wordlen, wrd, wrdlen) < 0) // go RP
3436 // { // No ?w after RW - go RP
3437 // memcpy( &PseudoLinkedPointerNEW, PseudoLinkedPointer + 4 + 4, 4 ); //RP
3438 // if (StackPtr > 8192*3-1-1) { printf( "\nLeprechaun: Failure! 'B-tree order 3' simulated stack overflow!\n" ); return( 13 );}
3439 // BSTstack[StackPtr] = PseudoLinkedPointer; ++StackPtr; //pt to visited leaf
3440 // BSTstack[StackPtr] = 8; ++StackPtr; //LPoffset=0;MPoffset=4;RPOffset=8;
3441 // PseudoLinkedPointer = PseudoLinkedPointerNEW;
3442 // }
3443 // else FoundInLinkedList = 1; // wrd is RW
3444 // ***** 'P W P' section 2 ]
3445 // ++++++
3446 // }
3447 // else // RW empty - go MP
3448 // { memcpy( &PseudoLinkedPointerNEW, PseudoLinkedPointer + 4, 4 ); //MP
3449 // if (StackPtr > 8192*3-1-1) { printf( "\nLeprechaun: Failure! 'B-tree order 3' simulated stack overflow!\n" ); return( 13 );}
3450 // BSTstack[StackPtr] = PseudoLinkedPointer; ++StackPtr; //pt to visited leaf
3451 // BSTstack[StackPtr] = 4; ++StackPtr; //LPoffset=0;MPoffset=4;RPOffset=8;
3452 // PseudoLinkedPointer = PseudoLinkedPointerNEW;
3453 // }
3454 // }
3455 // else FoundInLinkedList = 1; // wrd is LW
3456 // ***** 'P W P' section ]
3457 // } // while
3458 // 2] if Search failed Trasirascht(pushing in stack PseudoLinkedPointer(visited LEAFs)) Search ]
3459
3460 // 3] Insert Iterative [
3461 // There are total 4 situations:
3462 // Case #1: Outer NODE(including ROOT) [ ][ ][ ][LW][ ]
3463 // Case #2: Outer NODE(including ROOT) [ ][ ][ ][LW][RW] Split occurs ----
3464 // Case #3: ROOT [LP][MP][ ][LW][ ] 'wrdUP' (wrdlen bytes)
3465 // Case #4: Inner NODE(including ROOT) [LP][MP][RP][LW][RW] Split occurs --- | &
3466 // | | 'PseudoLinkedPointerNEW' (ptr to NEW LEAF)
3467 // There are total 2 situations for PARENT LEAF: <----- ARE GOING UP
3468 // Case #3: [LP][MP][ ][LW][ ]

```

```

3469 // Case #4: [LP][MP][RP][LW][RW] Split Occurs
3470
3471 // ~ First deal alonely with the OUTER NODE(LEAF) where Search stopped i.e Case #1 & Case #2:
3472 POffsetInLEAF = BSTstack[--StackPtr];
3473 PseudoLinkedPointer = BSTstack[--StackPtr];
3474 // NOTE: ONE LEAF IS FULL ONLY WHEN LAST CELL FOR KEY(here RW) EXISTS!
3475 // RW: existence check if ( *(char *) (PseudoLinkedPointer+4+4+4+wordlen) != 0 )
3476 if ( *(char *) (PseudoLinkedPointer+4+4+4+wordlen) != 0 ) // If LEAF is full: Case #2
3477 { SplitOccured = 1; WORDcountDistinct++; bufNumberOfWords[LetterOffset]++;
3478 // ALlocate NEW LEAF:
3479 if( (unsigned long)(bufend[LetterOffset] - BufStart) + 2*wordlen + 4 + 4 + 4 < GRMBLFoolAgain[(int)wordlen] ) // +4 more for BST
instead of LL; + more (see LEAF)
3480 {
3481 memcpy( &PseudoLinkedPointerNEW, &bufend[LetterOffset], 4 );
3482 bufend[LetterOffset] = bufend[LetterOffset] + 4 + 4 + 4; // + 4 due to above commenting
3483 bufend[LetterOffset] = bufend[LetterOffset] + 2*wordlen;
3484 if (MAXusedBuffer[wordlen] < (unsigned long)(bufend[LetterOffset] - BufStart)) {MAXusedBuffer[wordlen] = (unsigned
long)(bufend[LetterOffset] - BufStart);}
3485 }
3486 else
3487 { printf( "\nLeprechaun: Failure! Increment 'Memory for each Letter' parameter(third one)!\n" );
3488 fprintf( fp_outLOG, "Input File with a list of TEXTual Files: %s\n", argv[1] );
3489 fprintf( fp_outLOG, "Size of all TEXTual Files: %s\n", _ui64toaKAZEcomma(FilesLEN, llToaDigits, 10) );
3490 fprintf( fp_outLOG, "word count: %s of them %s distinct\n", _ui64toaKAZEcomma(WORDcount, llToaDigits, 10), _ui64toaKAZEcomma((unsigned long
long)WORDcountDistinct, llToaDigits2, 10) );
3491 fprintf( fp_outLOG, "Number Of Files: %lu\n", NumberOfFiles );
3492 fprintf( fp_outLOG, "Number Of Lines: %lu\n", NumberOfLines );
3493 fprintf( fp_outLOG, "Allocated memory in MB: %lu\n", (unsigned long)(memory_size>>20)+1 );
3494 fprintf( fp_outLOG, "Total Attempts to Find/Put WORDs into B-trees order 3: %s\n", _ui64toaKAZEcomma(WORDcountAttemptsToPut, llToaDigits, 10)
);
3495 for( k = 1; k < 32; k++ )
3496 { fprintf( fp_outLOG, "words with length %s occupy %sKB of %sKB given i.e. %s%s utilization\n", _ui64toaKAZEzerocomma(k, llToaDigits, 10)+(26-
2), _ui64toaKAZEzerocomma((MAXusedBuffer[k]>>10)+1, llToaDigits2, 10)+(26-5), _ui64toaKAZEzerocomma((((GRMBLhill[(int)k] *
LetterBuffer)/31)>>10)+1, llToaDigits3, 10)+(26-5), _ui64toaKAZEzerocomma((unsigned long long)(MAXusedBuffer[k]*100)/((GRMBLhill[(int)k] *
LetterBuffer)/31), llToaDigits4, 10)+(26-2), "%0" ); // 26 are all 26-DESIRED=24
3497 }
3498 fprintf( fp_outLOG, "Used value for third parameter in KB: %lu\n", (unsigned long)Thunderwith );
3499 fprintf( fp_outLOG, "Leprechaun: Failure! Increment 'Memory for each Letter' parameter(third one)!\n\n");
3500 return( 1 );
3501 }
3502 if (POffsetInLEAF == 0) // wrd < LW
3503 {
3504 memcpy( wrdUP, PseudoLinkedPointer+4+4+4, wordlen ); // LW up
3505 memcpy( PseudoLinkedPointer+4+4+4, wrd, wordlen ); // wrd go to OLD LEAF
3506 memcpy( PseudoLinkedPointerNEW+4+4+4, PseudoLinkedPointer+4+4+4+wordlen, wordlen ); // RW go to NEW LEAF
3507 *(char *) (PseudoLinkedPointer+4+4+4+wordlen) = 0; // RW mark unused in OLD LEAF
3508 }
3509 if (POffsetInLEAF == 4) // LW < wrd < RW
3510 {
3511 memcpy( wrdUP, wrd, wordlen ); // wrd up
3512 memcpy( PseudoLinkedPointerNEW+4+4+4, PseudoLinkedPointer+4+4+4+wordlen, wordlen ); // RW go to NEW LEAF
3513 *(char *) (PseudoLinkedPointer+4+4+4+wordlen) = 0; // RW mark unused in OLD LEAF
3514 }
3515 if (POffsetInLEAF == 8) // wrd > RW
3516 {
3517 memcpy( wrdUP, PseudoLinkedPointer+4+4+4+wordlen, wordlen ); // RW up
3518 *(char *) (PseudoLinkedPointer+4+4+4+wordlen) = 0; // RW mark unused in OLD LEAF
3519 memcpy( PseudoLinkedPointerNEW+4+4+4, wrd, wordlen ); // wrd go to NEW LEAF
3520 }
3521 }
3522 else // If LEAF is not full: Case #1
3523 { SplitOccured = 0; WORDcountDistinct++; bufNumberOfWords[LetterOffset]++;
3524 if (POffsetInLEAF == 0) // wrd < [LW][ ] so [LW][ ] -> [ ][LW] -> [wrd][LW]
3525 {
3526 memcpy( PseudoLinkedPointer+4+4+4+wordlen, PseudoLinkedPointer+4+4+4, wordlen );
3527 memcpy( PseudoLinkedPointer+4+4+4, wrd, wordlen );
3528 }
3529 if (POffsetInLEAF == 4) // wrd > [LW][ ] so [LW][ ] -> [LW][wrd]
3530 {
3531 memcpy( PseudoLinkedPointer+4+4+4+wordlen, wrd, wordlen );
3532 }
3533 }
3534
3535 if (SplitOccured != 0)
3536 {
3537 // ~ Second deal with the INNER NODE(S) i.e Case #3 & Case #4:
3538 while (StackPtr != 0 || SplitOccured != 0)
3539 {
3540 // 'PseudoLinkedPointerNEW' is new LEAF to be inserted
3541 // 'wrdUP' is NEW word to be inserted
3542 if (StackPtr != 0)
3543 {
3544 POffsetInLEAF = BSTstack[--StackPtr];
3545 PseudoLinkedPointer = BSTstack[--StackPtr];
3546 if ( *(char *) (PseudoLinkedPointer+4+4+4+wordlen) != 0 ) // If LEAF is full: Case #4
3547 { SplitOccured = 1;
3548 memcpy( wrdUPold, wrdUP, wordlen ); // LW up
3549 PseudoLinkedPointerNEWold = PseudoLinkedPointerNEW;

```

```

3550 // ALlocate NEW LEAF:
3551 if( (unsigned long)(bufend[LetterOffset] - BufStart) + 2*wordlen + 4 + 4 + 4 < GRMBLFoolAgain[(int)wordlen] ) // +4 more for BST
instead of LL; + more(see LEAF)
3552 {
3553     memcpy( &PseudoLinkedPointerNEW, &bufend[LetterOffset], 4 );
3554     bufend[LetterOffset] = bufend[LetterOffset] + 4 + 4 + 4; // + 4 due to above commenting
3555     bufend[LetterOffset] = bufend[LetterOffset] + 2*wordlen;
3556     if (MAXusedBuffer[wordlen] < (unsigned long)(bufend[LetterOffset] - BufStart)) {MAXusedBuffer[wordlen] = (unsigned
long)(bufend[LetterOffset] - BufStart);}
3557 }
3558 else
3559 { printf( "\nLeprechaun: Failure! Increment 'Memory for each Letter' parameter(third one)!\n" );
3560 fprintf( fp_outLOG, "Input File with a list of TEXTual Files: %s\n", argv[1] );
3561 fprintf( fp_outLOG, "Size of all TEXTual Files: %s\n", _ui64toaKAZEcomma(FilesLEN, 11ToaDigits, 10) );
3562 fprintf( fp_outLOG, "Word count: %s of them %s distinct\n", _ui64toaKAZEcomma(WORDcount, 11ToaDigits, 10), _ui64toaKAZEcomma((unsigned long
long)WORDcountDistinct, 11ToaDigits2, 10) );
3563 fprintf( fp_outLOG, "Number Of Files: %lu\n", NumberOfFiles );
3564 fprintf( fp_outLOG, "Number Of Lines: %lu\n", NumberOfLines );
3565 fprintf( fp_outLOG, "Allocated memory in MB: %lu\n", (unsigned long)(memory_size>>20)+1 );
3566 fprintf( fp_outLOG, "Total Attempts to Find/Put WORDS into B-trees order 3: %s\n", _ui64toaKAZEcomma(WORDcountAttemptsToPut, 11ToaDigits, 10)
);
3567 for( k = 1; k < 32; k++ )
3568 { fprintf( fp_outLOG, "Words with length %s occupy %sKB of %sKB given i.e. %s utilization\n", _ui64toaKAZEzerocomma(k, 11ToaDigits, 10)+(26-
2), _ui64toaKAZEzerocomma((MAXusedBuffer[k]>>10)+1, 11ToaDigits2, 10)+(26-5), _ui64toaKAZEzerocomma((((GRMBLh11[(int)k] *
LetterBuffer)/31)>>10)+1, 11ToaDigits3, 10)+(26-5), _ui64toaKAZEzerocomma((unsigned long long)(MAXusedBuffer[k]*100)/((GRMBLh11[(int)k] *
LetterBuffer)/31), 11ToaDigits4, 10)+(26-2), "%0" ); // 26 are all 26-DESIRED=24
3569 }
3570 fprintf( fp_outLOG, "Used value for third parameter in KB: %lu\n", (unsigned long)Thunderwith );
3571 fprintf( fp_outLOG, "Leprechaun: Failure! Increment 'Memory for each Letter' parameter(third one)!\n\n");
3572 return( 1 );
3573 }
3574 if (PoffsetInLEAF == 0) // wrdUPold < LW
3575 {
3576     memcpy( wrdUP, PseudoLinkedPointer+4+4+4, wrdlen ); // LW up
3577     memcpy( PseudoLinkedPointer+4+4+4, wrdUPold, wrdlen ); // wrdUPold go to OLD LEAF
3578     memcpy( PseudoLinkedPointerNEW+4+4+4, PseudoLinkedPointer+4+4+4+wrdlen, wrdlen ); // RW go to NEW LEAF
3579     *(char *) (PseudoLinkedPointer+4+4+4+wrdlen) = 0; // RW mark unused in OLD LEAF
3580     // [LP](PseudoLinkedPointerNEWold)[MP][RP](wrdUPold)[LW][RW] -----
3581     // pair [LW] PseudoLinkedPointerNEW goes up |
3582     // PseudoLinkedPointer: PseudoLinkedPointerNEW: |
3583     // [LP](PseudoLinkedPointerNEWold)[ ](wrdUPold) [MP][RP][ ] [RW] <----
3584     // no need to put zero in RP because logic is based on words existence:
3585     memcpy( PseudoLinkedPointerNEW+0, PseudoLinkedPointer+4, 4 );
3586     memcpy( PseudoLinkedPointerNEW+4, PseudoLinkedPointer+8, 4 );
3587     memcpy( PseudoLinkedPointer+4, &PseudoLinkedPointerNEWold, 4 );
3588 }
3589 if (PoffsetInLEAF == 4) // LW < wrdUPold < RW
3590 {
3591     memcpy( wrdUP, wrdUPold, wrdlen ); // wrdUPold up
3592     memcpy( PseudoLinkedPointerNEW+4+4+4, PseudoLinkedPointer+4+4+4+wrdlen, wrdlen ); // RW go to NEW LEAF
3593     *(char *) (PseudoLinkedPointer+4+4+4+wrdlen) = 0; // RW mark unused in OLD LEAF
3594     // [LP][MP](PseudoLinkedPointerNEWold)[RP][LW](wrdUPold)[RW] -----
3595     // pair [wrdUPold] PseudoLinkedPointerNEW goes up |
3596     // PseudoLinkedPointer: PseudoLinkedPointerNEW: |
3597     // [LP][MP][ ] [LW] (PseudoLinkedPointerNEWold)[RP][ ] [RW] <----
3598     // no need to put zero in RP because logic is based on words existence:
3599     memcpy( PseudoLinkedPointerNEW+0, &PseudoLinkedPointerNEWold, 4 );
3600     memcpy( PseudoLinkedPointerNEW+4, PseudoLinkedPointer+8, 4 );
3601 }
3602 if (PoffsetInLEAF == 8) // wrdUPold > RW
3603 {
3604     memcpy( wrdUP, PseudoLinkedPointer+4+4+4+wrdlen, wrdlen ); // RW up
3605     *(char *) (PseudoLinkedPointer+4+4+4+wrdlen) = 0; // RW mark unused in OLD LEAF
3606     memcpy( PseudoLinkedPointerNEW+4+4+4, wrdUPold, wrdlen ); // wrdUPold go to NEW LEAF
3607     // [LP][MP][RP](PseudoLinkedPointerNEWold)[LW][RW](wrdUPold) -----
3608     // pair [RW] PseudoLinkedPointerNEW goes up |
3609     // PseudoLinkedPointer: PseudoLinkedPointerNEW: |
3610     // [LP][MP][ ] [LW] [RP](PseudoLinkedPointerNEWold)[ ](wrdUPold) <---
3611     // no need to put zero in RP because logic is based on words existence:
3612     memcpy( PseudoLinkedPointerNEW+0, PseudoLinkedPointer+8, 4 );
3613     memcpy( PseudoLinkedPointerNEW+4, &PseudoLinkedPointerNEWold, 4 );
3614 }
3615 }
3616 else // If LEAF is not full: Case #3
3617 { SplitOccured = 0;
3618     if (PoffsetInLEAF == 0) // wrdUP < [LW][ ] so [LW][ ] -> [ ] [LW] -> [wrdUP][LW]
3619     {
3620         memcpy( PseudoLinkedPointer+4+4+4+wrdlen, PseudoLinkedPointer+4+4+4, wrdlen );
3621         memcpy( PseudoLinkedPointer+4+4+4, wrdUP, wrdlen );
3622         // [LP][MP][ ] -> [LP][ ] [MP] -> [LP][np][MP]
3623         memcpy( PseudoLinkedPointer+8, PseudoLinkedPointer+4, 4 );
3624         memcpy( PseudoLinkedPointer+4, &PseudoLinkedPointerNEW, 4 );
3625     }
3626     if (PoffsetInLEAF == 4) // wrdUP > [LW][ ] so [LW][ ] -> [LW][wrdUP]
3627     {
3628         memcpy( PseudoLinkedPointer+4+4+4+wrdlen, wrdUP, wrdlen );
3629         // [LP][MP][ ] -> [LP][MP][np]
3630         memcpy( PseudoLinkedPointer+8, &PseudoLinkedPointerNEW, 4 );
3631     }
3632 }

```

```

3631     }
3632     break;
3633 }
3634 }
3635 else // Empty stack means ROOT and more over ROOT is already splitted(Case #4 is off)
3636 {
3637     // If LEAF is not full: Case #3
3638     // THIS IS WHERE A NEW(SECOND) LEAF 'PseudoLinkedPointerROOT' must be allocated:
3639     if( (unsigned long)(bufend[LetterOffset] - BufStart) + 2*wordlen + 4 + 4 + 4 < GRMBLFoolAgain[(int)wordlen] ) // +4 more for BST
instead of LL; + more(see LEAF)
3640     {
3641         memcpy( &PseudoLinkedPointerROOT, &bufend[LetterOffset], 4 );
3642         bufend[LetterOffset] = bufend[LetterOffset] + 4 + 4 + 4; // + 4 due to above commenting
3643         memcpy( bufend[LetterOffset], wrdUP, wordlen );
3644         bufend[LetterOffset] = bufend[LetterOffset] + 2*wordlen;
3645         if (MAXusedBuffer[wordlen] < (unsigned long)(bufend[LetterOffset] - BufStart)) {MAXusedBuffer[wordlen] = (unsigned
long)(bufend[LetterOffset] - BufStart);}
3646     }
3647     else
3648     { printf( "\nLeprechaun: Failure! Increment 'Memory for each Letter' parameter(third one)!\n" );
3649     fprintf( fp_outLOG, "Input File with a list of TEXTual Files: %s\n", argv[1] );
3650     fprintf( fp_outLOG, "Size of all TEXTual Files: %s\n", _ui64toaKAZEcomma(FilesLEN, l1ToADigits, 10) );
3651     fprintf( fp_outLOG, "Word count: %s of them %s distinct\n", _ui64toaKAZEcomma(WORDcount, l1ToADigits, 10), _ui64toaKAZEcomma((unsigned long
long)WORDcountDistinct, l1ToADigits2, 10) );
3652     fprintf( fp_outLOG, "Number Of Files: %lu\n", NumberOfFiles );
3653     fprintf( fp_outLOG, "Number Of Lines: %lu\n", NumberOfLines );
3654     fprintf( fp_outLOG, "Allocated memory in MB: %lu\n", (unsigned long)(memory_size>>20)+1 );
3655     fprintf( fp_outLOG, "Total Attempts to Find/Put WORDS into B-trees order 3: %s\n", _ui64toaKAZEcomma(WORDcountAttemptsToPut, l1ToADigits, 10)
);
3656     for( k = 1; k < 32; k++ )
3657     { fprintf( fp_outLOG, "words with length %s occupy %sKB of %sKB given i.e. %s utilization\n", _ui64toaKAZEcomma(k, l1ToADigits, 10)+(26-
2), _ui64toaKAZEcomma((MAXusedBuffer[k]>>10)+1, l1ToADigits2, 10)+(26-5), _ui64toaKAZEcomma((((GRMBLhill[(int)k] *
LetterBuffer)/31)>>10)+1, l1ToADigits3, 10)+(26-5), _ui64toaKAZEcomma((unsigned long long)(MAXusedBuffer[k]*100)/((GRMBLhill[(int)k] *
LetterBuffer)/31), l1ToADigits4, 10)+(26-2), "%0" ); // 26 are all 26-DESIRED=24
3658     }
3659     fprintf( fp_outLOG, "used value for third parameter in KB: %lu\n", (unsigned long)Thunderwith );
3660     fprintf( fp_outLOG, "Leprechaun: Failure! Increment 'Memory for each Letter' parameter(third one)!\n\n");
3661     return( 1 );
3662     }
3663     // Here -- 'PseudoLinkedPointerROOT' --
3664     // | (wrdUP) |
3665     // 'PseudoLinkedPointer' <-- --> 'PseudoLinkedPointerNEW'
3666     // (LW) (RW)
3667     memcpy( PseudoLinkedPointerROOT, &PseudoLinkedPointer, 4 ); // LP
3668     memcpy( PseudoLinkedPointerROOT+4, &PseudoLinkedPointerNEW, 4 ); // MP
3669     // Here must NEW ROOT be updated i.e. HASH table(SLOT) must point it:
3670     memcpy( BufStart+Slot, &PseudoLinkedPointerROOT, 4 );
3671     break; //because it is ROOT without split
3672 }
3673 } // while
3674 } //if (SplitOccured != 0)
3675 // 3] Insert Iterative ]
3676 } //if (FoundInLinkedList == 0)
3677 // ##### B-tree order 3 ]
3678 } //if (BSTorBtree == 0)
3679 } else { //if (BSTorBtree != 2) {
3680 // External Btrees [
3681
3682 // - ASCII code 095
3683 // - ASCII code 096 \
3684 // a ASCII code 097 / In total 26+1+1 radix instead of 27 to avoid +1 for each '-', code 096 not used.
3685 // z ASCII code 122
3686 // The hash for 'a quadruplet_for_example' will be calculated for first 5 chars:
3687 // = (byte1-'_')*28*28*28*28 + (byte2-'_')*28*28*28 + (byte3-'_')*28*28 + (byte4-'_')*28 + (byte5-'_')
3688 // Hash slots are 28*28*28*28*28 = 17,210,368 each containing one 64bit pointer i.e. 8bytes in length.
3689 // Hash size = 17,210,368*8 = 137,682,944 bytes
3690 // When at end all these slots(17,210,368- Btrees) are traversed the outcome is a sorted wordlist - no need of sorting.
3691
3692 // D:\_KAZE_new-stuff\Leprechaun_quadruplet_r14_minus>Leprechaun_quadruplet.exe GRAFFITH_2048.lst GRAFFITH_2048.wrd z
3693 // Leprechaun(Fast Greedy Word-Ripper), rev. 14_minus_quadruplet, written by Svalqyatchx.
3694 // Leprechaun: 'Oh, well, didn't you hear? Bigger is good, but jumbo is dear.'
3695 // Kaze: Let's see what a 3-way hash + 6,602,752 Binary-Search-Trees can give us,
3696 // also the performance of a 3-way hash + 6,602,752 B-Trees of order 3,
3697 // also the performance of a 1-way hash + 17,210,368 external B-Trees of order 3.
3698 // Size of input file with files for Leprechauning: 42140
3699 // Allocating HASH memory 137,683,009 bytes ... OK
3700 // Allocating/ZEROing 1,047,566 bytes swap file ... OK
3701 // Size of Input TEXTual file: 33,470,581
3702 // |; word count: 3,045,077 of them 0 distinct; Done: 64/64
3703 // ...
3704 // Size of Input TEXTual file: 17,403,406
3705 // /; word count: 2,710,601,882 of them 0 distinct; Done: 64/64
3706 // Bytes per second performance: 17,694,246B/s
3707 // words per second performance: 1,730,907W/s
3708 // Leprechaun: Done.
3709 //
3710 // Leprechaun report:
3711 // Number Of Trees(GREATER THE BETTER): 1,646,004

```

```

3712
3713 // Very bad utilization: 10% - but no sort stage at end - only traversing-and-dumping!
3714
3715 BufStart = pointerflush;
3716 // Slot = ((wrd[0]-'_')*28*28*28*28 + (wrd[1]-'_')*28*28*28 + (wrd[2]-'_')*28*28 + (wrd[3]-'_')*28 + (wrd[4]-'_'))<<3;
3717 Slot = FNV1A_Hash_Jesteress_27bit(wrd, wrdlen)<<3;
3718 memcpy( &PseudoLinkedPointer_64, BufStart+Slot, 8 );
3719
3720 // ##### B-tree order 3 fragment 64bit [
3721 //
3722 // LEAF structure: [LeftPointer][MiddlePointer][RightPointer][Leftword][Rightword]
3723 //                  4bytes      4bytes      4bytes      wrdlen      wrdlen
3724 //                  *              *              *              *              *      <- if *(char *)==0 means the word cell is empty
3725 // ALL B-tree order 3 fragment consists of 3 sub-fragments:
3726 // 1] Search 2] if Search failed Trasirascht(pushing in stack PseudoLinkedPointer(visited LEAFs)) Search 3] Insert Iterative
3727
3728 // LEAF_64 structure: [LeftPointer][MiddlePointer][RightPointer][Leftword] [Rightword]
3729 //                   8bytes      8bytes      8bytes      LongestLineInclusive+1 LongestLineInclusive+1
3730 //                   *              *              *              *              *      <- if *(char *)==0 means the word cell is empty
3731 // Note: In order to use one fread(and strcmp) a NULL postfix for Leftword, Rightword i.e. Leftword_Length=len(Leftword)+1 a kinda stupid
3732 // choice ...
3733 // Note: BufEnd_64 in fact is the first free position after the BUFFER END!
3734
3735 // 1] Search [
3736 //           if (PseudoLinkedPointer_64 == 0) // means EMPTY-SLOT
3737 //           {
3738 //               //if( (unsigned long)(bufend[LetterOffset] - BufStart) + 2*wrdlen + 4 + 4 + 4 < GRMBLFoolAgain[(int)wrdlen] ) // +4 more for BST
3739 //               instead of LL; + more(see LEAF)
3740 //               {
3741 //                   memcpy( BufStart+Slot, &bufend[LetterOffset], 4 );
3742 //                   bufend[LetterOffset] = bufend[LetterOffset] + 4 + 4 + 4; // + 4 due to above commenting
3743 //                   memcpy( bufend[LetterOffset], wrd, wrdlen ); WORDcountDistinct++; bufNumberOfWords[LetterOffset]++;
3744 //                   bufend[LetterOffset] = bufend[LetterOffset] + 2*wrdlen;
3745 //                   if (MAXusedBuffer[wrdlen] < (unsigned long)(bufend[LetterOffset] - BufStart)) {MAXusedBuffer[wrdlen] = (unsigned
3746 //                   long)(bufend[LetterOffset] - BufStart);}
3747 //               }
3748 //               if( 8 + 8 + 8 + 2*(LongestLineInclusive+1) < size_in64_L14 - BufEnd_64 ) // the longest wrdlen is LongestLineInclusive but actual
3749 //               is LongestLineInclusive(+ CR char)
3750 //               {
3751 //                   memcpy( BufStart+Slot, &BufEnd_64, 8 );
3752 //                   BufEnd_64 = BufEnd_64 + 8 + 8 + 8;
3753 //                   fsetpos(fp_outRG, &BufEnd_64);
3754 //                   fwrite(wrd, wrdlen, 1, fp_outRG); WORDcountDistinct++;
3755 //                   fwrite(&OneChar_ieByte, 1, 1, fp_outRG); // Write ZERO ASCII code
3756 //                   BufEnd_64 = BufEnd_64 + 2*(LongestLineInclusive+1);
3757 //                   fsetpos(fp_outRG, &BufEnd_64);
3758 //               }
3759 //           }
3760 //           else
3761 //           { printf( "\nLeprechaun: Failure! Increment 'Memory for each Letter' parameter(third one)!\n" );
3762 //             fprintf( fp_outLOG, "word count: %s of them %s distinct\n", _ui64toaKAZEcomma(WORDcount, 11ToADigits, 10), _ui64toaKAZEcomma((unsigned long
3763 //             long)WORDcountDistinct, 11ToADigits2, 10) );
3764 //             fprintf( fp_outLOG, "Allocated memory: %s bytes\n", _ui64toaKAZEcomma(size_in64_L14, 11ToADigits, 10) );
3765 //             fprintf( fp_outLOG, "Total Attempts to Find/Put WORDS into B-trees order 3: %s\n", _ui64toaKAZEcomma(WORDcountAttemptsToPut, 11ToADigits, 10)
3766 //             );
3767 //             fprintf( fp_outLOG, "Used value for third parameter in KB: %lu\n", (unsigned long)Thunderwith );
3768 //             fprintf( fp_outLOG, "Leprechaun: Failure! Increment 'Memory for each Letter' parameter(third one)!\n\n");
3769 //             return( 1 );
3770 //           }
3771 //           FoundInLinkedList = 1;
3772 //       }
3773 //       else // means USED-SLOT
3774 //       { FoundInLinkedList = 0;
3775 //         StackPtr = 0;
3776 //         while (PseudoLinkedPointer != 0 && FoundInLinkedList == 0)
3777 //             while (PseudoLinkedPointer_64 != 0 && FoundInLinkedList == 0)
3778 //             {
3779 //                 // ***** 'P W P' section [
3780 //                 // LW: existence check if ( *(char *) (PseudoLinkedPointer+4+4+4) != 0 )
3781 //                 // RW: existence check if ( *(char *) (PseudoLinkedPointer+4+4+4+wrdlen) != 0 )
3782 //                 // here ALWAYS LW exists: no need for existence check - line below
3783 //                 // if ( *(char *) (PseudoLinkedPointer+4+4+4) != 0 )
3784 //                 // if (memcmp(PseudoLinkedPointer+4+4+4, wrd, wrdlen) > 0) // go LP
3785 //                 PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8;
3786 //                 fsetpos(fp_outRG, &PseudoLinkedPointerAUX_64);
3787 //                 fread(&FourGramL[0], (LongestLineInclusive+1), 1, fp_outRG);
3788 //                 if (strcmpKAZE13(FourGramL, wrd) > 0) // go LP
3789 //                 { memcpy( &PseudoLinkedPointerNEW, PseudoLinkedPointer + 0, 4 ); //LP
3790 //                   PseudoLinkedPointer = PseudoLinkedPointerNEW;
3791 //                 }
3792 //                 {
3793 //                     PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 0; //LP
3794 //                     if (StackPtr > 8192*3-1-1) { printf( "\nLeprechaun: Failure! 'B-tree order 3' simulated stack overflow!\n" ); return( 13 );}
3795 //                     BSTstack[StackPtr] = PseudoLinkedPointer_64; ++StackPtr; //pt to visited leaf
3796 //                     BSTstack[StackPtr] = 0; ++StackPtr; //LPoffset=0;MPoffset=8;RPoffset=16;
3797 //                     fsetpos(fp_outRG, &PseudoLinkedPointerAUX_64);
3798 //                     fread(&PseudoLinkedPointer_64, 8, 1, fp_outRG);
3799 //                 }
3800 //             }
3801 //             else if (memcmp(PseudoLinkedPointer+4+4+4, wrd, wrdlen) < 0) // go RP or MP

```

```

3794         else if (strcmpKAZE13(FourGramL, wrd) < 0) // go RP or MP
3795             { // RW existence check - line below:
3796 //             if ( *(char *) (PseudoLinkedPointer+4+4+4+wordlen) != 0 ) // RW exists
3797                 PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1); //RW
3798                 fsetpos(fp_outRG, &PseudoLinkedPointerAUX_64);
3799                 fread(&SomeByte, 1, 1, fp_outRG);
3800                 if (SomeByte != 0 ) // RW exists
3801                     { // Here all 'P W P' section is repeated; the way of handling case when dynamic number of words in leaf
3802 // ***** 'P W P' section 2 [
3803 // ***** 'P W P' section 2 [
3804 // LW: existence check if ( *(char *) (PseudoLinkedPointer+4+4+4) != 0 )
3805 // RW: existence check if ( *(char *) (PseudoLinkedPointer+4+4+4+wordlen) != 0 )
3806 //             // here ALWAYS RW exists: no need for existence check - line below
3807 //             // if ( *(char *) (PseudoLinkedPointer+4+4+4+wordlen) != 0 )
3808 //             if (memcmp(PseudoLinkedPointer+4+4+4+wordlen, wrd, wordlen) > 0) // go MP
3809                 PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1);
3810                 fsetpos(fp_outRG, &PseudoLinkedPointerAUX_64);
3811                 fread(&FourGramL[0], (LongestLineInclusive+1), 1, fp_outRG);
3812                 if (strcmpKAZE13(FourGramL, wrd) > 0) // go MP
3813                     { memcpy( &PseudoLinkedPointerNEW, PseudoLinkedPointer + 4, 4 ); //MP
3814                     PseudoLinkedPointer = PseudoLinkedPointerNEW;
3815                     }
3816                 {
3817                 PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8; //MP
3818                 if (StackPtr > 8192*3-1-1) { printf( "\nLeprechaun: Failure! 'B-tree order 3' simulated stack overflow!\n" ); return( 13 );}
3819                 BSTstack[StackPtr] = PseudoLinkedPointer_64; ++StackPtr; //pt to visited leaf
3820                 BSTstack[StackPtr] = 8; ++StackPtr; //LPoffset=0;MPoffset=8;RPoffset=16;
3821                 fsetpos(fp_outRG, &PseudoLinkedPointerAUX_64);
3822                 fread(&PseudoLinkedPointer_64, 8, 1, fp_outRG);
3823                 }
3824 //             else if (memcmp(PseudoLinkedPointer+4+4+4+wordlen, wrd, wordlen) < 0) // go RP
3825 //             else if (strcmpKAZE13(FourGramL, wrd) < 0) // go RP
3826 //             { // No ?w after RW - go RP
3827 //             memcpy( &PseudoLinkedPointerNEW, PseudoLinkedPointer + 4 + 4, 4 ); //RP
3828 //             PseudoLinkedPointer = PseudoLinkedPointerNEW;
3829 //             }
3830 //             {
3831                 PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8; //RP
3832                 if (StackPtr > 8192*3-1-1) { printf( "\nLeprechaun: Failure! 'B-tree order 3' simulated stack overflow!\n" ); return( 13 );}
3833                 BSTstack[StackPtr] = PseudoLinkedPointer_64; ++StackPtr; //pt to visited leaf
3834                 BSTstack[StackPtr] = 16; ++StackPtr; //LPoffset=0;MPoffset=8;RPoffset=16;
3835                 fsetpos(fp_outRG, &PseudoLinkedPointerAUX_64);
3836                 fread(&PseudoLinkedPointer_64, 8, 1, fp_outRG);
3837                 }
3838 //             else FoundInLinkedList = 1; // wrd is RW
3839 //             WORDcountAttemptsToPut++;
3840 // ***** 'P W P' section 2 ]
3841 // ***** 'P W P' section 2 ]
3842 // ***** 'P W P' section 2 ]
3843 //             else // RW empty - go MP
3844 //             { memcpy( &PseudoLinkedPointerNEW, PseudoLinkedPointer + 4, 4 ); //MP
3845 //             PseudoLinkedPointer = PseudoLinkedPointerNEW;
3846 //             }
3847 //             {
3848                 PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8; //MP
3849                 if (StackPtr > 8192*3-1-1) { printf( "\nLeprechaun: Failure! 'B-tree order 3' simulated stack overflow!\n" ); return( 13 );}
3850                 BSTstack[StackPtr] = PseudoLinkedPointer_64; ++StackPtr; //pt to visited leaf
3851                 BSTstack[StackPtr] = 8; ++StackPtr; //LPoffset=0;MPoffset=8;RPoffset=16;
3852                 fsetpos(fp_outRG, &PseudoLinkedPointerAUX_64);
3853                 fread(&PseudoLinkedPointer_64, 8, 1, fp_outRG);
3854                 }
3855 //             }
3856 //             else FoundInLinkedList = 1; // wrd is LW
3857 //             WORDcountAttemptsToPut++;
3858 // ***** 'P W P' section ]
3859 // ***** 'P W P' section ]
3860 // ***** 'P W P' section ]
3861 // ***** 'P W P' section ]
3862 // ***** 'P W P' section ]
3863 // ***** 'P W P' section ]
3864 // ***** 'P W P' section ]
3865 // ***** 'P W P' section ]
3866 // ***** 'P W P' section ]
3867 // ***** 'P W P' section ]
3868 // ***** 'P W P' section ]
3869 // ***** 'P W P' section ]
3870 // ***** 'P W P' section ]
3871 // ***** 'P W P' section ]
3872 // ***** 'P W P' section ]
3873 // ***** 'P W P' section ]
3874 // ***** 'P W P' section ]
3875 // ***** 'P W P' section ]
3876 // ***** 'P W P' section ]
3877 // ***** 'P W P' section ]
3878 // ***** 'P W P' section ]
3879 // ***** 'P W P' section ]
3880 // ***** 'P W P' section ]

```

```

3881 // if ( *(char *) (PseudoLinkedPointer+4+4+4) != 0 )
3882 if ( memcmp(PseudoLinkedPointer+4+4+4, wrd, wrdlen) > 0 ) // go LP
3883 { memcpy( &PseudoLinkedPointerNEW, PseudoLinkedPointer + 0, 4 ); //LP
3884 if ( StackPtr > 8192*3-1-1 ) { printf( "\nLeprechaun: Failure! 'B-tree order 3' simulated stack overflow!\n" ); return( 13 );}
3885 BSTstack[StackPtr] = PseudoLinkedPointer; ++StackPtr; //pt to visited leaf
3886 BSTstack[StackPtr] = 0; ++StackPtr; //LPoffset=0;MPoffset=4;RPoffset=8;
3887 PseudoLinkedPointer = PseudoLinkedPointerNEW;
3888 }
3889 else if ( memcmp(PseudoLinkedPointer+4+4+4, wrd, wrdlen) < 0 ) // go RP or MP
3890 { // RW existence check - line below:
3891 if ( *(char *) (PseudoLinkedPointer+4+4+4+wrdlen) != 0 ) // RW exists
3892 { // Here all 'P W P' section is repeated; the way of handling case when dynamic number of words in leaf
3893 // ++++++
3894 // ***** 'P W P' section 2 [
3895 // LW: existence check if ( *(char *) (PseudoLinkedPointer+4+4+4) != 0 )
3896 // RW: existence check if ( *(char *) (PseudoLinkedPointer+4+4+4+wrdlen) != 0 )
3897 // here ALWAYS RW exists: no need for existence check - line below
3898 // if ( *(char *) (PseudoLinkedPointer+4+4+4+wrdlen) != 0 )
3899 if ( memcmp(PseudoLinkedPointer+4+4+4+wrdlen, wrd, wrdlen) > 0 ) // go MP
3900 { memcpy( &PseudoLinkedPointerNEW, PseudoLinkedPointer + 4, 4 ); //MP
3901 if ( StackPtr > 8192*3-1-1 ) { printf( "\nLeprechaun: Failure! 'B-tree order 3' simulated stack overflow!\n" ); return( 13 );}
3902 BSTstack[StackPtr] = PseudoLinkedPointer; ++StackPtr; //pt to visited leaf
3903 BSTstack[StackPtr] = 4; ++StackPtr; //LPoffset=0;MPoffset=4;RPoffset=8;
3904 PseudoLinkedPointer = PseudoLinkedPointerNEW;
3905 }
3906 else if ( memcmp(PseudoLinkedPointer+4+4+4+wrdlen, wrd, wrdlen) < 0 ) // go RP
3907 { // No ?W after RW - go RP
3908 memcpy( &PseudoLinkedPointerNEW, PseudoLinkedPointer + 4 + 4, 4 ); //RP
3909 if ( StackPtr > 8192*3-1-1 ) { printf( "\nLeprechaun: Failure! 'B-tree order 3' simulated stack overflow!\n" ); return( 13 );}
3910 BSTstack[StackPtr] = PseudoLinkedPointer; ++StackPtr; //pt to visited leaf
3911 BSTstack[StackPtr] = 8; ++StackPtr; //LPoffset=0;MPoffset=4;RPoffset=8;
3912 PseudoLinkedPointer = PseudoLinkedPointerNEW;
3913 }
3914 else FoundInLinkedList = 1; // wrd is RW
3915 // ***** 'P W P' section 2 ]
3916 // ++++++
3917 }
3918 else // RW empty - go MP
3919 { memcpy( &PseudoLinkedPointerNEW, PseudoLinkedPointer + 4, 4 ); //MP
3920 if ( StackPtr > 8192*3-1-1 ) { printf( "\nLeprechaun: Failure! 'B-tree order 3' simulated stack overflow!\n" ); return( 13 );}
3921 BSTstack[StackPtr] = PseudoLinkedPointer; ++StackPtr; //pt to visited leaf
3922 BSTstack[StackPtr] = 4; ++StackPtr; //LPoffset=0;MPoffset=4;RPoffset=8;
3923 PseudoLinkedPointer = PseudoLinkedPointerNEW;
3924 }
3925 }
3926 else FoundInLinkedList = 1; // wrd is LW
3927 // ***** 'P W P' section ]
3928 } // while
3929 // 2] if Search failed Trasirascht(pushing in stack PseudoLinkedPointer(visited LEAFs)) Search ]
3930 // ===== [ The whole section/sub-fragment 2 is commented due to great time differences for Internal_vs_External memory
3931 // accesses - it is far more cheap to have the STACK overhead (moved to sub-fragment 1) ] ===== ]
3932 */
3933 // 3] Insert Iterative [
3934 // There are total 4 situations:
3935 // Case #1: Outer NODE(including ROOT) [ ][ ][ ][LP][ ]
3936 // Case #2: Outer NODE(including ROOT) [ ][ ][ ][LP][RW] Split occurs ---- 'wrduP' (wrdlen bytes)
3937 // Case #3: ROOT [LP][MP][ ][LP][ ] &
3938 // Case #4: Inner NODE(including ROOT) [LP][MP][RP][LP][RW] Split occurs --- | 'PseudoLinkedPointerNEW' (ptr to NEW LEAF)
3939 // | ARE GOING UP
3940 // There are total 2 situations for PARENT LEAF: <-----
3941 // Case #3: [LP][MP][ ][LP][ ]
3942 // Case #4: [LP][MP][RP][LP][RW] Split occurs
3943
3944 // ~ First deal alongly with the OUTER NODE(LEAF) where Search stopped i.e Case #1 & Case #2:
3945 PoffsetInLEAF = BSTstack[--StackPtr];
3946 PseudoLinkedPointer_64 = BSTstack[--StackPtr];
3947 // NOTE: ONE LEAF IS FULL ONLY WHEN LAST CELL FOR KEY(here RW) EXISTS!
3948 // RW: existence check if ( *(char *) (PseudoLinkedPointer+4+4+4+wrdlen) != 0 )
3949 //if ( *(char *) (PseudoLinkedPointer+4+4+4+wrdlen) != 0 ) // If LEAF is full: Case #2
3950 PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1); //RW
3951 fsetpos(fp_outRG, &PseudoLinkedPointerAUX_64);
3952 fread(&SomeByte, 1, 1, fp_outRG);
3953 if (SomeByte != 0) // RW exists
3954 { SplitOccured = 1; WORDcountDistinct++;
3955 // ALlocate NEW LEAF:
3956 // if( (unsigned long)(bufend[LetterOffset] - BufStart) + 2*wrdlen + 4 + 4 + 4 < GRMBLFOolAgain[(int)wrdlen] ) // +4 more for BST
3957 // instead of LL; + more(see LEAF)
3958 {
3959 memcpy( &PseudoLinkedPointerNEW, &bufend[LetterOffset], 4 );
3960 bufend[LetterOffset] = bufend[LetterOffset] + 4 + 4 + 4; // + 4 due to above commenting
3961 bufend[LetterOffset] = bufend[LetterOffset] + 2*wrdlen;
3962 if ( MAXusedBuffer[wrdlen] < (unsigned long)(bufend[LetterOffset] - BufStart) ) { MAXusedBuffer[wrdlen] = (unsigned
3963 long)(bufend[LetterOffset] - BufStart);}
3964 }
3965 if( 8 + 8 + 8 + 2*(LongestLineInclusive+1) < size_in64_L14 - BufEnd_64 ) // the longest wrdlen is LongestLineInclusive but actual
3966 is LongestLineInclusive(+ CR char)

```

```

3965     PseudoLinkedPointerNEW_64 = BufEnd_64;
3966     BufEnd_64 = BufEnd_64 + 8 + 8 + 8;
3967     BufEnd_64 = BufEnd_64 + 2*(LongestLineInclusive+1);
3968 }
3969 else
3970 { printf( "\nLeprechaun: Failure! Increment 'Memory for each Letter' parameter(third one)!\n" );
3971 fprintf( fp_outLOG, "word count: %s of them %s distinct\n", _ui64toaKAZEcomma(WORDCOUNT, l1ToADigits, 10), _ui64toaKAZEcomma((unsigned
long long)WORDCOUNTDISTINCT, l1ToADigits2, 10) );
3972 fprintf( fp_outLOG, "Allocated memory: %s bytes\n", _ui64toaKAZEcomma(size_in64_L14, l1ToADigits, 10) );
3973 fprintf( fp_outLOG, "Total Attempts to Find/Put WORDS into B-trees order 3: %s\n", _ui64toaKAZEcomma(WORDCOUNTATTEMPTSTOPUT,
l1ToADigits, 10) );
3974 fprintf( fp_outLOG, "Used value for third parameter in KB: %lu\n", (unsigned long)Thunderwith );
3975 fprintf( fp_outLOG, "Leprechaun: Failure! Increment 'Memory for each Letter' parameter(third one)!\n\n");
3976     return( 1 );
3977 }
3978 if (POffsetInLEAF == 0) // wrd < LW
3979 {
3980 //     memcpy( wrdUP, PseudoLinkedPointer+4+4+4, wrdlen ); // LW up
3981 //     memcpy( PseudoLinkedPointer+4+4+4, wrd, wrdlen ); // wrd go to OLD LEAF
3982 //     memcpy( PseudoLinkedPointerNEW+4+4+4, PseudoLinkedPointer+4+4+4+wrdlen, wrdlen ); // RW go to NEW LEAF
3983 //     *(char *) (PseudoLinkedPointer+4+4+4+wrdlen) = 0; // RW mark unused in OLD LEAF
3984     PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8;
3985     fsetpos(fp_outRG, &PseudoLinkedPointerAUX_64);
3986     fread(&wrdUP[0], (LongestLineInclusive+1), 1, fp_outRG);
3987     fsetpos(fp_outRG, &PseudoLinkedPointerAUX_64);
3988     fwrite(&wrd[0], (LongestLineInclusive+1), 1, fp_outRG);
3989     PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1);
3990     fsetpos(fp_outRG, &PseudoLinkedPointerAUX_64);
3991     fread(&wrdAUX[0], (LongestLineInclusive+1), 1, fp_outRG);
3992     PseudoLinkedPointerAUX_64 = PseudoLinkedPointerNEW_64 + 8 + 8 + 8;
3993     fsetpos(fp_outRG, &PseudoLinkedPointerAUX_64);
3994     fwrite(&wrdAUX[0], (LongestLineInclusive+1), 1, fp_outRG);
3995     PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1);
3996     fsetpos(fp_outRG, &PseudoLinkedPointerAUX_64);
3997     fwrite(&OneChar_ieByte, 1, 1, fp_outRG); // write ZERO ASCII code
3998 }
3999 if (POffsetInLEAF == 8) // LW < wrd < RW
4000 {
4001 //     memcpy( wrdUP, wrd, wrdlen ); // wrd up
4002 //     memcpy( PseudoLinkedPointerNEW+4+4+4, PseudoLinkedPointer+4+4+4+wrdlen, wrdlen ); // RW go to NEW LEAF
4003 //     *(char *) (PseudoLinkedPointer+4+4+4+wrdlen) = 0; // RW mark unused in OLD LEAF
4004 //     memcpy( wrdUP, wrd, (LongestLineInclusive+1) ); // wrd up
4005     PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1);
4006     fsetpos(fp_outRG, &PseudoLinkedPointerAUX_64);
4007     fread(&wrdAUX[0], (LongestLineInclusive+1), 1, fp_outRG);
4008     PseudoLinkedPointerAUX_64 = PseudoLinkedPointerNEW_64 + 8 + 8 + 8;
4009     fsetpos(fp_outRG, &PseudoLinkedPointerAUX_64);
4010     fwrite(&wrdAUX[0], (LongestLineInclusive+1), 1, fp_outRG);
4011     PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1);
4012     fsetpos(fp_outRG, &PseudoLinkedPointerAUX_64);
4013     fwrite(&OneChar_ieByte, 1, 1, fp_outRG); // write ZERO ASCII code
4014 }
4015 if (POffsetInLEAF == 16) // wrd > RW
4016 {
4017 //     memcpy( wrdUP, PseudoLinkedPointer+4+4+4+wrdlen, wrdlen ); // RW up
4018 //     *(char *) (PseudoLinkedPointer+4+4+4+wrdlen) = 0; // RW mark unused in OLD LEAF
4019 //     memcpy( PseudoLinkedPointerNEW+4+4+4, wrd, wrdlen ); // wrd go to NEW LEAF
4020     PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1);
4021     fsetpos(fp_outRG, &PseudoLinkedPointerAUX_64);
4022     fread(&wrdUP[0], (LongestLineInclusive+1), 1, fp_outRG);
4023     PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1);
4024     fsetpos(fp_outRG, &PseudoLinkedPointerAUX_64);
4025     fwrite(&OneChar_ieByte, 1, 1, fp_outRG); // write ZERO ASCII code
4026     PseudoLinkedPointerAUX_64 = PseudoLinkedPointerNEW_64 + 8 + 8 + 8;
4027     fsetpos(fp_outRG, &PseudoLinkedPointerAUX_64);
4028     fwrite(&wrd[0], (LongestLineInclusive+1), 1, fp_outRG);
4029 }
4030 }
4031 else // If LEAF is not full: Case #1
4032 { SplitOccured = 0; WORDCOUNTDISTINCT++;
4033   if (POffsetInLEAF == 0) // wrd < [LW][] so [LW][] -> [][LW] -> [wrd][LW]
4034   {
4035 //     memcpy( PseudoLinkedPointer+4+4+4+wrdlen, PseudoLinkedPointer+4+4+4, wrdlen );
4036 //     memcpy( PseudoLinkedPointer+4+4+4, wrd, wrdlen );
4037     PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8;
4038     fsetpos(fp_outRG, &PseudoLinkedPointerAUX_64);
4039     fread(&wrdAUX[0], (LongestLineInclusive+1), 1, fp_outRG);
4040     PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1);
4041     fsetpos(fp_outRG, &PseudoLinkedPointerAUX_64);
4042     fwrite(&wrdAUX[0], (LongestLineInclusive+1), 1, fp_outRG);
4043     PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8;
4044     fsetpos(fp_outRG, &PseudoLinkedPointerAUX_64);
4045     fwrite(&wrd[0], (LongestLineInclusive+1), 1, fp_outRG);
4046   }
4047   if (POffsetInLEAF == 8) // wrd > [LW][] so [LW][] -> [LW][wrd]
4048   {
4049 //     memcpy( PseudoLinkedPointer+4+4+4+wrdlen, wrd, wrdlen );
4050     PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1);

```



```

4051         fsetpos(fp_outRG, &PseudoLinkedPointerAUX_64);
4052         fwrite(&wrd[0], (LongestLineInclusive+1), 1, fp_outRG);
4053     }
4054 }
4055
4056 if (SplitOccured != 0)
4057 {
4058     // ~ Second deal with the INNER NODE(S) i.e Case #3 & Case #4:
4059     while (StackPtr != 0 || SplitOccured != 0)
4060     {
4061         // 'PseudoLinkedPointerNEW' is new LEAF to be inserted
4062         // 'wrdUP' is NEW word to be inserted
4063         if (StackPtr != 0)
4064         {
4065             POffsetInLEAF = BSTstack[--StackPtr];
4066             PseudoLinkedPointer_64 = BSTstack[--StackPtr];
4067             //if ( *(char *) (PseudoLinkedPointer+4+4+4+wrdlen) != 0 ) // If LEAF is full: Case #4
4068             PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1); //RW
4069             fsetpos(fp_outRG, &PseudoLinkedPointerAUX_64);
4070             fread(&SomeByte, 1, 1, fp_outRG);
4071             if (SomeByte != 0) // RW exists
4072             { SplitOccured = 1;
4073             // memcp( wrdUPold, wrdUP, wrdlen ); // LW up
4074             // PseudoLinkedPointerNEWold = PseudoLinkedPointerNEW;
4075             memcp( wrdUPold, wrdUP, (LongestLineInclusive+1) );
4076             PseudoLinkedPointerNEWold_64 = PseudoLinkedPointerNEW_64;
4077             // ALlocate NEW LEAF:
4078             if( (unsigned long)(bufend[LetterOffset] - BufStart) + 2*wrdlen + 4 + 4 + 4 < GRMBLFoolAgain[(int)wrdlen] ) // +4 more for BST
4079             instead of LL; + more(see LEAF)
4080             {
4081                 memcp( &PseudoLinkedPointerNEW, &bufend[LetterOffset], 4 );
4082                 bufend[LetterOffset] = bufend[LetterOffset] + 4 + 4 + 4; // + 4 due to above commenting
4083                 bufend[LetterOffset] = bufend[LetterOffset] + 2*wrdlen;
4084                 if (MAXusedBuffer[wrdlen] < (unsigned long)(bufend[LetterOffset] - BufStart)) {MAXusedBuffer[wrdlen] = (unsigned
4085                 long)(bufend[LetterOffset] - BufStart);}
4086             }
4087             if( 8 + 8 + 8 + 2*(LongestLineInclusive+1) < size_in64_L14 - BufEnd_64 ) // the longest wrdlen is LongestLineInclusive but actual
4088             is LongestLineInclusive(+ CR char)
4089             {
4090                 PseudoLinkedPointerNEW_64 = BufEnd_64;
4091                 BufEnd_64 = BufEnd_64 + 8 + 8 + 8;
4092                 BufEnd_64 = BufEnd_64 + 2*(LongestLineInclusive+1);
4093             }
4094             else
4095             { printf( "\nLeprechaun: Failure! Increment 'Memory for each Letter' parameter(third one)!\n" );
4096             fprintf( fp_outLOG, "word count: %s of them %s distinct\n", _ui64toaKAZEcomma(WORdcount, l1ToaDigits, 10), _ui64toaKAZEcomma((unsigned
4097             long long)WORdcountDistinct, l1ToaDigits2, 10) );
4098             fprintf( fp_outLOG, "Allocated memory: %s bytes\n", _ui64toaKAZEcomma(size_in64_L14, l1ToaDigits, 10) );
4099             fprintf( fp_outLOG, "Total Attempts to Find/Put WORDS into B-trees order 3: %s\n", _ui64toaKAZEcomma(WORdcountAttemptsToPut,
4100             l1ToaDigits, 10) );
4101             fprintf( fp_outLOG, "Used value for third parameter in KB: %lu\n", (unsigned long)Thunderwith );
4102             fprintf( fp_outLOG, "Leprechaun: Failure! Increment 'Memory for each Letter' parameter(third one)!\n\n" );
4103             return( 1 );
4104             }
4105             if (POffsetInLEAF == 0) // wrdUPold < LW
4106             {
4107                 memcp( wrdUP, PseudoLinkedPointer+4+4+4, wrdlen ); // LW up
4108                 memcp( PseudoLinkedPointer+4+4+4, wrdUPold, wrdlen ); // wrdUPold go to OLD LEAF
4109                 memcp( PseudoLinkedPointerNEW+4+4+4, PseudoLinkedPointer+4+4+4+wrdlen, wrdlen ); // RW go to NEW LEAF
4110                 *(char *) (PseudoLinkedPointer+4+4+4+wrdlen) = 0; // RW mark unused in OLD LEAF
4111                 // [LP](PseudoLinkedPointerNEWold)[MP][RP](wrdUPold)[LW][RW] -----
4112                 // pair [LW] PseudoLinkedPointerNEW goes up
4113                 // PseudoLinkedPointer: PseudoLinkedPointerNEW: |
4114                 // [LP](PseudoLinkedPointerNEWold)[wrdUPold] [MP][RP][LW] <----
4115                 // no need to put zero in RP because logic is based on words existence:
4116                 memcp( PseudoLinkedPointerNEW+0, PseudoLinkedPointer+4, 4 );
4117                 memcp( PseudoLinkedPointerNEW+4, PseudoLinkedPointer+8, 4 );
4118                 memcp( PseudoLinkedPointer+4, &PseudoLinkedPointerNEWold, 4 );
4119                 PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8;
4120                 fsetpos(fp_outRG, &PseudoLinkedPointerAUX_64);
4121                 fread(&wrdUP[0], (LongestLineInclusive+1), 1, fp_outRG);
4122                 fsetpos(fp_outRG, &PseudoLinkedPointerAUX_64);
4123                 fwrite(&wrdUPold[0], (LongestLineInclusive+1), 1, fp_outRG);
4124                 PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1);
4125                 fsetpos(fp_outRG, &PseudoLinkedPointerAUX_64);
4126                 fread(&wrdAUX[0], (LongestLineInclusive+1), 1, fp_outRG);
4127                 PseudoLinkedPointerAUX_64 = PseudoLinkedPointerNEW_64 + 8 + 8 + 8;
4128                 fsetpos(fp_outRG, &PseudoLinkedPointerAUX_64);
4129                 fwrite(&OneChar_ieByte, 1, 1, fp_outRG); // write ZERO ASCII code
4130                 PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8;
4131                 fsetpos(fp_outRG, &PseudoLinkedPointerAUX_64);
4132                 fread(&PseudoLinkedPointerAUXdumbo_64, 8, 1, fp_outRG);
4133                 PseudoLinkedPointerAUX_64 = PseudoLinkedPointerNEW_64 + 0;
4134                 fsetpos(fp_outRG, &PseudoLinkedPointerAUX_64);
4135                 fwrite(&PseudoLinkedPointerAUXdumbo_64, 8, 1, fp_outRG);

```

```

4134 PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 16;
4135 fsetpos(fp_outRG, &PseudoLinkedPointerAUX_64);
4136 fread(&PseudoLinkedPointerAUXdumbo_64, 8, 1, fp_outRG);
4137 PseudoLinkedPointerAUX_64 = PseudoLinkedPointerNEW_64 + 8;
4138 fsetpos(fp_outRG, &PseudoLinkedPointerAUX_64);
4139 fwrite(&PseudoLinkedPointerAUXdumbo_64, 8, 1, fp_outRG);
4140 PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8;
4141 fsetpos(fp_outRG, &PseudoLinkedPointerAUX_64);
4142 fwrite(&PseudoLinkedPointerNEWold_64, 8, 1, fp_outRG);
4143 }
4144 if (PoffsetInLEAF == 8) // LW < wrdUPold < RW
4145 {
4146 // memcpy( wrdUP, wrdUPold, wrdlen ); // wrdUPold up
4147 // memcpy( PseudoLinkedPointerNEW+4+4+4, PseudoLinkedPointer+4+4+4+wrdlen, wrdlen ); // RW go to NEW LEAF
4148 // *(char *) (PseudoLinkedPointer+4+4+4+wrdlen) = 0; // RW mark unused in OLD LEAF
4149 // // [LP][MP](PseudoLinkedPointerNEWold)[RP][LW](wrdUPold)[RW] -----
4150 // // pair [wrdUPold] PseudoLinkedPointerNEW goes up |
4151 // // PseudoLinkedPointer: PseudoLinkedPointerNEW: |
4152 // // [LP][MP][LW] (PseudoLinkedPointerNEWold)[RP][RW] <----
4153 // // no need to put zero in RP because logic is based on words existence:
4154 // memcpy( PseudoLinkedPointerNEW+0, &PseudoLinkedPointerNEWold, 4 );
4155 // memcpy( PseudoLinkedPointerNEW+4, PseudoLinkedPointer+8, 4 );
4156 // memcpy( wrdUP, wrdUPold, (LongestLineInclusive+1) );
4157 PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1);
4158 fsetpos(fp_outRG, &PseudoLinkedPointerAUX_64);
4159 fread(&wrdAUX[0], (LongestLineInclusive+1), 1, fp_outRG);
4160 PseudoLinkedPointerAUX_64 = PseudoLinkedPointerNEW_64 + 8 + 8 + 8;
4161 fsetpos(fp_outRG, &PseudoLinkedPointerAUX_64);
4162 fwrite(&wrdAUX[0], (LongestLineInclusive+1), 1, fp_outRG);
4163 PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1);
4164 fsetpos(fp_outRG, &PseudoLinkedPointerAUX_64);
4165 fwrite(&OneChar_ieByte, 1, 1, fp_outRG); // Write ZERO ASCII code
4166 PseudoLinkedPointerAUX_64 = PseudoLinkedPointerNEW_64 + 0;
4167 fsetpos(fp_outRG, &PseudoLinkedPointerAUX_64);
4168 fwrite(&PseudoLinkedPointerNEWold_64, 8, 1, fp_outRG);
4169 PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 16;
4170 fsetpos(fp_outRG, &PseudoLinkedPointerAUX_64);
4171 fread(&PseudoLinkedPointerAUXdumbo_64, 8, 1, fp_outRG);
4172 PseudoLinkedPointerAUX_64 = PseudoLinkedPointerNEW_64 + 8;
4173 fsetpos(fp_outRG, &PseudoLinkedPointerAUX_64);
4174 fwrite(&PseudoLinkedPointerAUXdumbo_64, 8, 1, fp_outRG);
4175 }
4176 if (PoffsetInLEAF == 16) // wrdUPold > RW
4177 {
4178 // memcpy( wrdUP, PseudoLinkedPointer+4+4+4+wrdlen, wrdlen ); // RW up
4179 // *(char *) (PseudoLinkedPointer+4+4+4+wrdlen) = 0; // RW mark unused in OLD LEAF
4180 // memcpy( PseudoLinkedPointerNEW+4+4+4, wrdUPold, wrdlen ); // wrdUPold go to NEW LEAF
4181 // // [LP][MP][RP](PseudoLinkedPointerNEWold)[LW][RW](wrdUPold) -----
4182 // // pair [RW] PseudoLinkedPointerNEW goes up |
4183 // // PseudoLinkedPointer: PseudoLinkedPointerNEW: |
4184 // // [LP][MP][LW] [RP](PseudoLinkedPointerNEWold)[wrdUPold] <---
4185 // // no need to put zero in RP because logic is based on words existence:
4186 // memcpy( PseudoLinkedPointerNEW+0, PseudoLinkedPointer+8, 4 );
4187 // memcpy( PseudoLinkedPointerNEW+4, &PseudoLinkedPointerNEWold, 4 );
4188 PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1);
4189 fsetpos(fp_outRG, &PseudoLinkedPointerAUX_64);
4190 fread(&wrdUP[0], (LongestLineInclusive+1), 1, fp_outRG);
4191 PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1);
4192 fsetpos(fp_outRG, &PseudoLinkedPointerAUX_64);
4193 fwrite(&OneChar_ieByte, 1, 1, fp_outRG); // Write ZERO ASCII code
4194 PseudoLinkedPointerAUX_64 = PseudoLinkedPointerNEW_64 + 8 + 8 + 8;
4195 fsetpos(fp_outRG, &PseudoLinkedPointerAUX_64);
4196 fwrite(&wrdUPold[0], (LongestLineInclusive+1), 1, fp_outRG);
4197 PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 16;
4198 fsetpos(fp_outRG, &PseudoLinkedPointerAUX_64);
4199 fread(&PseudoLinkedPointerAUXdumbo_64, 8, 1, fp_outRG);
4200 PseudoLinkedPointerAUX_64 = PseudoLinkedPointerNEW_64 + 0;
4201 fsetpos(fp_outRG, &PseudoLinkedPointerAUX_64);
4202 fwrite(&PseudoLinkedPointerAUXdumbo_64, 8, 1, fp_outRG);
4203 PseudoLinkedPointerAUX_64 = PseudoLinkedPointerNEW_64 + 8;
4204 fsetpos(fp_outRG, &PseudoLinkedPointerAUX_64);
4205 fwrite(&PseudoLinkedPointerNEWold_64, 8, 1, fp_outRG);
4206 }
4207 }
4208 else // If LEAF is not full: Case #3
4209 { splitOccured = 0;
4210 if (PoffsetInLEAF == 0) // wrdUP < [LW][ ] so [LW][ ] -> [ ][LW] -> [wrdUP][LW]
4211 {
4212 // memcpy( PseudoLinkedPointer+4+4+4+wrdlen, PseudoLinkedPointer+4+4+4, wrdlen );
4213 // memcpy( PseudoLinkedPointer+4+4+4, wrdUP, wrdlen );
4214 // // [LP][MP][ ] -> [LP][ ][MP] -> [LP][np][MP]
4215 // memcpy( PseudoLinkedPointer+8, PseudoLinkedPointer+4, 4 );
4216 // memcpy( PseudoLinkedPointer+4, &PseudoLinkedPointerNEW, 4 );
4217 PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8;
4218 fsetpos(fp_outRG, &PseudoLinkedPointerAUX_64);
4219 fread(&wrdAUX[0], (LongestLineInclusive+1), 1, fp_outRG);
4220 PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1);
4221 fsetpos(fp_outRG, &PseudoLinkedPointerAUX_64);

```

```

4222     fwrite(&wrdAUX[0], (LongestLineInclusive+1), 1, fp_outRG);
4223     PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8;
4224     fsetpos(fp_outRG, &PseudoLinkedPointerAUX_64);
4225     fwrite(&wrdUP[0], (LongestLineInclusive+1), 1, fp_outRG);
4226     PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8;
4227     fsetpos(fp_outRG, &PseudoLinkedPointerAUX_64);
4228     fread(&PseudoLinkedPointerAUXdumbo_64, 8, 1, fp_outRG);
4229     PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 16;
4230     fsetpos(fp_outRG, &PseudoLinkedPointerAUX_64);
4231     fwrite(&PseudoLinkedPointerAUXdumbo_64, 8, 1, fp_outRG);
4232     PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8;
4233     fsetpos(fp_outRG, &PseudoLinkedPointerAUX_64);
4234     fwrite(&PseudoLinkedPointerNEW_64, 8, 1, fp_outRG);
4235 }
4236 if (PoffsetInLEAF == 8) // wrdUP > [LW][ ] so [LW][ ] -> [LW][wrdUP]
4237 {
4238     // memcpy( PseudoLinkedPointer+4+4+4+wrdlen, wrdUP, wrdlen );
4239     // // [LP][MP][ ] -> [LP][MP][np]
4240     // memcpy( PseudoLinkedPointer+8, &PseudoLinkedPointerNEW, 4 );
4241     PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1);
4242     fsetpos(fp_outRG, &PseudoLinkedPointerAUX_64);
4243     fwrite(&wrdUP[0], (LongestLineInclusive+1), 1, fp_outRG);
4244     PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 16;
4245     fsetpos(fp_outRG, &PseudoLinkedPointerAUX_64);
4246     fwrite(&PseudoLinkedPointerNEW_64, 8, 1, fp_outRG);
4247 }
4248 break;
4249 }
4250 }
4251 else // Empty stack means ROOT and more over ROOT is already splitted(Case #4 is off)
4252 {
4253     // If LEAF is not full: Case #3
4254     // THIS IS WHERE A NEW(SECOND) LEAF 'PseudoLinkedPointerROOT' must be allocated:
4255     // if( (unsigned long)(bufend[LetterOffset] - BufStart) + 2*wrdlen + 4 + 4 + 4 < GRMBLFoolAgain[(int)wrdlen] ) // +4 more for BST
4256     // instead of LL; + more(see LEAF)
4257     // {
4258     //     memcpy( &PseudoLinkedPointerROOT, &bufend[LetterOffset], 4 );
4259     //     bufend[LetterOffset] = bufend[LetterOffset] + 4 + 4 + 4; // + 4 due to above commenting
4260     //     memcpy( bufend[LetterOffset], wrdUP, wrdlen );
4261     //     bufend[LetterOffset] = bufend[LetterOffset] + 2*wrdlen;
4262     //     if (MAXusedBuffer[wrdlen] < (unsigned long)(bufend[LetterOffset] - BufStart)) {MAXusedBuffer[wrdlen] = (unsigned
4263     //         long)(bufend[LetterOffset] - BufStart);}
4264     //     if( 8 + 8 + 8 + 2*(LongestLineInclusive+1) < size_in64_L14 - BufEnd_64 ) // the longest wrdlen is LongestLineInclusive but actual
4265     //     is LongestLineInclusive(+ CR char)
4266     //     {
4267     //         PseudoLinkedPointerROOT_64 = BufEnd_64;
4268     //         BufEnd_64 = BufEnd_64 + 8 + 8 + 8;
4269     //         BufEnd_64 = BufEnd_64 + 2*(LongestLineInclusive+1);
4270     //         PseudoLinkedPointerAUX_64 = PseudoLinkedPointerROOT_64 + 8 + 8 + 8;
4271     //         fsetpos(fp_outRG, &PseudoLinkedPointerAUX_64);
4272     //         fwrite(&wrdUP[0], (LongestLineInclusive+1), 1, fp_outRG);
4273     //     }
4274     //     else
4275     //     { printf( "\nLeprechaun: Failure! Increment 'Memory for each Letter' parameter(third one)!\n" );
4276     //         fprintf( fp_outLOG, "word count: %s of them %s distinct\n", _ui64toaKAZEcomma(WORDcount, 11ToADigits, 10), _ui64toaKAZEcomma((unsigned
4277     //         long)WORDcountDistinct, 11ToADigits2, 10) );
4278     //         fprintf( fp_outLOG, "Allocated memory: %s bytes\n", _ui64toaKAZEcomma(size_in64_L14, 11ToADigits, 10) );
4279     //         fprintf( fp_outLOG, "Total Attempts to Find/Put WORDS into B-trees order 3: %s\n", _ui64toaKAZEcomma(WORDcountAttemptsToPut,
4280     //         11ToADigits, 10) );
4281     //         fprintf( fp_outLOG, "Used value for third parameter in KB: %lu\n", (unsigned long)Thunderwith );
4282     //         fprintf( fp_outLOG, "Leprechaun: Failure! Increment 'Memory for each Letter' parameter(third one)!\n\n");
4283     //         return( 1 );
4284     //     }
4285     //     // Here -- 'PseudoLinkedPointerROOT' --
4286     //     // | (wrdUP) |
4287     //     // 'PseudoLinkedPointer' <-- --> 'PseudoLinkedPointerNEW'
4288     //     // (LW) (RW)
4289     //     memcpy( PseudoLinkedPointerROOT, &PseudoLinkedPointer, 4 ); // LP
4290     //     memcpy( PseudoLinkedPointerROOT+4, &PseudoLinkedPointerNEW, 4 ); // MP
4291     //     // Here must NEW ROOT be updated i.e. HASH table(SLOT) must point it:
4292     //     memcpy( BufStart+Slot, &PseudoLinkedPointerROOT, 4 );
4293     //     PseudoLinkedPointerAUX_64 = PseudoLinkedPointerROOT_64;
4294     //     fsetpos(fp_outRG, &PseudoLinkedPointerAUX_64);
4295     //     fwrite(&PseudoLinkedPointer_64, 8, 1, fp_outRG);
4296     //     PseudoLinkedPointerAUX_64 = PseudoLinkedPointerROOT_64 + 8;
4297     //     fsetpos(fp_outRG, &PseudoLinkedPointerAUX_64);
4298     //     fwrite(&PseudoLinkedPointerNEW_64, 8, 1, fp_outRG);
4299     //     memcpy( BufStart+Slot, &PseudoLinkedPointerROOT_64, 8 );
4300     //     break; //because it is ROOT without split
4301     // }
4302     // } // while
4303 } //if (SplitOccured != 0)
4304 // 3] Insert Iterative ]
4305 //if (FoundInLinkedList == 0)
4306 // ##### B-tree order 3 64bit ]
4307 //
4308 // External Btrees ]

```

```

4305 } //if (BStorBtree != 2) {
4306     } // if ( ( PLE_words == 4 ) && ( wrdlen <= 31 ) ) {
4307     } // if( 1 <= wrdlen && wrdlen <= 31 )
4308 if ( PLE_words_INITflag == 1 ) { PLE_words = 0; PLE_words_INITflag = 0; } // Quadruple!
4309     wrdlen = 0;
4310     // This fragment is MIRRORed: #1 copy ]
4311     }
4312     //else if( workbyte >= 'A' && workbyte <= 'Z' )
4313     else if( workbyte <= 'Z' )
4314     {
4315         //if( wrdlen < 31 )
4316         if( wrdlen < LongestLineInclusive )
4317         { wrd[ wrdlen ] = workbyte + 32 ; }
4318         wrdlen++;
4319     }
4320     else if( workbyte >= 'a' && workbyte <= 'z' )
4321     {
4322         //if( wrdlen < 31 )
4323         if( wrdlen < LongestLineInclusive )
4324         { wrd[ wrdlen ] = workbyte; }
4325         wrdlen++;
4326     }
4327     else
4328     {
4329         // This fragment is MIRRORed: #2 copy [
4330         goto ElStupido;
4331         // This fragment is MIRRORed: #2 copy ]
4332     }
4333     } // i 'for'
4334     //~~~~~
4335 //++MeInitchka;
4336 //MeInitchka = MeInitchka % 4;
4337 //if (MeInitchka == 0){ printf( "|; word count: %s of them %s distinct; Done: %lu/64\n", _ui64toaKAZEcomma(WORDcount, l1ToaDigits, 10),
4338 //_ui64toaKAZEcomma((unsigned long long)WORDcountDistinct, l1ToaDigits2, 10), 64 ); }
4339 //if (MeInitchka == 1){ printf( "/; word count: %s of them %s distinct; Done: %lu/64\n", _ui64toaKAZEcomma(WORDcount, l1ToaDigits, 10),
4340 //_ui64toaKAZEcomma((unsigned long long)WORDcountDistinct, l1ToaDigits2, 10), 64 ); }
4341 //if (MeInitchka == 2){ printf( "-; word count: %s of them %s distinct; Done: %lu/64\n", _ui64toaKAZEcomma(WORDcount, l1ToaDigits, 10),
4342 //_ui64toaKAZEcomma((unsigned long long)WORDcountDistinct, l1ToaDigits2, 10), 64 ); }
4343 //if (MeInitchka == 3){ printf( "\\; word count: %s of them %s distinct; Done: %lu/64\n", _ui64toaKAZEcomma(WORDcount, l1ToaDigits, 10),
4344 //_ui64toaKAZEcomma((unsigned long long)WORDcountDistinct, l1ToaDigits2, 10), 64 ); }
4345 MeInitchka = MeInitchka & 3; // 0 1 2 3: 00 01 10 11
4346 printf( "%s; word count: %s of them %s distinct; Done: %lu/64\n", Auberge[MeInitchka++], _ui64toaKAZEcomma(WORDcount, l1ToaDigits, 10),
4347 _ui64toaKAZEcomma((unsigned long long)WORDcountDistinct, l1ToaDigits2, 10), 64 );
4348
4349     LINE10len = 0;
4350     LINE10[ LINE10len ] = 0;
4351     fclose( fp_inLINE );
4352     }
4353     } // k 'for'
4354
4355 (void) time(&t3);
4356 if (t3 <= t1) {t3 = t1; t3++;}
4357 printf( "Bytes per second performance: %sB/s\n", _ui64toaKAZEcomma(FilesLEN/((int) t3-t1), l1ToaDigits, 10) ); // Rev. 12+
4358 printf( "words per second performance: %sw/s\n", _ui64toaKAZEcomma(WORDcount/((int) t3-t1), l1ToaDigits, 10) ); // Rev. 12+
4359
4360 if (BStorBtree != 2) {
4361     // FLUSH FLUSH FLUSH FLUSH FLUSH FLUSH FLUSH FLUSH FLUSH FLUSH FLUSH FLUSH
4362     printf("Flushing unsorted words ...\n");
4363     if ( ( fp_out = fopen( argv[2], "wb+" ) ) == NULL )
4364     { printf( "Leprechaun: Can't create file %s \n", argv[2] ); return( 1 ); }
4365     ZEROS[0] = 0; ZEROS[1] = 0; ZEROS[2] = 0; ZEROS[3] = 0;
4366     CRdLFa[0] = 13; CRdLFa[1] = 10;
4367
4368     for( i = 0; i < 806; i++ )
4369     { //BufStart = pointerflush + i * LetterBuffer; // OLD
4370         BufStart = pointerflush + (i / 31) * WHOLEletter_BufferSize + OffsetsInBuffer[i % 31];
4371         // for( j = 0; j < NumberofSLOTS; j++ )
4372         // {
4373         //     slot = j<<2;
4374         //     memcpy( &PseudoLinkedPointer, BufStart+slot, 4 );
4375         //     while (PseudoLinkedPointer != 0)
4376         //     { memcpy( &PseudoLinkedPointerNEW, PseudoLinkedPointer, 4 );
4377         //       memcpy( PseudoLinkedPointer, ZEROS, 4 );
4378         //       PseudoLinkedPointer = PseudoLinkedPointerNEW;
4379         //     }
4380         // }
4381         // Start of COUPLES [OFFSET: 4byte(ZEROS)][WORD:up to 31bytes]
4382         // fwrite(BufStart+(NumberOfSLOTS+1)*4, bufend[i] - (BufStart+(NumberOfSLOTS+1)*4), 1, fp_out );
4383         // /* Follows STATE OF UGLINESS: */
4384         // Flushing = BufStart+(NumberOfSLOTS+1)*4 + 4; // '+ 4' in order to skip first 4 zeros
4385         // //in case of current buffer not have been used then NOT entering in this cycle
4386         // while(Flushing < bufend[i])
4387         // { if (*Flushing != 0) {fwrite(Flushing, 1, 1, fp_out ); TotalWLchars++;}
4388         //   // Below 'Flushing-1' works due to skipped first 4 zeros!
4389         //   if (*(Flushing-1) != 0 && *Flushing == 0) {fwrite(CRdLFa, 2, 1, fp_out);}
4390         //   //last word must be suffixed with 1310 too
4391         //   if (Flushing == bufend[i]-1) {fwrite(CRdLFa, 2, 1, fp_out );}

```

```

4388 // Flushing++;
4389 // }
4390
4391 for( j = 0; j < NumberOfSLOTS; j++ )
4392 {
4393     slot = j<<2;
4394     memcpy( &PseudoLinkedPointer, BufStart+Slot, 4 );
4395     if (PseudoLinkedPointer != 0)
4396     {
4397         NumberOfTrees++;
4398         if (BSTorBtree == 0)
4399         {
4400             // ===== BST traverse [
4401             // DONE JOB:
4402             // Must be written BST traverse ! with simulated stack i.e. non-recursive.
4403             // ...
4404             // /*
4405             // Given a binary search tree, print out
4406             // its data elements in increasing
4407             // sorted order.
4408             // */
4409             // void printTree(struct node* node) {
4410             // if (node == NULL) return;
4411             // printTree(node->left);
4412             // printf("%d ", node->data);
4413             // printTree(node->right);
4414             // }
4415
4416             // FUTURE JOB:
4417             // I need functions:
4418             // BST_LeafNumber() // greater the better
4419             // BST_NodeNumber() // 'BSTcurrent' below
4420             // BST_Peak() // i.e. levels, root has height = 1
4421             // BST_PeakIB() // IBBST(Ideal Balanced BST) has 1 + lgNodeNumber height
4422             // I need 'Ideal Balancing BST FRAGMENT' with simulated stack:
4423             // I need 'Ideal Balancing BST FRAGMENT' to be executed when Peak() >= PeakIB()<<1:
4424
4425             // ----- [
4426             BSTcurrentNode = 0; BSTcurrentPeak = 0; BSTcurrentLeaf = 0;
4427             BSTcurrentPeakMAX = 0; // Height of current BST
4428             StackPtr = 0;
4429             while ( 2==2 ) {
4430                 while (PseudoLinkedPointer != 0)
4431                 {
4432                     if (StackPtr > 8192*3-1-3) { printf( "\nLeprechaun: Failure! BST simulated stack overflow, too high BST!\n" ); return( 13 );}
4433                     memcpy( &PseudoLinkedPointerNEWleft, PseudoLinkedPointer, 4 );
4434                     PseudoLinkedPointer = PseudoLinkedPointer + 4;
4435                     memcpy( &PseudoLinkedPointerNEWright, PseudoLinkedPointer, 4 );
4436                     BSTstack[StackPtr] = PseudoLinkedPointer + 4; ++StackPtr; //ptr to wrd
4437                     BSTstack[StackPtr] = PseudoLinkedPointerNEWright; ++StackPtr;
4438                     BSTstack[StackPtr] = PseudoLinkedPointerNEWleft; ++StackPtr; //needed for stats not for recursion
4439                     // BST stats [
4440                     if (PseudoLinkedPointerNEWleft == 0 && PseudoLinkedPointerNEWright == 0) {BSTcurrentLeaf++; BSTsTotalLEAFs++;}
4441                     BSTcurrentPeak++;
4442                     if (BSTcurrentPeakMAX < BSTcurrentPeak) BSTcurrentPeakMAX = BSTcurrentPeak;
4443                     BSTstack[StackPtr] = BSTcurrentPeak; ++StackPtr; //needed for stats not for recursion
4444                     // BST stats ]
4445                     PseudoLinkedPointer = PseudoLinkedPointerNEWright; // choose right instead of 'PseudoLinkedPointerNEWleft' because of stats
4446                 }
4447                 if (StackPtr == 0) break;
4448                 BSTcurrentPeak = BSTstack[--StackPtr]; // level of the node(1 is root) needed only for stats(print)
4449                 PseudoLinkedPointerNEWleft = BSTstack[--StackPtr]; // left pointer needed only for stats(print)
4450                 PseudoLinkedPointerNEWright = BSTstack[--StackPtr]; // right pointer
4451                 memcpy( wrd, BSTstack[--StackPtr], i%31+1 );
4452                 fwrite(wrd, i%31+1, 1, fp_out); TotalWLchars = TotalWLchars + i%31+1;
4453                 fwrite(CrDLfa, 2, 1, fp_out);
4454                 BSTcurrentNode++;
4455                 PseudoLinkedPointer = PseudoLinkedPointerNEWleft;
4456             }
4457             // ----- ]
4458             // BST stats [
4459             if (BSTwithMAXnode < BSTcurrentNode) {
4460                 BSTwithMAXnode = BSTcurrentNode;
4461                 BSTwithMAXnodePEAK = BSTcurrentPeakMAX;
4462                 BSTwithMAXnodeLEAF = BSTcurrentLeaf;
4463                 BSTcurrentNodeMAXQUANTITY = 0;
4464             }
4465             if (BSTwithMAXnode == BSTcurrentNode) BSTcurrentNodeMAXQUANTITY++;
4466             if (BSTwithMAXpeak < BSTcurrentPeakMAX) {
4467                 BSTwithMAXpeak = BSTcurrentPeakMAX;
4468                 BSTwithMAXpeakNODE = BSTcurrentNode;
4469                 BSTwithMAXpeakLEAF = BSTcurrentLeaf;
4470                 BSTcurrentPeakMAXQUANTITY = 0; iBSTwithMAXpeak=i; jBSTwithMAXpeak=j;
4471             }
4472             if (BSTwithMAXpeak == BSTcurrentPeakMAX) BSTcurrentPeakMAXQUANTITY++;
4473             if (BSTwithMAXleaf < BSTcurrentLeaf) {
4474                 BSTwithMAXleaf = BSTcurrentLeaf;

```

```

4475     BSTwithMAXleafNODE = BSTcurrentNode;
4476     BSTwithMAXleafPEAK = BSTcurrentPeakMAX;
4477     BSTcurrentLeafMAXQUANTITY = 0;
4478 }
4479 if (BSTwithMAXleaf == BSTcurrentLeaf) BSTcurrentLeafMAXQUANTITY++;
4480 // BST stats ]
4481 // ===== BST traverse ]
4482 } else
4483 {
4484 // $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$ B-tree order 3 traverse [
4485 // DONE JOB:
4486 // Must be written B-tree traverse ! with simulated stack i.e. non-recursive.
4487 // ...
4488 StackPtr = 0;
4489 while ( 2==2 ) {
4490     while (PseudoLinkedPointer != 0)
4491     {
4492         if (StackPtr > 8192*3-1) { printf( "\nLeprechaun: Failure! B-tree simulated stack overflow, too high B-tree!\n" ); return( 13 );}
4493         BSTstack[StackPtr] = PseudoLinkedPointer; ++StackPtr; //ptr to Rwrdr
4494         if ( *(char *) (PseudoLinkedPointer + 4 + 4 + 4 + i%31+1) == 0 ) {memcpy( PseudoLinkedPointer + 4 + 4, &BufStart[NumberOfSLOTS*4], 4 );}
4495         memcpy( &PseudoLinkedPointerNEWleft, PseudoLinkedPointer, 4 );
4496         memcpy( &PseudoLinkedPointerNEWMiddle, PseudoLinkedPointer + 4, 4 );
4497         memcpy( &PseudoLinkedPointerNEWRright, PseudoLinkedPointer + 4 + 4, 4 );
4498 // Give first from right to left non-zero PTR
4499         if (PseudoLinkedPointerNEWRright !=0 )
4500         { memcpy( PseudoLinkedPointer + 4 + 4, &BufStart[NumberOfSLOTS*4], 4 );
4501           PseudoLinkedPointer = PseudoLinkedPointerNEWRright;
4502         }
4503         else if (PseudoLinkedPointerNEWMiddle !=0 )
4504         { memcpy( PseudoLinkedPointer + 4, &BufStart[NumberOfSLOTS*4], 4 );
4505           PseudoLinkedPointer = PseudoLinkedPointerNEWMiddle;
4506         }
4507         else if (PseudoLinkedPointerNEWleft !=0 )
4508         { memcpy( PseudoLinkedPointer, &BufStart[NumberOfSLOTS*4], 4 );
4509           PseudoLinkedPointer = PseudoLinkedPointerNEWleft;
4510         }
4511         else
4512         {
4513             PseudoLinkedPointer = 0;
4514         }
4515     }
4516     if (StackPtr == 0) break;
4517     PseudoLinkedPointer = BSTstack[--StackPtr];
4518     memcpy( &PseudoLinkedPointerNEWleft, PseudoLinkedPointer, 4 );
4519     memcpy( &PseudoLinkedPointerNEWMiddle, PseudoLinkedPointer + 4, 4 );
4520     memcpy( &PseudoLinkedPointerNEWRright, PseudoLinkedPointer + 4 + 4, 4 );
4521     if (PseudoLinkedPointerNEWleft+PseudoLinkedPointerNEWMiddle+PseudoLinkedPointerNEWRright == 0) // One LEAF is PRINTED when LP=0 MP=0
4522     {
4523         RP=0
4524         memcpy( wrd, PseudoLinkedPointer + 4 + 4 + 4, i%31+1 );
4525         fwrite(wrd, i%31+1, 1, fp_out); TotalWLchars = TotalWLchars + i%31+1;
4526         fwrite(CRDLFa, 2, 1, fp_out);
4527         if ( *(char *) (PseudoLinkedPointer + 4 + 4 + 4 + i%31+1) != 0 )
4528         { memcpy( wrd, PseudoLinkedPointer + 4 + 4 + 4 + i%31+1, i%31+1 );
4529           fwrite(wrd, i%31+1, 1, fp_out); TotalWLchars = TotalWLchars + i%31+1;
4530           fwrite(CRDLFa, 2, 1, fp_out);
4531         }
4532         PseudoLinkedPointer = 0;
4533     }
4534 // $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$ B-tree order 3 traverse ]
4535 }
4536 }
4537 } // j
4538 } // i
4539 } // i
4540 if (BSTorBtree == 0)
4541 {
4542 // ~~~~~ Longest path ~~~~~ [
4543 i=iBSTwithMAXpeak; j=jBSTwithMAXpeak;
4544 BufStart = pointerfFlush + (i / 31) * WHOLEletter_BufferSize + OffsetsInBuffer[i % 31];
4545 Slot = j<<2;
4546 memcpy( &PseudoLinkedPointer, BufStart+Slot, 4 );
4547 if (PseudoLinkedPointer != 0)
4548 {
4549 // ----- [
4550 BSTcurrentNode = 0; BSTcurrentPeak = 0; BSTcurrentLeaf = 0;
4551 BSTcurrentPeakMAX = 0; // Height of current BST
4552 StackPtr = 0;
4553 // BST print [
4554 fprintf( fp_outLOG, "A(not always THE) Binary-Search-Tree with the longest path(height, PEAK, number of levels):\n" );
4555 // BST print ]
4556 while ( 2==2 ) {
4557     while (PseudoLinkedPointer != 0)
4558     {
4559         if (StackPtr > 8192*3-1-3) { printf( "\nLeprechaun: Failure! BST simulated stack overflow, too high BST!\n" ); return( 13 );}

```

```

4561 memcopy( &PseudoLinkedPointerNEWleft, PseudoLinkedPointer, 4 );
4562 PseudoLinkedPointer = PseudoLinkedPointer + 4;
4563 memcopy( &PseudoLinkedPointerNEWright, PseudoLinkedPointer, 4 );
4564 BSTstack[StackPtr] = PseudoLinkedPointer + 4; ++StackPtr; //ptr to wrd
4565 BSTstack[StackPtr] = PseudoLinkedPointerNEWright; ++StackPtr;
4566 BSTstack[StackPtr] = PseudoLinkedPointerNEWleft; ++StackPtr; //needed for stats not for recursion
4567 // BST stats [
4568 if (PseudoLinkedPointerNEWleft == 0 && PseudoLinkedPointerNEWright == 0) {BSTcurrentLeaf++;} //BSTsTotalLEAFs++;} // REMOVED
to avoid mess in TOTAL stats
4569 BSTcurrentPeak++;
4570 if (BSTcurrentPeakMAX < BSTcurrentPeak) BSTcurrentPeakMAX = BSTcurrentPeak;
4571 BSTstack[StackPtr] = BSTcurrentPeak; ++StackPtr; //needed for stats not for recursion
4572 // BST stats ]
4573 PseudoLinkedPointer = PseudoLinkedPointerNEWright; // choose right instead of 'PseudoLinkedPointerNEWleft' because of stats
print
4574 }
4575 if (StackPtr == 0) break;
4576 BSTcurrentPeak = BSTstack[--StackPtr]; // level of the node(1 is root) needed only for stats(print)
4577 PseudoLinkedPointerNEWleft = BSTstack[--StackPtr]; // left pointer needed only for stats(print)
4578 PseudoLinkedPointerNEWright = BSTstack[--StackPtr]; // right pointer
4579 memcopy( wrd, BSTstack[--StackPtr], i%31+1 );
4580 //fwrite(wrd, i%31+1, 1, fp_out); TotalWLchars = TotalWLchars + i%31+1;
4581 //fwrite(CrDLFa, 2, 1, fp_out);
4582 BSTcurrentNode++;
4583 PseudoLinkedPointer = PseudoLinkedPointerNEWleft;
4584 // BST print [
4585 for( k = 0; k < BSTcurrentPeak; k++ ) fprintf( fp_outLOG, "%c", ' ' );
4586 if (PseudoLinkedPointerNEWleft == 0) fprintf( fp_outLOG, "[" ); else fprintf( fp_outLOG, "]" );
4587 for( k = 0; k < i%31+1; k++ ) fprintf( fp_outLOG, "%c", *(char *) (wrd+k) );
4588 if (PseudoLinkedPointerNEWright == 0) fprintf( fp_outLOG, "]" ); else fprintf( fp_outLOG, "[" );
4589 if (BSTcurrentPeak == 1) fprintf( fp_outLOG, " ROOT" );
4590 fprintf( fp_outLOG, "\n" );
4591 // BST print ]
4592 }
4593 // ----- ]
4594 }
4595 fprintf( fp_outLOG, "Above Binary-Search-Tree with MaxPEAK = %s has NODES = %s and LEAFs = %s\n", _ui64toaKAZEcomma(BSTwithMAXpeak,
11ToaDigits2, 10), _ui64toaKAZEcomma(BSTwithMAXpeakNODE, 11ToaDigits3, 10), _ui64toaKAZEcomma(BSTwithMAXpeakLEAF, 11ToaDigits, 10));
4596 fprintf( fp_outLOG, "Legend:\n" );
4597 fprintf( fp_outLOG, "At left side of the word - '[' means no left successor\n" );
4598 fprintf( fp_outLOG, "At left side of the word - '[' means left successor exists\n" );
4599 fprintf( fp_outLOG, "At right side of the word - ']' means no right successor\n" );
4600 fprintf( fp_outLOG, "At right side of the word - '[' means right successor exists\n" );
4601 // ~~~~ Longest path ~~~~ ]
4602
4603 // BST stats [
4604 PEAKibBST=1+floorLog2(BSTwithMAXnode);
4605 //PEAKibBST=1;
4606 //while (BSTwithMAXnode>>PEAKibBST) PEAKibBST++;
4607 // BST stats ]
4608 }
4609
4610 (void) time(&t2);
4611 if (t2 <= t1) {t2 = t1; t2++;}
4612 printf("Time for making unsorted wordlist: %d second(s)\n", (int) t2-t1);
4613 fprintf( fp_outLOG, "Bytes per second performance: %sB/s\n", _ui64toaKAZEcomma(FilesLEN/((int) t3-t1), 11ToaDigits, 10) ); // Rev. 12+
4614 fprintf( fp_outLOG, "Words per second performance: %sW/s\n", _ui64toaKAZEcomma(WORDcount/((int) t3-t1), 11ToaDigits, 10) ); // Rev. 12+
4615 fprintf( fp_outLOG, "Input File with a list of TEXTual Files: %s\n", argv[1] );
4616 fprintf( fp_outLOG, "Size of all TEXTual Files: %s\n", _ui64toaKAZEcomma(FilesLEN, 11ToaDigits, 10) );
4617 fprintf( fp_outLOG, "Word count: %s of them %s distinct\n", _ui64toaKAZEcomma(WORDcount, 11ToaDigits, 10), _ui64toaKAZEcomma((unsigned long
long)WORDcountDistinct, 11ToaDigits2, 10) );
4618 fprintf( fp_outLOG, "Number Of Files: %lu\n", NumberOfFiles );
4619 fprintf( fp_outLOG, "Number Of Lines: %lu\n", NumberOfLines );
4620 fprintf( fp_outLOG, "Allocated memory in MB: %lu\n", (unsigned long)(memory_size>>20)+1 );
4621 NumberOfHashCollisions = WORDcountDistinct - NumberOfTrees;
4622 fprintf( fp_outLOG, "Number Of Trees(GREATER THE BETTER): %lu\n", NumberOfTrees );
4623 fprintf( fp_outLOG, "Forest population(Hash Function Quality regarding Collisions i.e. Hash Table Utilization): %lu%s\n",
(NumberOfTrees*100)/(26*31*8192), "%0" );
4624 fprintf( fp_outLOG, "Number Of Hash Collisions(Distinct WORDs - Number Of Trees): %lu\n", NumberOfHashCollisions );
4625
4626 if (BSTorBtree == 0)
4627 {
4628 fprintf( fp_outLOG, "Maximum Attempts to Find/Put a WORD into a Binary-Search-Tree: '%s'\n", _ui64toaKAZEcomma(BSTwithMAXpeak, 11ToaDigits,
10) );
4629 fprintf( fp_outLOG, "Total Attempts to Find/Put WORDs into Binary-Search-Trees: %s\n", _ui64toaKAZEcomma(WORDcountAttemptsToPut, 11ToaDigits,
10) );
4630 fprintf( fp_outLOG, "Total Number of LEAFs in Binary-Search-Trees(GREATER THE BETTER): %s\n", _ui64toaKAZEcomma(BSTsTotalLEAFs, 11ToaDigits,
10) );
4631 fprintf( fp_outLOG, "Perfectly-Balanced-Binary-Search-Tree for MaxNODEs = %s must have PEAK = %s = rounding down of integer (1+lb(%s))\n",
_ui64toaKAZEcomma(BSTwithMAXnode, 11ToaDigits, 10), _ui64toaKAZEcomma(PEAKibBST, 11ToaDigits2, 10), _ui64toaKAZEcomma(BSTwithMAXnode,
11ToaDigits3, 10));
4632 fprintf( fp_outLOG, "Binary-Search-Tree(1st out of %s) with MaxNODEs = %s has PEAK = %s and LEAFs = %s\n",
_ui64toaKAZEcomma(BSTcurrentNodeMAXQUANTITY, 11ToaDigits4, 10), _ui64toaKAZEcomma(BSTwithMAXnode, 11ToaDigits2, 10),
_ui64toaKAZEcomma(BSTwithMAXnodePEAK, 11ToaDigits3, 10), _ui64toaKAZEcomma(BSTwithMAXnodeLEAF, 11ToaDigits, 10));
4633 fprintf( fp_outLOG, "Binary-Search-Tree(1st out of %s) with MaxPEAK = '%s' has NODEs = %s and LEAFs = %s\n",
_ui64toaKAZEcomma(BSTcurrentPeakMAXQUANTITY, 11ToaDigits4, 10), _ui64toaKAZEcomma(BSTwithMAXpeak, 11ToaDigits2, 10),
_ui64toaKAZEcomma(BSTwithMAXpeakNODE, 11ToaDigits3, 10), _ui64toaKAZEcomma(BSTwithMAXpeakLEAF, 11ToaDigits, 10));
4634 fprintf( fp_outLOG, "Binary-Search-Tree(1st out of %s) with MaxLEAFs = %s has NODEs = %s and PEAK = %s\n",

```

```

        _ui64toaKAZEcomma(BSTcurrentLeafMAXQUANTITY, 11ToaDigits4, 10), _ui64toaKAZEcomma(BSTwithMAXleaf, 11ToaDigits2, 10),
        _ui64toaKAZEcomma(BSTwithMAXleafNODE, 11ToaDigits3, 10), _ui64toaKAZEcomma(BSTwithMAXleafPEAK, 11ToaDigits, 10));
4635 } else
4636 {
4637     fprintf( fp_outLOG, "Total Attempts to Find/Put WORDS into B-trees order 3: %s\n", _ui64toaKAZEcomma(WORDcountAttemptsToPut, 11ToaDigits, 10)
    );
4638 }
4639
4640 for( k = 1; k < 32; k++ )
4641 { fprintf( fp_outLOG, "words with length %s occupy %sKB of %sKB given i.e. %s utilization\n", _ui64toaKAZEzerocomma(k, 11ToaDigits, 10)+(26-
    2), _ui64toaKAZEzerocomma((MAXusedBuffer[k]>>10)+1, 11ToaDigits2, 10)+(26-5), _ui64toaKAZEzerocomma(((GRMBLhi11[(int)k] *
    LetterBuffer)/31)>>10)+1, 11ToaDigits3, 10)+(26-5), _ui64toaKAZEzerocomma((unsigned long long)(MAXusedBuffer[k]*100)/((GRMBLhi11[(int)k] *
    LetterBuffer)/31), 11ToaDigits4, 10)+(26-2), "%0" ); // 26 are all 26-DESIRED=24
4642     if ( MAXusedBufferABS < (31 * ((MAXusedBuffer[k]>>10)+1)) / GRMBLhi11[(int)k] ) {MAXusedBufferABS = 1+(31 * ((MAXusedBuffer[k]>>10)+1)) /
    GRMBLhi11[(int)k];}
4643     Utiliza1 = Utiliza1 + (MAXusedBuffer[k]>>10)+1;
4644     Utiliza2 = Utiliza2 + (((GRMBLhi11[(int)k] * LetterBuffer)/31)>>10)+1;
4645 }
4646 fprintf( fp_outLOG, "Total pseudo(including hash table) memory utilization: %s\n", _ui64toaKAZEzerocomma((Utiliza1*100)/Utiliza2,
    11ToaDigits, 10)+(26-2), "%0" ); // 26 are all 26-DESIRED=24
4647 fprintf( fp_outLOG, "Total real(wordlist's words VS allocated block) memory utilization: %s/1000\n", _i64toaKAZE(((unsigned long
    long)TotalWLchars*1000)/memory_size, 11ToaDigits, 10) ); // 26 are all 26-DESIRED=24
4648 fprintf( fp_outLOG, "Used value for third parameter in KB: %lu\n", (unsigned long)Thunderwith );
4649 fprintf( fp_outLOG, "Use next time as third parameter: %lu\n", MAXusedBufferABS ); // 26 are all 26-DESIRED=24
4650 fprintf( fp_outLOG, "Time for making unsorted wordlist: %d second(s)\n", (int) t2-t1);
4651
4652 // EXIT EXIT EXIT EXIT EXIT EXIT EXIT EXIT EXIT EXIT EXIT EXIT EXIT EXIT EXIT
4653 printf( "Deallocated memory in MB: %lu\n", (memory_size>>20)+1 );
4654 free(pointerflushUNALIGN);
4655 fclose(fp_out);
4656 fclose(fp_outLOG);
4657 // SORT SORT SORT SORT SORT SORT SORT SORT SORT SORT SORT SORT SORT SORT SORT
4658 //printf("Uploading unsorted wordlist ...\n");
4659 if ((nlines = readlines(argv[2], &backup)) >= 0)
4660 { //printf("Number of words(lines) uploaded: %lu\n", nlines);
4661     //printf("Note1: Press 'Ctrl+C' to abort sorting, unsorted wordlist(second parameter)\n");
4662     //printf("    will remain intact(unless flushing is in progress) because of\n");
4663     //printf("    pointers-to-data are being sorted not the data itself.\n");
4664     //printf("Note2: In near future 'InsertionX26Sort' will be replaced with 'QuickX26Sort':\n");
4665     //printf("    which is much faster than 'QuickSort' applied for data-at-once!\n");
4666
4667     // !!!!! I AM DISAPPOINTED x26 is just an illusion !!!!!
4668     // x26 x26 x26 x26 x26 x26 x26 x26 x26 x26 x26 x26 x26 x26 x26 x26
4669     // argc is 4|5|6 due to eventual missing BufferSize
4670     if( argc == 4 ) // not 5 due to eventual missing BufferSize
4671     { k_FIX = 3;
4672     if( argc == 5 || argc == 6 )
4673     { k_FIX = 4;
4674     if (*argv[k_FIX] != 'A' && *argv[k_FIX] != 'a' && *argv[k_FIX] != 'B' && *argv[k_FIX] != 'b' && *argv[k_FIX] != 'C' && *argv[k_FIX] != 'c' &&
        *argv[k_FIX] != 'D' && *argv[k_FIX] != 'd')
4675     { printf("Sorting(with 'MultikeyQuickSortX26Sort' by J. Bentley and R. Sedgewick) ...\n");
4676     /* ????!! what an unexpected behavior! I have been hit: for SOED5.HTM(259,835 distinct words): InsertionX26Sort gives 46s, InsertionSort
        gives 28s */
4677     for( k = 0; k < 26; k++ )
4678     { printf( "Sort pass %s/26 ...\n", _ui64toaKAZEzerocomma(k+1, 11ToaDigits, 10)+(26-2));
4679         HEADOffsetFromStartBUKVA = TAILOffsetFromStartBUKVA;
4680         while( (TAILOffsetFromStartBUKVA < nlines) && (*backup[TAILOffsetFromStartBUKVA] - 'a' == k) )
4681         { TAILOffsetFromStartBUKVA++;
4682         }
4683         if (HEADOffsetFromStartBUKVA != TAILOffsetFromStartBUKVA)
4684         { mkqsort_main(backup + HEADOffsetFromStartBUKVA, TAILOffsetFromStartBUKVA - HEADOffsetFromStartBUKVA + 0); // backup[0..nlines-1]
4685         }
4686     }
4687     }
4688     else
4689     {
4690     if (*argv[k_FIX] == 'A' || *argv[k_FIX] == 'a')
4691     { printf("Sorting(with 'InsertionSort') ...");
4692     InsertSortKAZE(backup, nlines, 0); // backup[0..nlines-1]
4693     }
4694     if (*argv[k_FIX] == 'B' || *argv[k_FIX] == 'b')
4695     { printf("Sorting(with 'InsertionX26Sort') ...\n");
4696     /* ????!! what an unexpected behavior! I have been hit: for SOED5.HTM(259,835 distinct words): InsertionX26Sort gives 46s, InsertionSort
        gives 28s */
4697     for( k = 0; k < 26; k++ )
4698     { printf( "Sort pass %s/26 ...\n", _ui64toaKAZEzerocomma(k+1, 11ToaDigits, 10)+(26-2));
4699         HEADOffsetFromStartBUKVA = TAILOffsetFromStartBUKVA;
4700         while( (TAILOffsetFromStartBUKVA < nlines) && (*backup[TAILOffsetFromStartBUKVA] - 'a' == k) )
4701         { TAILOffsetFromStartBUKVA++;
4702         }
4703         if (HEADOffsetFromStartBUKVA != TAILOffsetFromStartBUKVA)
4704         { InsertSortKAZE(backup + HEADOffsetFromStartBUKVA, TAILOffsetFromStartBUKVA - HEADOffsetFromStartBUKVA + 0, 0); // backup[0..nlines-1]
4705         }
4706     }
4707     }
4708     if (*argv[k_FIX] == 'C' || *argv[k_FIX] == 'c')
4709     { printf("Sorting(with 'MultikeyQuickSortSort' by J. Bentley and R. Sedgewick) ...");
4710     mkqsort_main(backup, nlines); // backup[0..nlines-1]

```



```

4711 }
4712 if (*argv[k_FIX] == 'd' || *argv[k_FIX] == 'd')
4713 { printf("Sorting(with 'MultiKeyQuickSortX26Sort' by J. Bentley and R. Sedgewick) ...\n");
4714 /* ???!!! what an unexpected behavior! I have been hit: for SOED5.HTM(259,835 distinct words): InsertionX26Sort gives 46s, InsertionSort
gives 28s */
4715 for( k = 0; k < 26; k++ )
4716 { printf( "Sort pass %s/26 ...\n", _ui64toaKAZEzerocomma(k+1, 11ToaDigits, 10)+(26-2));
4717 HEADOffsetFromStartBUKVA = TAILOffsetFromStartBUKVA;
4718 while( (TAILOffsetFromStartBUKVA < nlines) && (*backup[TAILOffsetFromStartBUKVA] - 'a' == k) )
4719 { TAILOffsetFromStartBUKVA++;
4720 }
4721 if (HEADOffsetFromStartBUKVA != TAILOffsetFromStartBUKVA)
4722 { mkqsrt_main(backup + HEADOffsetFromStartBUKVA, TAILOffsetFromStartBUKVA - HEADOffsetFromStartBUKVA + 0); // backup[0..nlines-1]
4723 }
4724 }
4725 }
4726 }
4727 // x26 x26 x26 x26 x26 x26 x26 x26 x26 x26 x26 x26 x26 x26 x26 x26
4728
4729 printf("\nFlushing sorted words ...\n");
4730 if( ( fp_out = fopen( argv[2], "wb+" ) ) == NULL )
4731 { printf( "Leprechaun: Can't create file %s \n", argv[2] ); return( 1 ); }
4732 for( j = 0; j < nlines; j++ )
4733 { //Slot = KuxHash3plus(backup[j]);
4734 //fprintf(fp_out, "Hashcode: %s - ", _ui64toaKAZEzerocomma(Slot, 11ToaDigits, 10)+(26-5));
4735 fprintf(fp_out, "%s", backup[j]);
4736 fwrite(CRDLFa, 2, 1, fp_out );
4737 }
4738 (void) time(&t3);
4739 if (t3 <= t2) {t3 = t2; t3++;}
4740 printf("Time for sorting unsorted wordlist: %d second(s)\n", (int) t3-t2);
4741
4742 /*
4743 // Hash benchmarking ----- [
4744
4745 // 5[
4746 clocks1 = clock();
4747 for (Bozan=0; Bozan < (1<<4); Bozan++) // 16 times, at end >>4
4748 {
4749 for( j = 0; j < nlines; j++ )
4750 { //Slot = KuxHash3plus(backup[j]);
4751 //fprintf(fp_out, "Hashcode: %s - ", _ui64toaKAZEzerocomma(Slot, 11ToaDigits, 10)+(26-5));
4752 Slot = FNV1A_Hash_4_OCTETS(backup[j], (strlen(backup[j])>>2)); //13+++
4753 }
4754 }
4755 clocks2 = clock();
4756 printf( "Performance of 'FNV1A_Hash_4_OCTETS': %lu words/clock or %lu MB/s\n", (nlines/((long)(clocks2 - clocks1 + 1)>>4)) ,
((TotalWLchars>>10)/((long)(clocks2 - clocks1 + 1)>>4)) );
4757 // 5]
4758
4759 // 1[
4760 clocks1 = clock();
4761 for (Bozan=0; Bozan < (1<<4); Bozan++) // 16 times, at end >>4
4762 {
4763 for( j = 0; j < nlines; j++ )
4764 { //Slot = KuxHash3plus(backup[j]);
4765 //fprintf(fp_out, "Hashcode: %s - ", _ui64toaKAZEzerocomma(Slot, 11ToaDigits, 10)+(26-5));
4766 // To make it EVEN !!!
4767 //wrdlen = strlen(backup[j]);
4768 //if (strlen(backup[j]) != 0)
4769 Slot = FNV1A_Hash(backup[j]); //13+++
4770 }
4771 }
4772 clocks2 = clock();
4773 printf( "Performance of 'FNV1A_Hash': %lu words/clock or %lu MB/s\n", (nlines/((long)(clocks2 - clocks1 + 1)>>4)) ,
((TotalWLchars>>10)/((long)(clocks2 - clocks1 + 1)>>4)) );
4774 // 1]
4775
4776 // 2[
4777 clocks1 = clock();
4778 for (Bozan=0; Bozan < (1<<4); Bozan++) // 16 times, at end >>4
4779 {
4780 for( j = 0; j < nlines; j++ )
4781 { //Slot = KuxHash3plus(backup[j]);
4782 //fprintf(fp_out, "Hashcode: %s - ", _ui64toaKAZEzerocomma(Slot, 11ToaDigits, 10)+(26-5));
4783 Slot = FNV1A_Hash_4_OCTETS_31(backup[j], (strlen(backup[j])>>2)); //13+++
4784 }
4785 }
4786 clocks2 = clock();
4787 printf( "Performance of 'FNV1A_Hash_4_OCTETS_31': %lu words/clock or %lu MB/s\n", (nlines/((long)(clocks2 - clocks1 + 1)>>4)) ,
((TotalWLchars>>10)/((long)(clocks2 - clocks1 + 1)>>4)) );
4788 // 2]
4789
4790 // 4[
4791 clocks1 = clock();
4792 for (Bozan=0; Bozan < (1<<4); Bozan++) // 16 times, at end >>4
4793 {
4794 for( j = 0; j < nlines; j++ )

```

```

4795     { //Slot = KuxHash3plus(backup[j]);
4796         //fprintf(fp_out, "Hashcode: %s - ", _ui64toaKAZEzerocomma(Slot, 11ToaDigits, 10)+(26-5));
4797 // To make it EVEN !!!
4798 //wrklen = strlen(backup[j]);
4799 //if (strlen(backup[j]) != 0)
4800     Slot = KuxHash3plus(backup[j]); //13++
4801     }
4802 }
4803 clocks2 = clock();
4804 printf( "Performance of 'KuxHash3plus': %lu words/clock or %lu MB/s\n", (nlines/((long)(clocks2 - clocks1 + 1)>>4)) ,
         ((TotalWLchars>>10)/((long)(clocks2 - clocks1 + 1)>>4)) );
4805 // 4]
4806
4807 // 6[
4808 clocks1 = clock();
4809     for (Bozan=0; Bozan < (1<<4); Bozan++) // 16 times, at end >>4
4810     {
4811         for( j = 0; j < nlines; j++ )
4812             { //Slot = KuxHash3plus(backup[j]);
4813                 //fprintf(fp_out, "Hashcode: %s - ", _ui64toaKAZEzerocomma(Slot, 11ToaDigits, 10)+(26-5));
4814                 wrklen = strlen(backup[j]);
4815                 if (wrklen<=19) // 4x4+3=19 i.e. last contains 7 clashes
4816                     Slot = FNV1A_Hash_Granularity(backup[j], wrklen>>2, 2); //13++++
4817                 else // 2x8+4=20 i.e. first contains 6 clashes
4818                     Slot = FNV1A_Hash_Granularity(backup[j], wrklen>>3, 3); //13++++
4819             } // Conclusion: two functions > 64 bytes lead to horrible slowness, so unite them in one: fit in the cache line.
4820     }
4821     clocks2 = clock();
4822     printf( "Performance of 'FNV1A_Hash_Granularity': %lu words/clock or %lu MB/s\n", (nlines/((long)(clocks2 - clocks1 + 1)>>4)) ,
         ((TotalWLchars>>10)/((long)(clocks2 - clocks1 + 1)>>4)) );
4823 // 6]
4824
4825 // Hash benchmarking ----- ]
4826 */
4827
4828 if( ( fp_outLOG = fopen( "Leprechaun.LOG", "a" ) ) == NULL )
4829 { printf( "Leprechaun: Can't open file Leprechaun.LOG.\n" ); return( 1 ); }
4830 fprintf( fp_outLOG, "Time for sorting unsorted wordlist: %d second(s)\n\n", (int) t3-t2);
4831 printf( "Leprechaun: Done.\n" );
4832     return 0;
4833 }
4834 else
4835 { printf("Leprechaun: Input file too large, wordlist remains unsorted!\n");
4836     return 1;
4837 }
4838 // SORT SORT SORT SORT SORT SORT SORT SORT SORT SORT SORT SORT SORT SORT
4839 } else { //if (BSTorBtree != 2) {
4840 // External Btrees [
4841
4842 printf("Flushing Unsorted words...\n");
4843 if( ( fp_out = fopen( argv[2], "wb+" ) ) == NULL )
4844 { printf( "Leprechaun: Can't create file %s\n", argv[2] ); return( 1 ); }
4845 CRDLFa[0] = 13; CRDLFa[1] = 10;
4846
4847 BufStart = pointerflush;
4848 // for( j = 0; j < 28*28*28*28; j++ )
4849 for( j = 0; j < 134217728; j++ )
4850 {
4851     Slot = j<<3;
4852     memcpy( &PseudoLinkedPointer_64, BufStart+Slot, 8 );
4853     if (PseudoLinkedPointer_64 != 0)
4854     {
4855         NumberOfTrees++;
4856
4857 // $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$ B-tree order 3 traverse 64bit [
4858 // DONE JOB:
4859 // Must be written B-tree traverse ! with simulated stack i.e. non-recursive.
4860 // ...
4861 StackPtr = 0;
4862 while ( 2==2 ) {
4863     while (PseudoLinkedPointer_64 != 0)
4864     {
4865         if (StackPtr > 8192*3-1) { printf( "\nLeprechaun: Failure! B-tree simulated stack overflow, too high B-tree!\n" ); return( 13
4866     );}
4867         BSTstack[StackPtr] = PseudoLinkedPointer_64; ++StackPtr; //ptr to Rword
4868 //if ( *(char *) (PseudoLinkedPointer + 4 + 4 + i%31+1) == 0 ) {memcpy( PseudoLinkedPointer + 4 + 4, &BufStart[NumberOfSLOTS*4], 4 );}
4869         PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1); //RW
4870         fsetpos(fp_outRG, &PseudoLinkedPointerAUX_64);
4871         fread(&SomeByte, 1, 1, fp_outRG);
4872         if (SomeByte == 0) // RW exists not
4873         {
4874             PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8;
4875             fsetpos(fp_outRG, &PseudoLinkedPointerAUX_64);
4876             fwrite(&NULLs_64, 8, 1, fp_outRG);
4877         }
4878         memcpy( &PseudoLinkedPointerNEWleft, PseudoLinkedPointer, 4 );
4879         memcpy( &PseudoLinkedPointerNEWMiddle, PseudoLinkedPointer + 4, 4 );
4880         memcpy( &PseudoLinkedPointerNEWright, PseudoLinkedPointer + 4 + 4, 4 );

```

```

4880 PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64;
4881 fsetpos(fp_outRG, &PseudoLinkedPointerAUX_64);
4882 fread(&PseudoLinkedPointerNEWleft_64, 8, 1, fp_outRG);
4883 PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8;
4884 fsetpos(fp_outRG, &PseudoLinkedPointerAUX_64);
4885 fread(&PseudoLinkedPointerNEWmiddle_64, 8, 1, fp_outRG);
4886 PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8;
4887 fsetpos(fp_outRG, &PseudoLinkedPointerAUX_64);
4888 fread(&PseudoLinkedPointerNEWright_64, 8, 1, fp_outRG);
4889 // Give first from right to left non-zero PTR
4890 if (PseudoLinkedPointerNEWright_64 !=0 )
4891 { memcpy( PseudoLinkedPointer + 4 + 4, &BufStart[NumberOfSLOTS*4], 4 );
4892 // PseudoLinkedPointer = PseudoLinkedPointerNEWright;
4893 // }
4894 {
4895 PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8;
4896 fsetpos(fp_outRG, &PseudoLinkedPointerAUX_64);
4897 fwrite(&NULLs_64, 8, 1, fp_outRG);
4898 PseudoLinkedPointer_64 = PseudoLinkedPointerNEWright_64;
4899 }
4900 else if (PseudoLinkedPointerNEWmiddle_64 !=0 )
4901 // { memcpy( PseudoLinkedPointer + 4, &BufStart[NumberOfSLOTS*4], 4 );
4902 // PseudoLinkedPointer = PseudoLinkedPointerNEWmiddle;
4903 // }
4904 {
4905 PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8;
4906 fsetpos(fp_outRG, &PseudoLinkedPointerAUX_64);
4907 fwrite(&NULLs_64, 8, 1, fp_outRG);
4908 PseudoLinkedPointer_64 = PseudoLinkedPointerNEWmiddle_64;
4909 }
4910 else if (PseudoLinkedPointerNEWleft_64 !=0 )
4911 // { memcpy( PseudoLinkedPointer, &BufStart[NumberOfSLOTS*4], 4 );
4912 // PseudoLinkedPointer = PseudoLinkedPointerNEWleft;
4913 // }
4914 {
4915 PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64;
4916 fsetpos(fp_outRG, &PseudoLinkedPointerAUX_64);
4917 fwrite(&NULLs_64, 8, 1, fp_outRG);
4918 PseudoLinkedPointer_64 = PseudoLinkedPointerNEWleft_64;
4919 }
4920 else
4921 {
4922 PseudoLinkedPointer_64 = 0;
4923 }
4924 }
4925 if (StackPtr == 0) break;
4926 PseudoLinkedPointer_64 = BSTstack[--StackPtr];
4927 // memcpy( &PseudoLinkedPointerNEWleft, PseudoLinkedPointer, 4 );
4928 // memcpy( &PseudoLinkedPointerNEWmiddle, PseudoLinkedPointer + 4, 4 );
4929 // memcpy( &PseudoLinkedPointerNEWright, PseudoLinkedPointer + 4 + 4, 4 );
4930 PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64;
4931 fsetpos(fp_outRG, &PseudoLinkedPointerAUX_64);
4932 fread(&PseudoLinkedPointerNEWleft_64, 8, 1, fp_outRG);
4933 PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8;
4934 fsetpos(fp_outRG, &PseudoLinkedPointerAUX_64);
4935 fread(&PseudoLinkedPointerNEWmiddle_64, 8, 1, fp_outRG);
4936 PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8;
4937 fsetpos(fp_outRG, &PseudoLinkedPointerAUX_64);
4938 fread(&PseudoLinkedPointerNEWright_64, 8, 1, fp_outRG);
4939 if (PseudoLinkedPointerNEWleft_64 + PseudoLinkedPointerNEWmiddle_64 + PseudoLinkedPointerNEWright_64 == 0) // One LEAF is PRINTED when
LP=0 MP=0 RP=0
4940 {
4941 // memcpy( wrd, PseudoLinkedPointer + 4 + 4 + 4, i%31+1 );
4942 // fwrite(wrd, i%31+1, 1, fp_out);
4943 // fwrite(CRDLFa, 2, 1, fp_out);
4944 // if ( *(char *) (PseudoLinkedPointer + 4 + 4 + 4 + i%31+1) != 0 )
4945 // { memcpy( wrd, PseudoLinkedPointer + 4 + 4 + 4 + i%31+1, i%31+1 );
4946 // fwrite(wrd, i%31+1, 1, fp_out);
4947 // fwrite(CRDLFa, 2, 1, fp_out);
4948 // }
4949 // PseudoLinkedPointer = 0;
4950 PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8;
4951 fsetpos(fp_outRG, &PseudoLinkedPointerAUX_64);
4952 fread(&wrd[0], (LongestLineInclusive+1), 1, fp_outRG);
4953 fprintf(fp_out, "%s", wrd);
4954 fwrite(CRDLFa, 2, 1, fp_out);
4955 PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1); //RW
4956 fsetpos(fp_outRG, &PseudoLinkedPointerAUX_64);
4957 fread(&SomeByte, 1, 1, fp_outRG);
4958 if (SomeByte != 0 ) // RW exists
4959 {
4960 PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1);
4961 fsetpos(fp_outRG, &PseudoLinkedPointerAUX_64);
4962 fread(&wrd[0], (LongestLineInclusive+1), 1, fp_outRG);
4963 fprintf(fp_out, "%s", wrd);
4964 fwrite(CRDLFa, 2, 1, fp_out);
4965 }
4966 PseudoLinkedPointer_64 = 0;

```

```

4967     }
4968 }
4969 // $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$ B-tree order 3 traverse 64bit ]
4970
4971     }
4972 }
4973
4974 fclose(fp_out);
4975
4976 // TO DO ...
4977 //printf( "Leprechaun: THE DUMP NOT SORTED?! HASH sucks also - these two issues ought to be fixed in r.14. This is revision 14-.\n" );
4978
4979 fprintf( fp_outLOG, "Number Of Trees(GREATER THE BETTER): %s\n", _ui64toaKAZEcomma(NumberOfTrees, 11ToaDigits, 10) );
4980 fprintf( fp_outLOG, "Used value for third parameter in KB: %s\n", _ui64toaKAZEcomma(Thunderwith, 11ToaDigits, 10) );
4981 fprintf( fp_outLOG, "Use next time as third parameter: %s\n", _ui64toaKAZEcomma(((BufEnd_64+1)>>10)+1, 11ToaDigits, 10) );
4982
4983 // External Btrees ]
4984 free(pointerflushUNALIGN);
4985 fclose(fp_outRG);
4986 fclose(fp_outLOG);
4987 printf( "Leprechaun: Done.\n" );
4988 exit(0);
4989 } //if (BSTorBtree != 2) {
4990 } // main()
4991
4992 /*
4993 TO BE DONE: Ideal Balancing BST [
4994
4995 link rotR(link h)
4996 { link x = h->l; h->l = x->r; x->r = h;
4997   return x; }
4998
4999 link rotL(link h)
5000 { link x = h->r; h->r = x->l; x->l = h;
5001   return x; }
5002
5003 link parTR(link h, int k)
5004 { int t = h->l->N;
5005   if (t > k)
5006     { h->l = parTR(h->l, k); h = rotR(h); }
5007   if (t < k)
5008     { h->r = parTR(h->r, k-t-1); h = rotL(h); }
5009   return h;
5010 }
5011
5012 link balancer(link h)
5013 {
5014   if (h->N < 2) return h;
5015   h = parTR(h, h->N/2);
5016   h->l = balancer(h->l);
5017   h->r = balancer(h->r);
5018   return h;
5019 }
5020
5021 TO BE DONE: Ideal Balancing BST ]
5022 */
5023
5024 /*
5025 #include <stdlib.h>
5026 #include "Item.h"
5027 typedef struct STnode* link;
5028 struct STnode { Item item; link l, r; int N };
5029 static link head, z;
5030 link NEW(Item item, link l, link r, int N)
5031 { link x = malloc(sizeof *x);
5032   x->item = item; x->l = l; x->r = r; x->N = N;
5033   return x;
5034 }
5035 void STinit()
5036 { head = (z = NEW(NULLitem, 0, 0, 0)); }
5037 int STcount() { return head->N; }
5038 Item searchR(link h, Key v)
5039 { Key t = key(h->item);
5040   if (h == z) return NULLitem;
5041   if (eq(v, t) return h->item;
5042   if (less(v, t) return searchR(h->l, v);
5043   else return searchR(h->r, v);
5044 }
5045 Item STsearch(Key v)
5046 { return searchR(head, v); }
5047 link insertR(link h, Item item)
5048 { Key v = key(item), t = key(h->item);
5049   if (h == z) return NEW(item, z, z, 1);
5050   if (less(v, t)
5051     h->l = insertR(h->l, item);
5052   else h->r = insertR(h->r, item);
5053   (h->N)++; return h;
5054 }

```

```

5055 void STinsert(Item item)
5056 { head = insertR(head, item); }
5057 */
5058
5059 /*
5060 int count(link h)
5061 {
5062     if (h == NULL) return 0;
5063     return count(h->l) + count(h->r) + 1;
5064 }
5065
5066 int height(link h)
5067 { int u, v;
5068     if (h == NULL) return -1;
5069     u = height(h->l); v = height(h->r);
5070     if (u > v) return u+1; else return v+1;
5071 }
5072
5073 void printnode(char c, int h)
5074 { int i;
5075     for (i = 0; i < h; i++) printf(" ");
5076     printf("%c\n", c);
5077 }
5078
5079 void show(link x, int h)
5080 {
5081     if (x == NULL) { printnode("x", h); return; }
5082     show(x->r, h+1);
5083     printnode(x->item, h);
5084     show(x->l, h+1);
5085 }
5086 */

```

D:_KAZE~1\LEPREC~1\LEPREC~1>cl /Ox /wp64 /Tcleprechaun_quadrupleton.c /FaLeprechaun_quadrupleton /Facs
Microsoft (R) 32-bit C/C++ Optimizing Compiler Version 16.00.30319.01 for 80x86
Copyright (C) Microsoft Corporation. All rights reserved.

cl : Command line warning D9035 : option 'wp64' has been deprecated and will be removed in a future release

Leprechaun_quadrupleton.c
Leprechaun_quadrupleton.c(857) : warning C4312: 'type cast' : conversion from 'int' to 'string' of greater size
Leprechaun_quadrupleton.c(877) : warning C4312: 'type cast' : conversion from 'int' to 'string *' of greater size
Leprechaun_quadrupleton.c(2755) : warning C4312: 'type cast' : conversion from 'int' to 'char *' of greater size
Leprechaun_quadrupleton.c(2771) : warning C4311: 'type cast' : pointer truncation from 'char *' to 'unsigned long'
Leprechaun_quadrupleton.c(2776) : warning C4311: 'type cast' : pointer truncation from 'char *' to 'unsigned long'
Leprechaun_quadrupleton.c(2801) : warning C4312: 'type cast' : conversion from 'int' to 'char *' of greater size
Leprechaun_quadrupleton.c(3221) : warning C4312: 'type cast' : conversion from 'unsigned long' to 'char *' of greater size
Leprechaun_quadrupleton.c(3420) : warning C4312: 'type cast' : conversion from 'unsigned long' to 'char *' of greater size
Leprechaun_quadrupleton.c(3476) : warning C4312: 'type cast' : conversion from 'unsigned long' to 'char *' of greater size
Leprechaun_quadrupleton.c(3507) : warning C4312: 'type cast' : conversion from 'unsigned long' to 'char *' of greater size
Leprechaun_quadrupleton.c(3513) : warning C4312: 'type cast' : conversion from 'unsigned long' to 'char *' of greater size
Leprechaun_quadrupleton.c(3518) : warning C4312: 'type cast' : conversion from 'unsigned long' to 'char *' of greater size
Leprechaun_quadrupleton.c(3546) : warning C4312: 'type cast' : conversion from 'unsigned long' to 'char *' of greater size
Leprechaun_quadrupleton.c(3579) : warning C4312: 'type cast' : conversion from 'unsigned long' to 'char *' of greater size
Leprechaun_quadrupleton.c(3593) : warning C4312: 'type cast' : conversion from 'unsigned long' to 'char *' of greater size
Leprechaun_quadrupleton.c(3605) : warning C4312: 'type cast' : conversion from 'unsigned long' to 'char *' of greater size
Microsoft (R) Incremental Linker Version 10.00.30319.01
Copyright (C) Microsoft Corporation. All rights reserved.

/out:Leprechaun_quadrupleton.exe
Leprechaun_quadrupleton.obj

D:_KAZE~1\LEPREC~1\LEPREC~1>

```

0001 // QuickSortExternal_4+GB r.2+, copleyleft 2011 May 08, Kaze. A pseudo-problem exists in r.1++ and r.2, namely the sorted file is smaller than
the incoming one (for example for graffith_L.log), bug?!
0002
0003 /*
0004 [kaze@gea ~]$ cd /home/kaze/r2
0005 [kaze@gea r2]$ ls -l
0006 -rwxrwxrwx 1 kaze kaze 7773570191 2011-05-11 05:46 GRAFFITH_T.log
0007 -rwxrwxrwx 1 kaze kaze 129 2011-05-13 00:18 Linux_QuickSortExternal_4+GB_Compile_Line.script
0008 -rwxrwxrwx 1 kaze kaze 103015 2011-05-13 01:01 QuickSortExternal_4+GB.c
0009 [kaze@gea r2]$ ./Linux_QuickSortExternal_4+GB_Compile_Line.script
0010 QuickSortExternal_4+GB.c: In function 'CompareStringsEndingWith13':
0011 QuickSortExternal_4+GB.c:757: warning: passing argument 2 of 'fsetpos' from incompatible pointer type
0012 /usr/include/stdio.h:783: note: expected 'const struct fpos_t *' but argument is of type 'long long unsigned int *'
0013 QuickSortExternal_4+GB.c:770: warning: passing argument 2 of 'fsetpos' from incompatible pointer type
0014 /usr/include/stdio.h:783: note: expected 'const struct fpos_t *' but argument is of type 'long long unsigned int *'
0015 QuickSortExternal_4+GB.c: In function 'CompareStringsEndingWith13_EXTERNAL':
0016 QuickSortExternal_4+GB.c:823: warning: passing argument 2 of 'fsetpos' from incompatible pointer type
0017 /usr/include/stdio.h:783: note: expected 'const struct fpos_t *' but argument is of type 'long long unsigned int *'
0018 QuickSortExternal_4+GB.c:836: warning: passing argument 2 of 'fsetpos' from incompatible pointer type
0019 /usr/include/stdio.h:783: note: expected 'const struct fpos_t *' but argument is of type 'long long unsigned int *'
0020 QuickSortExternal_4+GB.c: In function 'main':
0021 QuickSortExternal_4+GB.c:2022: warning: passing argument 2 of 'fsetpos' from incompatible pointer type
0022 /usr/include/stdio.h:783: note: expected 'const struct fpos_t *' but argument is of type 'long long unsigned int *'
0023 QuickSortExternal_4+GB.c:2041: warning: passing argument 2 of 'fsetpos' from incompatible pointer type
0024 /usr/include/stdio.h:783: note: expected 'const struct fpos_t *' but argument is of type 'long long unsigned int *'
0025 QuickSortExternal_4+GB.c:2502: warning: passing argument 2 of 'fsetpos' from incompatible pointer type
0026 /usr/include/stdio.h:783: note: expected 'const struct fpos_t *' but argument is of type 'long long unsigned int *'
0027 QuickSortExternal_4+GB.c:2554: warning: passing argument 2 of 'fsetpos' from incompatible pointer type
0028 /usr/include/stdio.h:783: note: expected 'const struct fpos_t *' but argument is of type 'long long unsigned int *'
0029 QuickSortExternal_4+GB.c:2581: warning: passing argument 2 of 'fsetpos' from incompatible pointer type
0030 /usr/include/stdio.h:783: note: expected 'const struct fpos_t *' but argument is of type 'long long unsigned int *'
0031 QuickSortExternal_4+GB.c:2602: warning: passing argument 2 of 'fsetpos' from incompatible pointer type
0032 /usr/include/stdio.h:783: note: expected 'const struct fpos_t *' but argument is of type 'long long unsigned int *'
0033 [kaze@gea r2]$ ./QuickSortExternal_4+GB_r2+_generic_64bits.elf
0034 QuickSortExternal_4+GB r.2+, written by Kaze.
0035 Usage: QuickSortExternal_4+GB wordlistfile {/fast|/slow}
0036 Note1: The lines in wordlistfile must be up to 2048 chars/bytes and end with CRLF i.e. maximum 2050bytes long.
0037 Note2: The goal is to sort a file unfittable in physical memory, and with 32bit code.
0038 Note3: when /fast is specified wordlistfile is uploaded into internal memory if it is possible.
0039 Note4: when /slow is specified wordlistfile is NOT uploaded into internal memory even if it is possible.
0040 [kaze@gea r2]$ ./Corpus_script_T.txt
0041 QuickSortExternal_4+GB r.2+, written by Kaze.
0042 Size of input file: 7,773,570,191
0043 Counting lines ...
0044 Allocated memory for pointers-to-words in MB: 2602
0045 Assigning pointers ...
0046 Trying to allocate memory for the file itself in MB: 7414 ... UNSUCCESSFUL! Get on with slow external accesses.
0047 Sorting 340,981,845 Pointers ...
0048 Pass #1: Quicksort started ...
0049 | RightEnd-LeftEnd: 000,000,000,019; NumberofSplittings: 0,029,549,542 ...
0050 Pass #2: Insertionsort started ...
0051 | i: 000,340,981,845 ...
0052 NumberOfComparisons: 11,329,560,863
0053 The time to sort 340,981,845 items via Quicksort+Insertionsort was 38,001,370,000 clocks.
0054 Dumping the sorted data ...
0055 Dumped 340,981,845 lines.
0056 OK! Incoming and resultant file's sizes match.
0057 Dumping the sorted data [deduplicated] ...
0058 Dumped 115,672,221 distinct lines.
0059 Total time: 41,224,560,000 clocks.
0060 Performance: 184 KB/s.
0061 Done successfully.
0062 [kaze@gea r2]$
0063
0064 [kaze@gea ~]$ cd r2/
0065 [kaze@gea r2]$ ls -l
0066 -rwxrwxrwx 1 kaze kaze 5641065963 2011-05-13 01:20 GRAFFITH_A.log
0067 -rwxrwxr-x 1 kaze kaze 719059 2011-05-13 00:34 QuickSortExternal_4+GB_r2+_generic_64bits.elf
0068 [kaze@gea r2]$ ./Corpus_script_A.txt
0069 QuickSortExternal_4+GB r.2+, written by Kaze.
0070 Size of input file: 5,641,065,963
0071 Counting lines ...
0072 Allocated memory for pointers-to-words in MB: 1898
0073 Assigning pointers ...
0074 UNSUCCESSFUL! Get on with slow external accesses.
0075 Sorting 248,673,965 Pointers ...
0076 Pass #1: Quicksort started ...
0077 / RightEnd-LeftEnd: 000,000,000,025; NumberofSplittings: 0,021,635,123 ...
0078 Pass #2: Insertionsort started ...
0079 \ i: 000,248,673,965 ...
0080 NumberOfComparisons: 7,999,804,609
0081 The time to sort 248,673,965 items via Quicksort+Insertionsort was 25,793,490,000 clocks.
0082 Dumping the sorted data ...
0083 Dumped 248,673,965 lines.
0084 OK! Incoming and resultant file's sizes match.
0085 Dumping the sorted data [deduplicated] ...
0086 Dumped 92,148,559 distinct lines.
0087 Total time: 28,007,460,000 clocks.

```

```

0088 Performance: 196 KB/s.
0089 Done successfully.
0090 [kaze@gea r2]$
0091 */
0092
0093 /*
0094 Following data is taken during execution i.e. before going to prompt:
0095
0096     1,141,536,585 GRAFFITH_N.log
0097     1,141,534,720 QuickSortExternal_4+GB.txt
0098
0099 D:\_KAZE_new-stuff\_BUG_HIDDEN>sort GRAFFITH_N.log /o GRAFFITH_N.log.WIN
0100     1,141,536,585 GRAFFITH_N.log.WIN
0101
0102 D:\_KAZE_new-stuff\_BUG_HIDDEN>fc GRAFFITH_N.log.WIN QuickSortExternal_4+GB.txt
0103 Comparing files GRAFFITH_N.log.WIN and QUICKSORTEXTTERNAL_4+GB.TXT
0104 Resync Failed. Files are too different.
0105 ***** GRAFFITH_N.log.WIN
0106 nzxd_xxtm_nxt_x
0107 nzxd_xxtnsz_ze_kn
0108 nzxdvx_t_k_p
0109 nzxdvx_t_kj_p
0110 nzxdvx_t_u_u
0111 nzxdvx_tk_k_xzbt
0112 nzxdvx_tm_nxtpxnrn_bt
0113 nzxdvx_tm_nxtpxnrn_btk
0114 nzxdvx_tpnb_k_s
0115 nzxdvx_tq_k_x
0116 nzxdvxxt_kj_z_mnk
0117 nzxdvxxtkj_z_m_kxmn
0118 nzxf_nz_u_nxyo
0119 nzxf_nzucnxhor_morx_u
0120 nzxfz_zt_x_zt
0121 nzxir_nq_ahi_ocow
0122 nzxt_can_do_a
0123 nzxt_case_with_w
0124 nzxt_has_the_answer
0125 nzxt_khaos_case_with
0126 nzxt_khaos_corsair_hx
0127 nzxt_lexa_blackline_case
0128 nzxt_s_tempest_out
0129 nzxt_sentry_we_ve
0130 nzxt_tempest_how_the
0131 nzxt_tempest_if_you
0132 nzxt_x_nemesis_tower
0133 nzxt_x_nemesis_tower
0134 nzxt_zero_full_tower
0135 nzxt_zero_ii_a
0136 nzxt_zero_ii_is
0137 nzxt_zero_ii_maximumpc
0138 nzxt_zero_ii_s
0139 nzxt_zero_watt_full
0140 nzxtw_mod_see_through
0141 nzxyz_ztsh_x_c
0142 nzy_dx_dy_dz
0143 nzy_dx_dy_dz
0144 nzy_loefj_jmjoy_lkp
0145 nzy_loefj_jmjoy_lkpa
0146 nzy_nn_nno_njz
0147 nzy_ntcwnw_wah_hbxldzgf
0148 nzy_nze_oz_input
0149 nzy_nzx_nxz_nxy
0150 nzy_on_hjwaoogbojy_ejy
0151 nzy_ywny_i_n
0152 nzybq_jmnzo_wxy_wljmp
0153 nzycm_supplemented_with_a
0154 nzycm_without_casamino_acids
0155 nzyd_lbny_hnby_ym
0156 nzyd_vyb_jsv_mn
0157 nzym_nzym_without_casamino
0158 nzym_without_yeast_extract
0159 nzymatic_background_reaction_to
0160 nzymatic_filling_of_a
0161 nzymatic_filling_of_a
0162 nzymatic_reaction_to_form
0163 nzyme_called_a_required
0164 nzyme_called_a_required
0165 nzyme_dna_from_any
0166 nzyme_dna_from_any
0167 nzyme_l_inked_i
0168 nzyme_l_inked_i
0169 nzyme_of_a_novice
0170 nzyme_of_a_novice
0171 nzyme_responsible_for_transport
0172 nzyme_responsible_for_transport
0173 nzyme_that_breaks_internal
0174 nzyme_that_breaks_internal
0175 nzyme_that_breaks_phos

```

0176 nzyme_that_breaks_phos
 0177 nzyme_that_catalyze_sspe
 0178 nzyme_that_catalyze_sspe
 0179 nzyme_that_introduces_or
 0180 nzyme_that_introduces_or
 0181 nzyme_that_is_needed
 0182 nzyme_that_is_needed
 0183 nzymes_in_the_b
 0184 nzyr_lhym_ny_mb
 0185 nzyr_lhym_yhyh_hnjr
 0186 nzyrk_l_tb_r
 0187 nzyryh_mslg_xv_mx1b
 0188 nzythologie_et_d_archiologie
 0189 nzz_bck_qymcq_swr
 0190 nzz_came_after_nz
 0191 nzz_i_s_fhenf
 0192 nzz_nx_sh_psh
 0193 nzz_tzzvf_u_l_de1
 0194 nzz_zt_xxf_x
 0195 nzzmp_eskd_writing_the
 0196 ***** QUICKSORTEXTERNAL_4+GB.TXT
 0197 nzxd_xxtm_nxt_x
 0198 nts_the
 0199 nts_the
 0200 nts_the
 0201 nts_the
 0202 nts_the
 0203 nts_the
 0204 nts_the
 0205 nts_the
 0206 nts_the
 0207 nts_the
 0208 nts_the
 0209 nts_the
 0210 nts_the
 0211 nts_the
 0212 nts_the
 0213 nts_the
 0214 nts_the
 0215 nts_the
 0216 nts_the
 0217 nts_the
 0218 nts_the
 0219 nts_the
 0220 nts_the
 0221 nts_the
 0222 nts_the
 0223 nts_the
 0224 nts_the
 0225 nts_the
 0226 nts_the
 0227 nts_the
 0228 nts_the
 0229 nts_the
 0230 nts_the
 0231 nts_the
 0232 nts_the
 0233 nts_the
 0234 nts_the
 0235 nts_the
 0236 nts_the
 0237 nts_the
 0238 nts_the
 0239 nts_the
 0240 nts_the
 0241 nts_the
 0242 nts_the
 0243 nts_the
 0244 nts_the
 0245 nts_the
 0246 nts_the
 0247 nts_the
 0248 nts_the
 0249 nts_the
 0250 nts_the
 0251 nts_the
 0252 nts_the
 0253 nts_the
 0254 nts_the
 0255 nts_the
 0256 nts_the
 0257 nts_the
 0258 nts_the
 0259 nts_the
 0260 nts_the
 0261 nts_the
 0262 nts_the
 0263 nts_the


```

0264 nts_the
0265 nts_the
0266 nts_the
0267 nts_the
0268 nts_the
0269 nts_the
0270 nts_the
0271 nts_the
0272 nts_the
0273 nts_the
0274 nts_the
0275 nts_the
0276 nts_the
0277 nts_the
0278 nts_the
0279 nts_the
0280 nts_the
0281 nts_the
0282 nts_the
0283 nts_the
0284 nts_the
0285 nts_the
0286 nts_the
0287 nts_the
0288 nts_the
0289 nts_the
0290 nts_the
0291 nts_the
0292 nts_the
0293 nts_the
0294 nts_the
0295 nts_the
0296 nts_the
0297 *****
0298
0299 204-106+1=99 'nts_the' lines
0300
0301 D:\_KAZE_new-stuff\BUG_HIDDEN>dir GRAFFITH_N.log.WIN/b>q
0302
0303 D:\_KAZE_new-stuff\BUG_HIDDEN>Linereporter.exe q
0304 Linereporter, revision 1+, written by Kaze.
0305 Purpose: Reports number of lines(LFs) in files from a given filelist.
0306 Example:
0307 D:\>Linereporter.exe LQ2048.lst
0308 Note: Files can exceed 4GB limit.
0309 Counting ...
0310 Linereporter: Encountered lines in all files: 50,360,855
0311 Linereporter: Longest line: 32
0312
0313 D:\_KAZE_new-stuff\BUG_HIDDEN>dir "QuickSortExternal_4+GB.txt"/b>q
0314
0315 D:\_KAZE_new-stuff\BUG_HIDDEN>Linereporter.exe q
0316 Linereporter, revision 1+, written by Kaze.
0317 Purpose: Reports number of lines(LFs) in files from a given filelist.
0318 Example:
0319 D:\>Linereporter.exe LQ2048.lst
0320 Note: Files can exceed 4GB limit.
0321 Counting ...
0322 Linereporter: Encountered lines in all files: 50,360,766
0323 Linereporter: Longest line: 32
0324
0325 50,360,855-50,360,766=89 lines shorter than original
0326
0327 D:\_KAZE_new-stuff\BUG_HIDDEN>type "QuickSortExternal_4+GB.txt"
0328 ...
0329 nzx_o_modulus_kcodb
0330 nzx_o_modulus_kcode
0331 nzx_produced_by_the
0332 nzx_sin_nkq_dx
0333 nzx_there_are_jtsoil
0334 nzx_x_x_rotations
0335 nxd_xxt_k_k
0336 nxd_xxt_k_ua
0337 nxd_xxt_u_upu
0338 nxd_xxtm_nxt_x
0339 n
0340 D:\_KAZE_new-stuff\BUG_HIDDEN>
0341
0342 !!! why sometimes the cache is not flushed ??? !!! With immediate fclose maybe this pseudo-bug will vanish?!?!?!
0343
0344 */
0345
0346 // To-do #1: remove Insertionsort as pass 2, it is so stupid/slow - sort the groups inside QuickSort instead.
0347 // To-do #2: make my own cache - something like look-ahead and look-ahead buffers of size 4KB i.e. 4KB ahead of LeftEnd and 4KB ahead of RightEnd, of course constantly updating them/buffers with the moving of LeftPtr and RightPtr. The goal: to reduce/replace freads with a single fread of size 4KB.
0348 // To-do #3: make also 'QuickSortExternal_4+GB.distinct.txt' with removed duplicates with a compression-friendly (highly compressable) number-of-occurrences stats:

```

```

0349 //      were_most_sought_after\t100,000+
0350 //      were_most_sought_after\t010,000+
0351 //      were_most_sought_after\t001,000+
0352 //      were_most_sought_after\t000,100+
0353 //      were_most_sought_after\t000,010+
0354 //      were_most_sought_after\t000,010-
0355
0356 /*
0357 D:\QuickSortExternal_4+GB_r1+>copy con test.txt
0358 aa
0359 aa
0360 aa
0361 aa
0362 bb
0363 aa
0364 dd
0365 zz
0366 aa
0367 zz
0368 qq
0369 make good
0370 made good
0371 make good
0372 ^Z
0373      1 file(s) copied.
0374
0375 D:\QuickSortExternal_4+GB_r1+>"QuickSortExternal_4+GB.exe" test.txt
0376 QuickSortExternal_4+GB r.1++, written by Kaze.
0377 Size of input file: 77
0378 Allocated memory for pointers-to-words in MB: 1
0379 Sorting 14 Pointers ...
0380 Pass #1: Quicksort started ...
0381 / RightEnd-LeftEnd: 000,000,000,013; NumberOfSplittings: 0,000,000,001 ...
0382 Pass #2: Insertionsort started ...
0383 - i: 000,000,000,015 ...
0384 The time to sort 14 items via Quicksort was 0 clocks.
0385 Dumping the sorted data ...
0386 Dumped 14 lines.
0387 Dumped 7 distinct lines.
0388 Done.
0389
0390 D:\QuickSortExternal_4+GB_r1+>type "QuickSortExternal_4+GB.txt"
0391 aa
0392 aa
0393 aa
0394 aa
0395 aa
0396 aa
0397 bb
0398 dd
0399 made good
0400 make good
0401 make good
0402 qq
0403 zz
0404 zz
0405
0406 D:\QuickSortExternal_4+GB_r1+>type "QuickSortExternal_4+GB.distinct.txt"
0407 0,000,006      aa
0408 0,000,001      bb
0409 0,000,001      dd
0410 0,000,001      made good
0411 0,000,002      make good
0412 0,000,001      qq
0413 0,000,002      zz
0414
0415 D:\QuickSortExternal_4+GB_r1+>
0416 */
0417
0418 /*
0419 Linux Fedora 10 64bit log:
0420
0421 [kaze@saturn QuickSortExternal_4+GB_r1+]$ sh ./Linux_QuickSortExternal_4+GB_Compile_Line.script
0422 QuickSortExternal_4+GB.c: In function 'CompareStringsEndingWith13':
0423 QuickSortExternal_4+GB.c:363: warning: passing argument 2 of 'fsetpos' from incompatible pointer type
0424 QuickSortExternal_4+GB.c:376: warning: passing argument 2 of 'fsetpos' from incompatible pointer type
0425 QuickSortExternal_4+GB.c: In function 'main':
0426 QuickSortExternal_4+GB.c:1937: warning: passing argument 2 of 'fsetpos' from incompatible pointer type
0427 [kaze@saturn QuickSortExternal_4+GB_r1+]$ cat ./Linux_QuickSortExternal_4+GB_Compile_Line.script
0428 gcc -D_FILE_OFFSET_BITS=64 -m64 -static -O3 -mtune=generic QuickSortExternal_4+GB.c -o QuickSortExternal_4+GB_generic_64bits.elf
0429 [kaze@saturn QuickSortExternal_4+GB_r1+]$ ls -l *.elf
0430 -rwxrwxr-x 1 kaze kaze 703669 2011-04-29 04:41 QuickSortExternal_4+GB_generic_64bits.elf
0431 [kaze@saturn QuickSortExternal_4+GB_r1+]$ ./QuickSortExternal_4+GB_generic_64bits.elf Poirot_plus_Holmes.txt
0432 QuickSortExternal_4+GB r.1+, written by Kaze.
0433 Size of input file: 84,413,747
0434 Allocated memory for pointers-to-words in MB: 30
0435 Sorting 3,848,740 Pointers ...
0436 Pass #1: Quicksort started ...

```

```

0437 - RightEnd-LeftEnd: 000,000,000,019; NumberOfSplittings: 0,000,439,697 ...
0438 Pass #2: Insertionsort started ...
0439 / i: 000,003,848,741 ...
0440 The time to sort 3,848,740 items via Quicksort was 389,070,000 clocks.
0441 Dumping the sorted data ...
0442 Done.
0443 [kaze@saturn QuickSortExternal_4+GB_r1+]$ ls -l Qu*.txt
0444 -rw-rw-r-- 1 kaze kaze 84413747 2011-04-29 05:25 QuickSortExternal_4+GB.txt
0445 [kaze@saturn QuickSortExternal_4+GB_r1+]$
0446
0447 windows XP 32bit log:
0448
0449 D:\QuickSortExternal_4+GB_r1+>QuickSortExternal_4+GB.exe Poirot_plus_Holmes.txt
0450 QuickSortExternal_4+GB r.1+, written by Kaze.
0451 Size of input file: 84,413,747
0452 Allocated memory for pointers-to-words in MB: 30
0453 Sorting 3,848,740 Pointers ...
0454 Pass #1: Quicksort started ...
0455 - RightEnd-LeftEnd: 000,000,000,019; NumberOfSplittings: 0,000,439,697 ...
0456 Pass #2: Insertionsort started ...
0457 / i: 000,003,848,741 ...
0458 The time to sort 3,848,740 items via Quicksort was 523,531 clocks.
0459 Dumping the sorted data ...
0460 Done.
0461 */
0462
0463 //For everyone interested in slow external sorting ...
0464
0465 //Time results for sorting a file (84,413,747 bytes) with 3,848,740 words of length up to 31 bytes:
0466 //- QuickSortExternal_4+GB_Microsoft.exe needs 743 seconds
0467 //- sort (the windows XP command utility) needs 16 seconds
0468
0469 //what to say: an unexpected far-cry. I had had some big illusions about OS (windows XP) caching performance.
0470 //I am still confused why the speed is so zombie-like: after all there are only I/O read accesses, not a single write?!
0471 //The test machine is Notebook Satellite Merom 2.16GHz, 4GB RAM, windows XP 32 bit, 7200rpm HDD.
0472 //It is interesting a comparison to be made on HDD and SSD on a same machine: I expect 50microseconds vs 10milliseconds seek-time to make the
difference I had expected initially, 200 times faster... hardly but who knows - must be tested.
0473 //As far as I understand the bottlenecks are:
0474 //- seek-time i.e. latency of external memory, not transfer rates;
0475 //- freads by byte accesses - not buffered as I thought [obviously].
0476
0477 //A few words about limitations: revision 1+ compiled as 32bit can sort some 250,000,000 pseudo-pointers(8bytes offsets to 64bit external
pool/file) to the CRLF ending strings in a file regarless of its size (it can exceed 4GB limit) due to maximum ~1950MB malloc with 32bit
windows.
0478 //I am afraid to think how ultra-slow will crawl the code if these pseudo-pointers are put into their own pool/file in order to be fully
independent from system ram allocations.
0479
0480 //To run the test yourself here is the package: QuickSortExternal_4+GB_r1.zip (22,865,439 bytes).
0481
0482 //I ran it with 5,641,065,963 bytes 4-gram wordlist consisted of 248,673,965 lines:
0483 //D:\QuickSortExternal_4+GB_r1>"QuickSortExternal_4+GB_Microsoft.exe" "Corpus_Gamera'_r14_all_quadruplets_A_GRAFFITH.log"
0484 //QuickSortExternal_4+GB r.1, written by Kaze.
0485 //Size of input file: 5,641,065,963
0486 //Allocated memory for pointers-to-words in MB: 1898
0487 //Sorting 248673965 Pointers ...
0488 //...
0489 //78,486,203clocks = 78,486seconds = 21.8hours
0490
0491 //Any ideas how to boost this particular brute-force approach!?
0492 //For any improvement/idea my e-mail: sanmayce@sanmayce.com
0493
0494 // Intel(R) C++ Compiler Professional for applications running on IA-32, Version 11.1:
0495 // QuickSortExternal_4+GB_Intel.exe, Mingroup = 12:
0496 //The current time is: 9:53:57.57
0497 //- MidPtr: 000,001,068,662; NumberOfSplittings: 0,000,439,697 ...
0498 //The time to sort 3848740 items via Quicksort was 1320953 clocks.
0499 //The current time is: 10:16:59.89
0500
0501 // Open Visual C++ Toolkit 2003:
0502 // QuickSortExternal_4+GB_Microsoft.exe, Mingroup = 12:
0503 //The current time is: 10:26:13.34
0504 //- MidPtr: 000,001,068,662; NumberOfSplittings: 0,000,439,697 ...
0505 //The time to sort 3848740 items via Quicksort was 743297 clocks.
0506 //The current time is: 10:38:58.31
0507
0508 // Open Visual C++ Toolkit 2003:
0509 // QuickSort+Insertionsort follows with Mingroup = 16:
0510 //The current time is: 8:22:58.70
0511 //The time to sort 3848740 items via Quicksort was 756297 clocks.
0512 //The current time is: 8:35:55.75
0513
0514 // Open Visual C++ Toolkit 2003:
0515 // QuickSort+Insertionsort follows with Mingroup = 3:
0516 //The current time is: 5:35:08.46
0517 //The time to sort 3848740 items via Quicksort was 838516 clocks.
0518 //The current time is: 5:49:27.45
0519
0520 // Open Visual C++ Toolkit 2003:

```

```

0521 // QuickSort+Insertionsort follows with MinGroup = 64:
0522 //The current time is: 5:54:38.84
0523 //The time to sort 3848740 items via Quicksort was 857656 clocks.
0524 //The current time is: 6:09:17.06
0525
0526 // Open Visual C++ Toolkit 2003:
0527 // QuickSort+Insertionsort follows with Mingroup = 16384/32:
0528 //The current time is: 4:34:04.51
0529 //The time to sort 3848740 items via Quicksort was 2750641 clocks.
0530 //The current time is: 5:20:17.20
0531
0532 /*
0533 D:\QuickSortExternal_4+GB_r1>dir
0534 Volume in drive D is H320_Vol5
0535 Volume Serial Number is 0CB3-C881
0536
0537 Directory of D:\QuickSortExternal_4+GB_r1
0538
0539 04/27/2011 10:25 AM <DIR>      .
0540 04/27/2011 10:25 AM <DIR>      ..
0541 04/27/2011 10:25 AM             0 Empty
0542 04/27/2011 10:25 AM             2 ENTER
0543 04/27/2011 10:25 AM      41,511 LBL2048.zip
0544 04/27/2011 10:25 AM      41,844 Linereporter.zip
0545 04/27/2011 10:25 AM      39,670 Overlapper-Blender_r1+1300MB_BUG-FIXED-WITH-FREE.rar
0546 04/27/2011 10:25 AM       3,396 qsort.c.zip
0547 04/27/2011 10:25 AM      74,248 QuickSortExternal_4+GB.c
0548 04/27/2011 10:25 AM     182,058 QuickSortExternal_4+GB.cod
0549 04/27/2011 10:25 AM       84 QuickSortExternal_4+GB_COMPILE_Intel.bat
0550 04/27/2011 10:25 AM       97 QuickSortExternal_4+GB_COMPILE_Microsoft.bat
0551 04/27/2011 10:25 AM     466,944 QuickSortExternal_4+GB_Intel.exe
0552 04/27/2011 10:25 AM     49,152 QuickSortExternal_4+GB_Microsoft.exe
0553 04/27/2011 10:25 AM       280 RUNME.BAT
0554 04/27/2011 10:25 AM    57,389,250 _Agatha Christie_Texts.txt
0555 04/27/2011 10:25 AM    27,024,497 _Sherlock Holmes_Texts.txt
0556             15 File(s)      85,313,033 bytes
0557             2 Dir(s)  33,349,967,872 bytes free
0558
0559 D:\QuickSortExternal_4+GB_r1>"QuickSortExternal_4+GB_Microsoft.exe"
0560 QuickSortExternal_4+GB r.1, written by Kaze.
0561 Usage: QuickSortExternal_4+GB wordlistfile
0562 Note1: wordlistfile's lines must be up to 31 chars/bytes and end with CRLF i.e. maximum 33bytes long.
0563 Note2: The goal is to sort a file unfittable in physical memory, and with 32bit code.
0564
0565 D:\QuickSortExternal_4+GB_r1>type RUNME.BAT
0566 copy "_Agatha Christie_Texts.txt"+"_Sherlock Holmes_Texts.txt" Poirot_plus_Holmes.txt /b
0567 time<ENTER
0568 QuickSortExternal_4+GB_Microsoft.exe Poirot_plus_Holmes.txt
0569 time<ENTER
0570 sort Poirot_plus_Holmes.txt>SystemSort.txt
0571 time<ENTER
0572 fc QuickSortExternal_4+GB.txt SystemSort.txt /b
0573
0574 D:\QuickSortExternal_4+GB_r1>RUNME.BAT
0575
0576 D:\QuickSortExternal_4+GB_r1>copy "_Agatha Christie_Texts.txt"+"_Sherlock Holmes_Texts.txt" Poirot_plus_Holmes.txt /b
0577 _Agatha Christie_Texts.txt
0578 _Sherlock Holmes_Texts.txt
0579      1 file(s) copied.
0580
0581 D:\QuickSortExternal_4+GB_r1>time0<ENTER
0582 The current time is: 10:26:13.34
0583 Enter the new time:
0584
0585 D:\QuickSortExternal_4+GB_r1>QuickSortExternal_4+GB_Microsoft.exe Poirot_plus_Holmes.txt
0586 QuickSortExternal_4+GB r.1, written by Kaze.
0587 Size of input file: 84413747
0588 Allocated memory for pointers-to-words in MB: 30
0589 Sorting 3848740 Pointers ...
0590 Pass #1: Quicksort started ...
0591 - MidPtr: 000,001,068,662; NumberOfSplittings: 0,000,439,697 ...
0592 Pass #2: Insertionsort started ...
0593 | i: 000,003,833,857 ...
0594 The time to sort 3848740 items via Quicksort was 743297 clocks.
0595 Dumping the sorted data ...
0596 Done.
0597
0598 D:\QuickSortExternal_4+GB_r1>time0<ENTER
0599 The current time is: 10:38:58.31
0600 Enter the new time:
0601
0602 D:\QuickSortExternal_4+GB_r1>sort Poirot_plus_Holmes.txt 1>SystemSort.txt
0603
0604 D:\QuickSortExternal_4+GB_r1>time0<ENTER
0605 The current time is: 10:39:14.56
0606 Enter the new time:
0607
0608 D:\QuickSortExternal_4+GB_r1>fc QuickSortExternal_4+GB.txt SystemSort.txt /b

```

```

0609 Comparing files QuickSortExternal_4+GB.txt and SYSTEMSORT.TXT
0610 FC: no differences encountered
0611 */
0612
0613 // Overlapper-Blender r.1+_SWAP, blend of Compare_Two_Wordlists, revision 1+ and Building-Blocks_DUMPER rev.1, written by Kaze.
0614
0615 //define DumboDumpLOG
0616 #define _WIN32_ENVIRONMENT_
0617 //define _POSIX_ENVIRONMENT_
0618
0619 #if defined(_WIN32_ENVIRONMENT_)
0620 #include <io.h> // needed for windows' 'lseeki64' and 'telli64'
0621 //Above line must be commented in order to compile with Intel C compiler: an error "can't find io.h" occurs.
0622 #else
0623 #endif /* defined(_WIN32_ENVIRONMENT_) */
0624
0625
0626 /*
0627 C:\workTemp\LEPREC~1\VISUAL~1\LEA56D~1>type osho.txt
0628 bxr bxr boban dodo
0629 C:\workTemp\LEPREC~1\VISUAL~1\LEA56D~1>dir osho.txt
0630 18 osho.txt
0631
0632 C:\workTemp\LEPREC~1\VISUAL~1\LEA56D~1>cl /Ox Building-Blocks_DUMPER.c
0633 Microsoft (R) 32-bit C/C++ Optimizing Compiler Version 13.10.3077 for 80x86
0634 Copyright (C) Microsoft Corporation 1984-2002. All rights reserved.
0635
0636 Building-Blocks_DUMPER.c
0637 Microsoft (R) Incremental Linker Version 7.10.3077
0638 Copyright (C) Microsoft Corporation. All rights reserved.
0639
0640 /out:Building-Blocks_DUMPER.exe
0641 Building-Blocks_DUMPER.obj
0642
0643 C:\workTemp\LEPREC~1\VISUAL~1\LEA56D~1>Building-Blocks_DUMPER.exe
0644 Building-Blocks_DUMPER rev.1, written by Kaze.
0645 Sorting 16 Pointers to Building-Blocks 3 chars in size ...
0646 Allocated memory for pointers-to-words in MB: 1
0647 Writing Sorted Building-Blocks to BB003.txt ...
0648
0649 In %c:
0650 C:\workTemp\LEPREC~1\VISUAL~1\LEA56D~1>type BB003.txt
0651 bo
0652 bx
0653 do
0654 an
0655 ban
0656 bob
0657 bxr
0658 bxr
0659 dod
0660 n d
0661 oba
0662 odo
0663 r b
0664 r b
0665 xr
0666 xr
0667
0668 In HEX:
0669 C:\workTemp\LEPREC~1\VISUAL~1\LEA56D~1>type BB003.txt
0670 20626f
0671 206278
0672 20646f
0673 616e20
0674 62616e
0675 626f62
0676 627872
0677 627872
0678 646f64
0679 6e2064
0680 6f6261
0681 6f646f
0682 722062
0683 722062
0684 787220
0685 787220
0686
0687 C:\workTemp\LEPREC~1\VISUAL~1\LEA56D~1>Building-Blocks_DUMPER.exe
0688 Building-Blocks_DUMPER rev.1, written by Kaze.
0689 Sorting 16 Pointers to Building-Blocks 3 chars in size ...
0690 Allocated memory for pointers-to-words in MB: 1
0691 Writing Sorted Building-Blocks to BB003.txt ...
0692 3|18-3+1|13|3
0693
0694 C:\workTemp\LEPREC~1\VISUAL~1\LEA56D~1>type BB003.txt
0695 bo
0696 bx

```

```

0697 do
0698 an
0699 ban
0700 bob
0701 bxr
0702 dod
0703 n d
0704 oba
0705 odo
0706 r b
0707 xr
0708
0709 C:\workTemp\LEPREC~1\VISUAL~1\LEA56D~1>type osho.txt
0710 */
0711
0712
0713 #include <stdio.h>
0714 #include <stdlib.h>
0715 #include <string.h>
0716 #include <time.h>
0717
0718 #ifndef NULL
0719 #define NULL ((void*)0)
0720 #endif
0721
0722 typedef unsigned char char_t;
0723 typedef char_t *string;
0724
0725 int Patternlen;
0726 unsigned long long NumberOfComparisons=0;
0727 FILE *fp_in; // Global - not to burden the extract/compare function with one more parameter
0728 #define LongestLineInclusive 2048 //31 former
0729
0730 //clock_t clocks1, clocks2;
0731 //clock_t clocks3, clocks4;
0732 double TotalRoughSearchTime = 0;
0733
0734 // SINHA fragment[
0735
0736 #define swapKAZE(a, b) { t = *(a); *(a) = *(b); *(b) = t; }
0737
0738 static void InsertSortKAZE(string *a, int n, int d) //void inssort(unsigned char **a, int n, int d)
0739 { string *pi, *pj, s, t; //unsigned char **pi, **pj, *s, *t;
0740   for (pi = a + 1; --n > 0; pi++)
0741     for (pj = pi; pj > a; pj--) {
0742       /* Inline strcmp: break if *(pj-1) <= *pj */
0743       for (s=*(pj-1)+d, t=*pj+d; *s==*t && *s!=0; s++, t++)
0744         ;
0745       if (*s <= *t)
0746         break;
0747       swapKAZE(pj, pj-1);
0748     }
0749 }
0750
0751 //int cmpit(unsigned char **h1, unsigned char **h2)
0752 //{
0753 //  return( strcmp(*h1, *h2) );
0754 //}
0755
0756 int strcmp( unsigned char *s1, unsigned char *s2 )
0757 {
0758   while( *s1 != '\0' && *s1 == *s2 )
0759   {
0760     s1++;
0761     s2++;
0762   }
0763   return( *s1-*s2 );
0764 }
0765
0766 static void simplesort(string a[], int n, int b)
0767 {
0768   int i, j;
0769   string tmp;
0770
0771   for (i = 1; i < n; i++)
0772     for (j = i; j > 0 && strcmp(a[j-1]+b, a[j]+b) > 0; j--)
0773       { tmp = a[j]; a[j] = a[j-1]; a[j-1] = tmp; }
0774 }
0775
0776 int strcmpKAZE13 (
0777   const char * src,
0778   const char * dst
0779 )
0780 {
0781   int ret = 0 ;
0782
0783   while( ! (ret = *(unsigned char *)src - *(unsigned char *)dst) && (*dst!=13))
0784     ++src, ++dst;

```

```

0785
0786     if ( ret < 0 )
0787         ret = -1 ;
0788     else if ( ret > 0 )
0789         ret = 1 ;
0790
0791     return( ret );
0792 }
0793
0794 int CompareStringsEndingwith13(unsigned long long AtPosition64L, unsigned long long AtPosition64R, char *POOLinternal) {
0795
0796 int i;
0797 char FourGramL[LongestLineInclusive+2]; // 31 longest 4-gram + CR + LF
0798 char FourGramR[LongestLineInclusive+2]; // 31 longest 4-gram + CR + LF
0799 unsigned long long *AtPosition64Lpointer=&AtPosition64L;
0800 unsigned long long *AtPosition64Rpointer=&AtPosition64R;
0801
0802 if (POOLinternal == NULL) { // INTERNAL [
0803
0804 // Caramba: seek and tell report OK but in fact they lie, only setpos works?!?!?!
0805
0806 //if defined(_WIN32_ENVIRONMENT_)
0807 //_lseeki64( fileno(fp_in), AtPosition64L, 0 );
0808 //else
0809 //fseeko( fp_in, AtPosition64L, SEEK_SET );
0810 //endif /* defined(_WIN32_ENVIRONMENT_) */
0811
0812 // _CRTIMP __int64 __cdecl _telli64(int);
0813 // off64_t ftello64 (FILE *stream)
0814
0815
0816 fsetpos(fp_in, AtPosition64Lpointer);
0817 #if defined(DumboDumpLOG)
0818 printf("Tell %llu:\n",_telli64(fileno(fp_in)));
0819 printf("L:\n");
0820 #endif
0821 for (i=0; i<(LongestLineInclusive+1); i++) {fread(&FourGramL[i], 1, 1, fp_in); if (FourGramL[i]==13) break;}
0822 //Commented line below is slower than the one above: 778156 clocks vs 756297 clocks.
0823 //fread(&FourGramL[0], 31+1, 1, fp_in);
0824 #if defined(DumboDumpLOG)
0825 for (i=0; i<(LongestLineInclusive+1); i++) {printf("%c",FourGramL[i]); if (FourGramL[i]==13) break;}
0826 printf("\n");
0827 #endif
0828
0829 fsetpos(fp_in, AtPosition64Rpointer);
0830 #if defined(DumboDumpLOG)
0831 printf("Tell %llu:\n",_telli64(fileno(fp_in)));
0832 printf("R:\n");
0833 #endif
0834 for (i=0; i<(LongestLineInclusive+1); i++) {fread(&FourGramR[i], 1, 1, fp_in); if (FourGramR[i]==13) break;}
0835 //Commented line below is slower than the one above: 778156 clocks vs 756297 clocks.
0836 //fread(&FourGramR[0], 31+1, 1, fp_in);
0837 #if defined(DumboDumpLOG)
0838 for (i=0; i<(LongestLineInclusive+1); i++) {printf("%c",FourGramR[i]); if (FourGramR[i]==13) break;}
0839 printf("\n");
0840 #endif
0841
0842 } else { // INTERNAL
0843
0844 for (i=0; i<(LongestLineInclusive+1); i++) {
0845 //fread(&FourGramL[i], 1, 1, fp_in);
0846 FourGramL[i] = *(char *) (POOLinternal + AtPosition64L + i);
0847 if (FourGramL[i]==13) break;
0848 }
0849
0850 for (i=0; i<(LongestLineInclusive+1); i++) {
0851 //fread(&FourGramR[i], 1, 1, fp_in);
0852 FourGramR[i] = *(char *) (POOLinternal + AtPosition64R + i);
0853 if (FourGramR[i]==13) break;
0854 }
0855
0856 } // INTERNAL ]
0857
0858 NumberOfComparisons++;
0859 return(strcmpKAZE13(FourGramL, FourGramR));
0860 }
0861
0862 int CompareStringsEndingwith13_EXTERNAL(unsigned long long AtPosition64L, unsigned long long AtPosition64R) {
0863
0864 int i;
0865 char FourGramL[LongestLineInclusive+2]; // 31 longest 4-gram + CR + LF
0866 char FourGramR[LongestLineInclusive+2]; // 31 longest 4-gram + CR + LF
0867 unsigned long long *AtPosition64Lpointer=&AtPosition64L;
0868 unsigned long long *AtPosition64Rpointer=&AtPosition64R;
0869
0870 // Caramba: seek and tell report OK but in fact they lie, only setpos works?!?!?!
0871
0872 //if defined(_WIN32_ENVIRONMENT_)

```

```

0873 // _lseeki64( fileno(fp_in), AtPosition64L, 0 );
0874 // #else
0875 // fseeko( fp_in, AtPosition64L, SEEK_SET );
0876 // #endif /* defined(_WIN32_ENVIRONMENT_) */
0877
0878 // _CRTIMP __int64 __cdecl _telli64(int);
0879 // off64_t ftello64 (FILE *stream)
0880
0881
0882 fsetpos(fp_in, AtPosition64Lpointer);
0883 #if defined(DumboDumpLOG)
0884 printf("Tell %llu:\n", _telli64(fileno(fp_in)));
0885 printf("L:\n");
0886 #endif
0887 for (i=0; i<(LongestLineInclusive+1); i++) {fread(&FourGramL[i], 1, 1, fp_in); if (FourGramL[i]==13) break;}
0888 //Commented line below is slower than the one above: 778156 clocks vs 756297 clocks.
0889 //fread(&FourGramL[0], 31+1, 1, fp_in);
0890 #if defined(DumboDumpLOG)
0891 for (i=0; i<(LongestLineInclusive+1); i++) {printf("%c", FourGramL[i]); if (FourGramL[i]==13) break;}
0892 printf("\n");
0893 #endif
0894
0895 fsetpos(fp_in, AtPosition64Rpointer);
0896 #if defined(DumboDumpLOG)
0897 printf("Tell %llu:\n", _telli64(fileno(fp_in)));
0898 printf("R:\n");
0899 #endif
0900 for (i=0; i<(LongestLineInclusive+1); i++) {fread(&FourGramR[i], 1, 1, fp_in); if (FourGramR[i]==13) break;}
0901 //Commented line below is slower than the one above: 778156 clocks vs 756297 clocks.
0902 //fread(&FourGramR[0], 31+1, 1, fp_in);
0903 #if defined(DumboDumpLOG)
0904 for (i=0; i<(LongestLineInclusive+1); i++) {printf("%c", FourGramR[i]); if (FourGramR[i]==13) break;}
0905 printf("\n");
0906 #endif
0907
0908 return(strcmpKAZE13(FourGramL, FourGramR));
0909 }
0910
0911 int CompareStringsEndingwith13_INTERNAL(unsigned long long AtPosition64L, unsigned long long AtPosition64R, char *POOLinternal) {
0912
0913 int i;
0914 char FourGramL[LongestLineInclusive+2]; // 31 longest 4-gram + CR + LF
0915 char FourGramR[LongestLineInclusive+2]; // 31 longest 4-gram + CR + LF
0916
0917 for (i=0; i<(LongestLineInclusive+1); i++) {
0918 //fread(&FourGramL[i], 1, 1, fp_in);
0919 FourGramL[i] = *(char *) (POOLinternal + AtPosition64L);
0920 if (FourGramL[i]==13) break;
0921 }
0922
0923 for (i=0; i<(LongestLineInclusive+1); i++) {
0924 //fread(&FourGramR[i], 1, 1, fp_in);
0925 FourGramR[i] = *(char *) (POOLinternal + AtPosition64R);
0926 if (FourGramR[i]==13) break;
0927 }
0928
0929 return(strcmpKAZE13(FourGramL, FourGramR));
0930 }
0931
0932 int memcmpKAZE (
0933     const void * buf1,
0934     const void * buf2,
0935     size_t count
0936 )
0937 {
0938     if (!count)
0939         return(0);
0940
0941     while ( --count && *(char *)buf1 == *(char *)buf2 ) {
0942         buf1 = (char *)buf1 + 1;
0943         buf2 = (char *)buf2 + 1;
0944     }
0945
0946     return( *((unsigned char *)buf1) - *((unsigned char *)buf2) );
0947 }
0948
0949 void * memcpyKAZE (
0950     void * dst,
0951     const void * src,
0952     size_t count
0953 )
0954 {
0955     void * ret = dst;
0956
0957     /*
0958     * copy from lower addresses to higher addresses
0959     */
0960     while (count--) {

```



```

0961         *(char *)dst = *(char *)src;
0962         dst = (char *)dst + 1;
0963         src = (char *)src + 1;
0964     }
0965     return(ret);
0966 }
0967
0968 static void simplesortFIXEDLength(string a[], int n, int b, int FIXEDLength)
0969 {
0970     int i, j;
0971     string tmp;
0972
0973     for (i = 1; i < n; i++)
0974         for (j = i; j > 0 && memcmpKAZE(a[j-1]+b, a[j]+b, FIXEDLength) > 0; j--)
0975             { tmp = a[j]; a[j] = a[j-1]; a[j-1] = tmp; }
0976 }
0977
0978
0979 // SINHA fragment]
0980
0981 // mkqsort.c BEGIN *****
0982 /*
0983  Multikey quicksort, a radix sort algorithm for arrays of character
0984  strings by Bentley and Sedgewick.
0985
0986  J. Bentley and R. Sedgewick. Fast algorithms for sorting and
0987  searching strings. In Proceedings of 8th Annual ACM-SIAM Symposium
0988  on Discrete Algorithms, 1997.
0989
0990  http://www.CS.Princeton.EDU/~rs/strings/index.html
0991
0992  The code presented in this file has been tested with care but is
0993  not guaranteed for any purpose. The writer does not offer any
0994  warranties nor does he accept any liabilities with respect to
0995  the code.
0996
0997  Ranjan sinha, 1 jan 2003.
0998
0999  School of Computer Science and Information Technology,
1000  RMIT University, Melbourne, Australia
1001  rsinha@cs.rmit.edu.au
1002
1003 */
1004
1005 //include "sortstring.h"
1006
1007 /* MULTIKEY QUICKSORT */
1008
1009 #ifndef min
1010 #define min(a, b) ((a)<=(b) ? (a) : (b))
1011 #endif
1012
1013
1014 /* ssort2 -- Faster Version of Multikey Quicksort */
1015
1016 void vecswap2(unsigned char **a, unsigned char **b, int n)
1017 { while (n-- > 0) {
1018     unsigned char *t = *a;
1019     *a++ = *b;
1020     *b++ = t;
1021 } }
1022
1023
1024 #define swap2(a, b) { t = *(a); *(a) = *(b); *(b) = t; }
1025 #define ptr2char(i) (*(i) + depth)
1026
1027 unsigned char **med3func(unsigned char **a, unsigned char **b, unsigned char **c, int depth)
1028 { int va, vb, vc;
1029   if ((va=ptr2char(a)) == (vb=ptr2char(b)))
1030     return a;
1031   if ((vc=ptr2char(c)) == va || vc == vb)
1032     return c;
1033   return va < vb ?
1034     (vb < vc ? b : (va < vc ? c : a)) :
1035     (vb > vc ? b : (va < vc ? a : c));
1036 }
1037 #define med3(a, b, c) med3func(a, b, c, depth)
1038
1039 void insort_NON_Physical(unsigned char **a, int n, int d)
1040 { unsigned char **pi, **pj, *s, *t;
1041   for (pi = a + 1; --n > 0; pi++)
1042     for (pj = pi; pj > a; pj--) {
1043         /* inline strcmp: break if *(pj-1) <= *pj */
1044         for (s=*pj-1+d, t=*pj+d; *s==*t && *s!=0; s++, t++)
1045             ;
1046         if (*s <= *t)
1047             break;
1048         swap2(pj, pj-1);

```

```

1049     }
1050 }
1051
1052
1053 void inssort(unsigned char **a, int n, int d)
1054 { unsigned char **pi, **pj, *s, *t;
1055   for (pi = a + 1; --n > 0; pi++)
1056     for (pj = pi; pj > a; pj--) {
1057       /* Inline strcmp: break if *(pj-1) <= *pj */
1058       for (s=*(pj-1)+d, t=*pj+d; *s==*t && *s!=0; s++, t++)
1059         ;
1060       if (*s <= *t)
1061         break;
1062       swap2(pj, pj-1);
1063     }
1064 }
1065
1066 void mkqsort(unsigned char **a, int n, int depth)
1067 { int d, r, partval;
1068   unsigned char **pa, **pb, **pc, **pd, **pl, **pm, **pn, *t;
1069   if (n < 20) {
1070     inssort(a, n, depth);
1071     //simplsortFIXEDLength(a, n, depth, Patternlen);
1072     return;
1073   }
1074   pl = a;
1075   pm = a + (n/2);
1076   pn = a + (n-1);
1077   if (n > 30) { /* On big arrays, pseudomedian of 9 */
1078     d = (n/8);
1079     pl = med3(pl, pl+d, pl+2*d);
1080     pm = med3(pm-d, pm, pm+d);
1081     pn = med3(pn-2*d, pn-d, pn);
1082   }
1083   pm = med3(pl, pm, pn);
1084   swap2(a, pm);
1085   partval = ptr2char(a);
1086   pa = pb = a + 1;
1087   pc = pd = a + n-1;
1088   for (;;) {
1089     while (pb <= pc && (r = ptr2char(pb)-partval) <= 0) {
1090       if (r == 0) { swap2(pa, pb); pa++; }
1091       pb++;
1092     }
1093     while (pb <= pc && (r = ptr2char(pc)-partval) >= 0) {
1094       if (r == 0) { swap2(pc, pd); pd--; }
1095       pc--;
1096     }
1097     if (pb > pc) break;
1098     swap2(pb, pc);
1099     pb++;
1100     pc--;
1101   }
1102   pn = a + n;
1103   r = min(pa-a, pb-pa); vecswap2(a, pb-r, r);
1104   r = min(pd-pc, pn-pd-1); vecswap2(pb, pn-r, r);
1105   if ((r = pb-pa) > 1)
1106     mkqsort(a, r, depth);
1107   if (ptr2char(a + r) != 0)
1108     mkqsort(a + r, pa-a + pn-pd-1, depth+1);
1109   if ((r = pd-pc) > 1)
1110     mkqsort(a + n-r, r, depth);
1111 }
1112
1113 void mkqsort_main(unsigned char **a, int n) { mkqsort(a, n, 0); }
1114 // mkqsort.c END *****
1115
1116 // why sinha uses int instead of long??!!
1117 static int readlines(char *file_name, string **lines)
1118 {
1119   int nlines = 0;
1120   size_t size;
1121   FILE *in_file;
1122   string basep, cur, next;
1123   string *ASbackup;
1124
1125   if (!(in_file = fopen(file_name, "rb"))) {
1126     printf( "Leprechaun: Can't open file %s \n", file_name );
1127     exit(-1);
1128   }
1129   fseek(in_file, 0, SEEK_END);
1130   size = ftell(in_file);
1131   fseek(in_file, 0, SEEK_SET);
1132   if (!(basep = (string) malloc(size*sizeof(char_t)))) return -1;
1133   printf( "Allocated memory for words in MB: %lu\n", ((size*sizeof(char_t))>>20)+1 );
1134   if (fread(basep, 1, size, in_file) < size) {
1135     printf( "Leprechaun: Can't read file %s \n", file_name );
1136     exit(-1);

```

```

1137 }
1138 fclose(in_file);
1139
1140 // GET nlines:
1141 cur = basep;
1142 while (cur < basep + size) {
1143     next = cur;
1144     while ((next < basep + size) && (*next != '\n')) {next++;}
1145     *--next = '\0'; // This is ala DOS i.e. windows
1146                     // 1310 not 10(\n=10)
1147     cur = next + 2;
1148     nlines++;
1149 }
1150
1151 // printf("%lu\n", (unsigned long)*lines); -> backup = *lines = 0
1152 ASbackup = (string *)malloc( nlines*sizeof(string) ); // sizeof(string) is 4
1153 if( ASbackup == NULL )
1154 { puts( "Leprechaun: Needed memory allocation denied!\n" ); return( 1 ); }
1155 printf( "Allocated memory for pointers-to-words in MB: %lu\n", ((nlines*sizeof(string))>>20)+1 );
1156 *lines = ASbackup;
1157 //printf("%lu\n", (unsigned long)*lines); -> backup = *lines = ASbackup = 6946888
1158
1159 // Upload nlines times:
1160 nlines = 0;
1161 cur = basep;
1162 while (cur < basep + size) {
1163     next = cur;
1164     while ((next < basep + size) && (*next != '\n')) {next++;}
1165     *--next = '\0'; // This is ala DOS i.e. windows
1166                     // 1310 not 10(\n=10)
1167     ASbackup[nlines] = cur;
1168     cur = next + 2;
1169     nlines++;
1170 }
1171 return nlines;
1172 }
1173
1174 void x64toaKAZE ( /* stdcall is faster and smaller... Might as well use it for the helper. */
1175     unsigned long long val,
1176     char *buf,
1177     unsigned radix,
1178     int is_neg
1179 )
1180 {
1181     char *p; /* pointer to traverse string */
1182     char *firstdig; /* pointer to first digit */
1183     char temp; /* temp char */
1184     unsigned digval; /* value of digit */
1185
1186     p = buf;
1187
1188     if ( is_neg )
1189     {
1190         *p++ = '-'; /* negative, so output '-' and negate */
1191         val = (unsigned long long)(-(long long)val);
1192     }
1193
1194     firstdig = p; /* save pointer to first digit */
1195
1196     do {
1197         digval = (unsigned) (val % radix);
1198         val /= radix; /* get next digit */
1199
1200         /* convert to ascii and store */
1201         if (digval > 9)
1202             *p++ = (char) (digval - 10 + 'a'); /* a letter */
1203         else
1204             *p++ = (char) (digval + '0'); /* a digit */
1205     } while (val > 0);
1206
1207     /* We now have the digit of the number in the buffer, but in reverse
1208     order. Thus we reverse them now. */
1209
1210     *p-- = '\0'; /* terminate string; p points to last digit */
1211
1212     do {
1213         temp = *p;
1214         *p = *firstdig;
1215         *firstdig = temp; /* swap *p and *firstdig */
1216         --p;
1217         ++firstdig; /* advance to next two digits */
1218     } while (firstdig < p); /* repeat until halfway */
1219 }
1220
1221 /* Actual functions just call conversion helper with neg flag set correctly,
1222 and return pointer to buffer. */
1223
1224 char * _i64toaKAZE (

```

```

1225     long long val,
1226     char *buf,
1227     int radix
1228 )
1229 {
1230     x64toakAZE((unsigned long long)val, buf, radix, (radix == 10 && val < 0));
1231     return buf;
1232 }
1233
1234 char * _ui64toakAZE (
1235     unsigned long long val,
1236     char *buf,
1237     int radix
1238 )
1239 {
1240     x64toakAZE(val, buf, radix, 0);
1241     return buf;
1242 }
1243
1244 char * _ui64toakAZEzerocomma (
1245     unsigned long long val,
1246     char *buf,
1247     int radix
1248 )
1249 {
1250     char *p;
1251     char temp;
1252     int txpman;
1253     int pxnman;
1254     x64toakAZE(val, buf, radix, 0);
1255     p = buf;
1256     do {
1257     } while (*++p != '\0');
1258     p--; // p points to last digit
1259     // buf points to first digit
1260     buf[26] = 0;
1261     txpman = 1;
1262     pxnman = 0;
1263     do
1264     { if (buf <= p)
1265       { temp = *p;
1266         buf[26-txpman] = temp; pxnman++;
1267         p--;
1268         if (pxnman % 3 == 0)
1269         { txpman++;
1270           buf[26-txpman] = (char) (' ');
1271         }
1272       }
1273     else
1274     { buf[26-txpman] = (char) ('0'); pxnman++;
1275       if (pxnman % 3 == 0)
1276       { txpman++;
1277         buf[26-txpman] = (char) (' ');
1278       }
1279     }
1280     txpman++;
1281     } while (txpman <= 26);
1282     return buf;
1283 }
1284
1285 char * _ui64toakAZEcomma (
1286     unsigned long long val,
1287     char *buf,
1288     int radix
1289 )
1290 {
1291     char *p;
1292     char temp;
1293     int txpman;
1294     int pxnman;
1295     x64toakAZE(val, buf, radix, 0);
1296     p = buf;
1297     do {
1298     } while (*++p != '\0');
1299     p--; // p points to last digit
1300     // buf points to first digit
1301     buf[26] = 0;
1302     txpman = 1;
1303     pxnman = 0;
1304     while (buf <= p)
1305     { temp = *p;
1306       buf[26-txpman] = temp; pxnman++;
1307       p--;
1308       if (pxnman % 3 == 0 && buf <= p)
1309       { txpman++;
1310         buf[26-txpman] = (char) (' ');
1311       }
1312     }
1313     txpman++;

```

```

1313     }
1314     return buf+26-(txpman-1);
1315 }
1316
1317 long Railgunhits=0;
1318
1319 #define ASIZE 256
1320
1321 // ### Boyer-Moore-Horspool algorithm [
1322 long HORSPOOL(y, x, n, m)
1323     char *y, *x;
1324     long n;
1325     int m;
1326     {
1327     long i;
1328     int a, j, bm_bc[ASIZE];
1329     unsigned char ch, lastch;
1330
1331     /* Preprocessing */
1332     for (a=0; a < ASIZE; a++) bm_bc[a]=m;
1333     for (j=0; j < m-1; j++) bm_bc[x[j]]=m-j-1;
1334
1335     /* Searching */
1336     lastch=x[m-1];
1337     i=0;
1338     while (i <= n-m) {
1339         ch=y[i+m-1];
1340         if (ch == lastch)
1341             //if (memcmp(&y[i],x,m-1) == 0) OUTPUT(i);
1342             if (memcmp(&y[i],x,m-1) == 0) return(i);
1343             i+=bm_bc[ch];
1344         }
1345     return(-1);
1346 }
1347 long Boyer_Moore_Horspool_Kaze(y, x, n, m)
1348     char *y, *x;
1349     long n;
1350     int m;
1351     {
1352     long i;
1353     int a, j, bm_bc[ASIZE];
1354     unsigned char ch;
1355     //unsigned char ch, lastch;
1356     //unsigned char firstch;
1357
1358     /* Preprocessing */
1359     for (a=0; a < ASIZE; a++) bm_bc[a]=m;
1360     for (j=0; j < m-1; j++) bm_bc[x[j]]=m-j-1;
1361
1362     /* Searching */
1363     //lastch=x[m-1];
1364     //firstch=x[0];
1365     i=0;
1366     while (i <= n-m) {
1367         ch=y[i+m-1];
1368         //if (ch == lastch)
1369         //if (memcmp(&y[i],x,m-1) == 0) OUTPUT(i);
1370 // Below line gives: 315KB/clock
1371 //if (ch == lastch && y[i] == firstch && memcmp(&y[i],x,m-1) == 0) return(i); // Kaze: The idea(to prevent execution of slower
'memcmp') is borrowed from Karp-Rabin i.e. to perform a slower check only when the target "looks like".
1372 // Below line gives: 328KB/clock
1373 if (ch == x[m-1] && y[i] == x[0] && memcmp(&y[i],x,m-1) == 0) return(i); // Kaze: The idea(to prevent execution of slower 'memcmp')
is borrowed from Karp-Rabin i.e. to perform a slower check only when the target "looks like".
1374 i+=bm_bc[ch];
1375 }
1376 return(-1);
1377 }
1378 // ### Boyer-Moore-Horspool algorithm ]
1379
1380
1381 // ### Brute force 'Dummy' algorithm [
1382 long Brute_Force_Dummy(char *y, char *x, long n, int m) {
1383     long i, j;
1384
1385     /* Searching */
1386     for (i=0; i <= n-m; i++) {
1387         j=0;
1388         while (j < m && y[i+j] == x[j]) j++;
1389         if (j >= m) return(i);
1390     }
1391     return(-1);
1392 }
1393 // ### Brute force 'Dummy' algorithm ]
1394
1395
1396 // ### Karp-Rabin algorithm [
1397 #define REHASH(a, b, h) (((h) - (a)*d) << 1) + (b))
1398 long Karp_Rabin(char *y, char *x, long n, int m) {

```

```

1399     int d, hx, hy, i, j;
1400
1401     /* Preprocessing */
1402     /* computes d = 2^(m-1) with
1403        the left-shift operator */
1404     for (d = i = 1; i < m; ++i)
1405         d = (d<<1);
1406
1407     for (hy = hx = i = 0; i < m; ++i) {
1408         hx = ((hx<<1) + x[i]);
1409         hy = ((hy<<1) + y[i]);
1410     }
1411
1412     /* Searching */
1413     j = 0;
1414     while (j <= n-m) {
1415         if (hx == hy && memcmp(x, y + j, m) == 0) return(j);
1416         hy = REHASH(y[j], y[j + m], hy);
1417         ++j;
1418     }
1419     return(-1);
1420 }
1421 // ### Karp-Rabin algorithm ]
1422
1423
1424 // ### Karp-Rabin-Kaze algorithm [
1425 char * KarpRabinKaze (char * pbTarget,
1426     char * pbPattern,
1427     unsigned long cbTarget,
1428     unsigned long cbPattern)
1429 {
1430     unsigned int i;
1431     char * pbTargetMax = pbTarget + cbTarget;
1432     char * pbPatternMax = pbPattern + cbPattern;
1433     unsigned long ulBaseToPowerMod = 1;
1434     register unsigned long ulHashPattern = 0;
1435     unsigned long ulHashTarget = 0;
1436     long hits = 0;
1437     //unsigned long count;
1438     //char * buf1;
1439     //char * buf2;
1440
1441     if (cbPattern > cbTarget)
1442         return(NULL);
1443
1444     // Compute the power of the left most character in base ulBase
1445     //for (i = 1; i < cbPattern; i++) ulBaseToPowerMod = (ulBase * ulBaseToPowerMod);
1446
1447     // Calculate the hash function for the src (and the first dst)
1448     while (pbPattern < pbPatternMax)
1449     {
1450         // Below lines give 366KB/clock for 'underdog':
1451         //ulHashPattern = (ulHashPattern*ulBase + *pbPattern);
1452         //ulHashTarget = (ulHashTarget*ulBase + *pbTarget);
1453         pbPattern++;
1454         pbTarget++;
1455     }
1456     // Below lines give 436KB/clock for 'underdog' + requirement pattern to be 4 chars min.:
1457     //ulHashPattern = ( (* (long *) (pbPattern-cbPattern)) & 0xffffffff ) + *(pbPattern-1);
1458     //ulHashTarget = ( (* (long *) (pbTarget-cbPattern)) & 0xffffffff ) + *(pbTarget-1);
1459     // Below lines give 482KB/clock for 'underdog' + requirement pattern to be 2 chars min.:
1460     //ulHashPattern = ( (* (unsigned short *) (pbPattern-cbPattern)) | *(pbPattern-1) );
1461     //ulHashTarget = ( (* (unsigned short *) (pbTarget-cbPattern)) | *(pbTarget-1) );
1462     // Below lines give 482KB/clock for 'underdog' + requirement pattern to be 2 chars min.:
1463     //ulHashPattern = ( (* (unsigned short *) (pbPattern-cbPattern)) & 0xff00 ) + *(pbPattern-1);
1464     //ulHashTarget = ( (* (unsigned short *) (pbTarget-cbPattern)) & 0xff00 ) + *(pbTarget-1);
1465     // Below lines give 605KB/clock for 'underdog' + requirement pattern to be 2 chars min.:
1466     //ulHashPattern = ( (* (unsigned short *) (pbPattern-cbPattern))<<8 ) + *(pbPattern-1);
1467     //ulHashTarget = ( (* (unsigned short *) (pbTarget-cbPattern))<<8 ) + *(pbTarget-1);
1468     // Below lines give 668KB/clock for 'underdog':
1469     ulHashPattern = ( (* (char *) (pbPattern-cbPattern))<<8 ) + *(pbPattern-1);
1470     ulHashTarget = ( (* (char *) (pbTarget-cbPattern))<<8 ) + *(pbTarget-1);
1471
1472     // Dynamically produce hash values for the string as we go
1473     for ( ;; )
1474     {
1475         if ( (ulHashPattern == ulHashTarget) && !memcmp(pbPattern-cbPattern, pbTarget-cbPattern, (unsigned int)cbPattern) )
1476             // if ( ulHashPattern == ulHashTarget ) {
1477             //
1478             // count = cbPattern;
1479             // buf1 = pbPattern-cbPattern;
1480             // buf2 = pbTarget-cbPattern;
1481             // while ( --count && *(char *)buf1 == *(char *)buf2 ) {
1482             //     buf1 = (char *)buf1 + 1;
1483             //     buf2 = (char *)buf2 + 1;
1484             // }
1485             //
1486             // if ( *((unsigned char *)buf1) - *((unsigned char *)buf2) == 0) hits++;

```

```

1487 // }
1488     return((pbTarget-cbPattern));
1489     //hits++;
1490
1491     if (pbTarget == pbTargetMax)
1492         return(NULL);
1493
1494     // Below line gives 482KB/clock for 'underdog' + requirement pattern to be 2 chars min.:
1495     //ulHashTarget = ( (*unsigned short *) (pbTarget+1-cbPattern)) | *pbTarget );
1496     // Below line gives 436KB/clock for 'underdog' + requirement pattern to be 4 chars min.:
1497     //ulHashTarget = ( (*long *) (pbTarget+1-cbPattern)) & 0xffffffff00 ) + *pbTarget;
1498 //; Line 696
1499 //     movsx     esi, BYTE PTR [ebx]
1500 //     mov      ecx, DWORD PTR [edx+1]
1501 //     and      ecx, -256                                ; ffffffff00H
1502 //     add      ecx, esi
1503 //     // Below line gives 482KB/clock for 'underdog' + requirement pattern to be 2 chars min.:
1504 //     //ulHashTarget = ( (*unsigned short *) (pbTarget+1-cbPattern)) & 0xff00 ) + *pbTarget;
1505 //; Line 691
1506 //     movsx     esi, BYTE PTR [ebx]
1507 //     xor      ecx, ecx
1508 //     mov      cx, WORD PTR [edx+1]
1509 //     and      ecx, 65280                                ; 0000ff00H
1510 //     add      ecx, esi
1511 //     // Below line gives 605KB/clock for 'underdog' + requirement pattern to be 2 chars min.:
1512 //     //ulHashTarget = ( (*unsigned short *) (pbTarget+1-cbPattern))<<8 ) + *pbTarget;
1513 //     // Below line gives 668KB/clock for 'underdog':
1514 //     ulHashTarget = ( (*char *) (pbTarget+1-cbPattern))<<8 ) + *pbTarget;
1515 //; Line 718
1516 //     movsx     ecx, BYTE PTR [eax+1]
1517 //     movsx     edx, BYTE PTR [ebp]
1518 //     shl      ecx, 8
1519 //     add      ecx, edx
1520 //     // Below line gives 366KB/clock for 'underdog':
1521 //     //ulHashTarget = (ulHashTarget - *(pbTarget-cbPattern)*ulBaseToPowerMod)*ulBase + *pbTarget;
1522 //     pbTarget++;
1523 }
1524 }
1525 // ### Karp-Rabin-Kaze algorithm ]
1526
1527
1528 // ### Mix(2in1) of Karp-Rabin & Boyer-Moore-Horspool algorithm [
1529 // Caution: For better speed the case 'if (cbPattern==1)' was removed, so Pattern must be longer than 1 char.
1530 char * Railgun (char * pbTarget,
1531                char * pbPattern,
1532                unsigned long cbTarget,
1533                unsigned long cbPattern)
1534 {
1535     char * pbTargetMax = pbTarget + cbTarget;
1536     register unsigned long ulHashPattern;
1537     unsigned long ulHashTarget;
1538     unsigned long count;
1539     unsigned long countSTATIC, countRemainder;
1540
1541     long i; //BMH needed
1542     int a, j, bm_bc[ASIZE]; //BMH needed
1543     unsigned char ch; //BMH needed
1544 //     unsigned char lastch, firstch; //BMH needed
1545
1546     if (cbPattern > cbTarget)
1547         return(NULL);
1548
1549     countSTATIC = cbPattern-2;
1550
1551 // Doesn't work when cbPattern = 1
1552 if (cbTarget<961) // This value is arbitrary(don't know how exactly), it ensures(at least must) better performance than
1553 // 'Boyer_Moore_Horspool'.
1554 {
1555     pbTarget = pbTarget+cbPattern;
1556     ulHashPattern = ( (*char *) (pbPattern))<<8 ) + *(pbPattern+(cbPattern-1));
1557     for ( ;; )
1558     {
1559         // The line below gives for 'cbPattern'>=1:
1560         // Karp_Rabin_Kaze_4_OCTETS_hits/Karp_Rabin_Kaze_4_OCTETS_clocks: 4/543
1561         // Karp_Rabin_Kaze_4_OCTETS performance: 372KB/clock
1562 /*
1563         if ( (ulHashPattern == ( (*char *) (pbTarget-cbPattern))<<8 ) + *(pbTarget-1)) && !memcmp(pbPattern, pbTarget-cbPattern, (unsigned
1564 int)cbPattern) )
1565             return((long)(pbTarget-cbPattern));
1566 */
1567         // The fragment below gives for 'cbPattern'>=2:
1568         // Karp_Rabin_Kaze_4_OCTETS_hits/Karp_Rabin_Kaze_4_OCTETS_clocks: 4/546
1569         // Karp_Rabin_Kaze_4_OCTETS performance: 370KB/clock
1570
1571         if ( ulHashPattern == ( (*char *) (pbTarget-cbPattern))<<8 ) + *(pbTarget-1) ) {
1572             count = countSTATIC;

```

```

1573     while ( count && *(char *) (pbPattern+1+(countSTATIC-count)) == *(char *) (pbTarget-cbPattern+1+(countSTATIC-count)) ) {
1574         count--;
1575     }
1576     if ( count == 0 ) return((pbTarget-cbPattern));
1577 }
1578
1579 // The fragment below gives for 'cbPattern'>=2:
1580 // Karp_Rabin_Kaze_4_OCTETS_hits/Karp_Rabin_Kaze_4_OCTETS_clocks: 4/554
1581 // Karp_Rabin_Kaze_4_OCTETS performance: 364KB/clock
1582 /*
1583     if ( ulHashPattern == ( (*(char *) (pbTarget-cbPattern))<<8 ) + *(pbTarget-1) ) {
1584         count = countSTATIC>>2;
1585         countRemainder = countSTATIC % 4;
1586
1587         while ( count && *(unsigned long *) (pbPattern+1+((count-1)<<2)) == *(unsigned long *) (pbTarget-cbPattern+1+((count-1)<<2)) ) {
1588             count--;
1589         }
1590 //if (count == 0) { // Disastrous degradation only from this line(317KB/clock when 1+2x4+2+1 bytes pattern: 'skilllessness'; 312KB/clock when
1+1x4+2+1 bytes pattern: 'underdog'), otherwise 368KB/clock.
1591     while ( countRemainder && *(char *) (pbPattern+1+(countSTATIC-countRemainder)) == *(char *) (pbTarget-cbPattern+1+(countSTATIC-
countRemainder)) ) {
1592         countRemainder--;
1593     }
1594     //if ( countRemainder == 0 ) return((long) (pbTarget-cbPattern));
1595     if ( count+countRemainder == 0 ) return((long) (pbTarget-cbPattern));
1596 }
1597 }
1598 */
1599
1600     pbTarget++;
1601     if (pbTarget > pbTargetMax)
1602         return(NULL);
1603 }
1604 }
1605 else
1606 {
1607     /* Preprocessing */
1608     for (a=0; a < ASIZE; a++) bm_bc[a]=cbPattern;
1609     for (j=0; j < cbPattern-1; j++) bm_bc[pbPattern[j]]=cbPattern-j-1;
1610
1611     /* Searching */
1612     //lastch=pbPattern[cbPattern-1];
1613     //firstch=pbPattern[0];
1614     i=0;
1615     while (i <= cbTarget-cbPattern) {
1616         ch=pbTarget[i+cbPattern-1];
1617         //if (ch==lastch)
1618             //if (memcmp(&pbTarget[i],pbPattern,cbPattern-1) == 0) OUTPUT(i);
1619             //if (ch == lastch && pbTarget[i] == firstch && memcmp(&pbTarget[i],pbPattern,cbPattern-1) == 0) return(i); // Kaze: The idea(to
prevent execution of slower 'memcmp') is borrowed from Karp-Rabin i.e. to perform a slower check only when the target "looks like".
1620             if (ch == pbPattern[cbPattern-1] && pbTarget[i] == pbPattern[0])
1621             {
1622                 count = countSTATIC;
1623                 while ( count && *(char *) (pbPattern+1+(countSTATIC-count)) == *(char *) (&pbTarget[i]+1+(countSTATIC-count)) ) {
1624                     count--;
1625                 }
1626                 if ( count == 0 ) return(pbTarget+i);
1627             }
1628             i+=bm_bc[ch];
1629     }
1630     return(NULL);
1631 }
1632 }
1633 // ### Mix(2in1) of Karp-Rabin & Boyer-Moore-Horspool algorithm ]
1634
1635
1636 // ### Railgun_totalhits [
1637 char * Railgun_totalhits (char * pbTarget,
1638     char * pbPattern,
1639     unsigned long cbTarget,
1640     unsigned long cbPattern)
1641 {
1642     char * pbTargetMax = pbTarget + cbTarget;
1643     register unsigned long ulHashPattern;
1644     unsigned long ulHashTarget;
1645     unsigned long count;
1646     unsigned long countSTATIC, countRemainder;
1647
1648     long i; //BMH needed
1649     int a, j, bm_bc[ASIZE]; //BMH needed
1650     unsigned char ch; //BMH needed
1651     // unsigned char lastch, firstch; //BMH needed
1652
1653     if (cbPattern > cbTarget)
1654         return(NULL);
1655
1656     countSTATIC = cbPattern-2;
1657

```



```

1658 // Doesn't work when cbPattern = 1
1659 if (cbTarget<961) // This value is arbitrary(don't know how exactly), it ensures(at least must) better performance than
    'Boyer_Moore_Horspool'.
1660 {
1661     pbTarget = pbTarget+cbPattern;
1662     ulHashPattern = ( (*char *) (pbPattern))<<8 ) + *(pbPattern+(cbPattern-1));
1663
1664     for ( ;; )
1665     {
1666         // The line below gives for 'cbPattern'>=1:
1667         // Karp_Rabin_Kaze_4_OCTETS_hits/Karp_Rabin_Kaze_4_OCTETS_clocks: 4/543
1668         // Karp_Rabin_Kaze_4_OCTETS performance: 372KB/clock
1669         /*
1670         if ( (ulHashPattern == ( (*char *) (pbTarget-cbPattern))<<8 ) + *(pbTarget-1) ) && !memcmp(pbPattern, pbTarget-cbPattern, (unsigned
1671         int)cbPattern) )
1672             return((long)(pbTarget-cbPattern));
1673         */
1674
1675         // The fragment below gives for 'cbPattern'>=2:
1676         // Karp_Rabin_Kaze_4_OCTETS_hits/Karp_Rabin_Kaze_4_OCTETS_clocks: 4/546
1677         // Karp_Rabin_Kaze_4_OCTETS performance: 370KB/clock
1678         if ( ulHashPattern == ( (*char *) (pbTarget-cbPattern))<<8 ) + *(pbTarget-1) ) {
1679             count = countSTATIC;
1680             while ( count && *(char *) (pbPattern+1+(countSTATIC-count)) == *(char *) (pbTarget-cbPattern+1+(countSTATIC-count)) ) {
1681                 count--;
1682             }
1683             if ( count == 0 ) Railgunhits++; //return((pbTarget-cbPattern));
1684         }
1685
1686         // The fragment below gives for 'cbPattern'>=2:
1687         // Karp_Rabin_Kaze_4_OCTETS_hits/Karp_Rabin_Kaze_4_OCTETS_clocks: 4/554
1688         // Karp_Rabin_Kaze_4_OCTETS performance: 364KB/clock
1689         /*
1690         if ( ulHashPattern == ( (*char *) (pbTarget-cbPattern))<<8 ) + *(pbTarget-1) ) {
1691             count = countSTATIC>>2;
1692             countRemainder = countSTATIC % 4;
1693             while ( count && *(unsigned long *) (pbPattern+1+((count-1)<<2)) == *(unsigned long *) (pbTarget-cbPattern+1+((count-1)<<2)) ) {
1694                 count--;
1695             }
1696             //if (count == 0) { // Disastrous degradation only from this line(317KB/clock when 1+2x4+2+1 bytes pattern: 'skilllessness'; 312KB/clock when
1697             //1+1x4+2+1 bytes pattern: 'underdog'), otherwise 368KB/clock.
1698             while ( countRemainder && *(char *) (pbPattern+1+(countSTATIC-countRemainder)) == *(char *) (pbTarget-cbPattern+1+(countSTATIC-
1699             countRemainder)) ) {
1700                 countRemainder--;
1701             }
1702             //if ( countRemainder == 0 ) return((long)(pbTarget-cbPattern));
1703             if ( count+countRemainder == 0 ) return((long)(pbTarget-cbPattern));
1704             //}
1705         */
1706
1707         pbTarget++;
1708         if (pbTarget > pbTargetMax)
1709             return(NULL);
1710     }
1711 }
1712 else
1713 {
1714     /* Preprocessing */
1715     for (a=0; a < ASIZE; a++) bm_bc[a]=cbPattern;
1716     for (j=0; j < cbPattern-1; j++) bm_bc[pbPattern[j]]=cbPattern-j-1;
1717
1718     /* Searching */
1719     //lastch=pbPattern[cbPattern-1];
1720     //firstch=pbPattern[0];
1721     i=0;
1722     while (i <= cbTarget-cbPattern) {
1723         ch=pbTarget[i+cbPattern-1];
1724         //if (ch == lastch)
1725             //if (memcmp(&pbTarget[i],pbPattern,cbPattern-1) == 0) OUTPUT(i);
1726             //if (ch == lastch && pbTarget[i] == firstch && memcmp(&pbTarget[i],pbPattern,cbPattern-1) == 0) return(i); // Kaze: The idea(to
1727             //prevent execution of slower 'memcmp') is borrowed from Karp-Rabin i.e. to perform a slower check only when the target "looks like".
1728             if (ch == pbPattern[cbPattern-1] && pbTarget[i] == pbPattern[0])
1729             {
1730                 count = countSTATIC;
1731                 while ( count && *(char *) (pbPattern+1+(countSTATIC-count)) == *(char *) (&pbTarget[i]+1+(countSTATIC-count)) ) {
1732                     count--;
1733                 }
1734                 if ( count == 0 ) Railgunhits++; //return(pbTarget+i);
1735             }
1736             i+=bm_bc[ch];
1737     }
1738     return(NULL);
1739 }
1740 // ### Railgun_totalhits ]

```

```

1741
1742
1743 // ### Karp-Rabin-Kaze_BOOSTED algorithm [
1744 char * KarpRabinKaze_BOOSTED (char * pbTarget,
1745     char * pbPattern,
1746     unsigned long cbTarget,
1747     unsigned long cbPattern)
1748 {
1749     char * pbTargetMax = pbTarget + cbTarget;
1750     register unsigned long ulHashPattern;
1751     unsigned long ulHashTarget;
1752
1753     if (cbPattern > cbTarget)
1754         return(NULL);
1755
1756     pbTarget = pbTarget+cbPattern;
1757     ulHashPattern = ( (*(char *) (pbPattern))<<8 ) + *(pbPattern+(cbPattern-1));
1758
1759     for ( ;; )
1760     {
1761         // Kaze: The idea(FAILED) here is to add an additional(second) layer in order to prevent execution of slower hash calculation(i.e.
first layer) which(hash) prevents execution of even slower 'memcmp'.
1762         // The line below gives: 314KB/clock
1763         //if ( *pbPattern == *(char *) (pbTarget-cbPattern) && ulHashPattern == ( (*(char *) (pbTarget-cbPattern))<<8 ) + *(pbTarget-1) ) &&
!memcmp(pbPattern, pbTarget-cbPattern, (unsigned int)cbPattern) )
1764         // The line below gives: 370KB/clock
1765         if ( (ulHashPattern == ( (*(char *) (pbTarget-cbPattern))<<8 ) + *(pbTarget-1) ) && !memcmp(pbPattern, pbTarget-cbPattern, (unsigned
int)cbPattern) )
1766             return((pbTarget-cbPattern));
1767
1768         pbTarget++;
1769         if (pbTarget > pbTargetMax)
1770             return(NULL);
1771     }
1772 }
1773 // ### Karp-Rabin-Kaze_BOOSTED algorithm ]
1774
1775 char * strstr_Microsoft (
1776     const char * str1,
1777     const char * str2
1778 )
1779 {
1780     char *cp = (char *) str1;
1781     char *s1, *s2;
1782
1783     if ( !*str2 )
1784         return((char *) str1);
1785
1786     while (*cp)
1787     {
1788         s1 = cp;
1789         s2 = (char *) str2;
1790
1791         while ( *s1 && *s2 && !(*s1-*s2) )
1792             s1++, s2++;
1793
1794         if (!*s2)
1795             return(cp);
1796
1797         cp++;
1798     }
1799     return(NULL);
1800 }
1801
1802 char *
1803 strstr_GNU_C_Library (phystack, pneedle)
1804     const char *phystack;
1805     const char *pneedle;
1806 {
1807     const unsigned char *haystack, *needle;
1808     char b;
1809     const unsigned char *rneedle;
1810
1811     haystack = (const unsigned char *) phystack;
1812
1813     if ((b = *(needle = (const unsigned char *) pneedle)))
1814     {
1815         char c;
1816         haystack--; /* possible ANSI violation */
1817
1818         {
1819             char a;
1820             do
1821                 if (!(a = *++haystack))
1822                     goto ret0;
1823             while (a != b);
1824         }
1825

```

```

1826     if (!(c = *++needle))
1827 goto foundneedle;
1828     ++needle;
1829     goto jin;
1830
1831     for (;;)
1832 {
1833 {
1834     char a;
1835     if (0)
1836     jin:{
1837         if ((a = *++haystack) == c)
1838             goto crest;
1839     }
1840     else
1841         a = *++haystack;
1842     do
1843     {
1844         for (; a != b; a = *++haystack)
1845         {
1846             if (!a)
1847                 goto ret0;
1848             if ((a = *++haystack) == b)
1849                 break;
1850             if (!a)
1851                 goto ret0;
1852         }
1853     }
1854     while ((a = *++haystack) != c);
1855 }
1856 crest:
1857 {
1858     char a;
1859     {
1860         const unsigned char *rhaystack;
1861         if (*(rhaystack = haystack-- + 1) == (a = *(rneedle = needle)))
1862         do
1863         {
1864             if (!a)
1865                 goto foundneedle;
1866             if (*++rhaystack != (a = *++needle))
1867                 break;
1868             if (!a)
1869                 goto foundneedle;
1870         }
1871         while (*++rhaystack == (a = *++needle));
1872         needle = rneedle; /* took the register-poor approach */
1873     }
1874     if (!a)
1875         break;
1876 }
1877 }
1878 }
1879 foundneedle:
1880 return (char *) haystack;
1881 ret0:
1882 return 0;
1883 }
1884
1885
1886 int main(argc, argv)
1887 int argc; char *argv[];
1888 {
1889 FILE *fp_inLINE;
1890 //FILE *fp_in;
1891 FILE *fp_in2;
1892 FILE *fp_out;
1893 FILE *fp_out2;
1894 FILE *fp_out3;
1895 FILE *fp_outRG;
1896 int Bozan;
1897 long ThunderwithL, ThunderwithR;
1898 long ThunderwithL2, ThunderwithR2;
1899 char *Strng;
1900 char *Strng2;
1901 char *StrngDUMMY;
1902 long Strnglen;
1903 long Strnglen2;
1904 long Hits;
1905 int HitsFound=0, HitsUnfamiliar=0;
1906 long OVERLAPPEDlines=0;
1907 long Unfamiliarlines=0;
1908 long nlines_A=0;
1909 long nlines_B=0;
1910
1911 long StrnglenTRAVERSED;
1912 char Pattern[20+2000]; // skilllessness=12 human consciousness=19 I should have known=19
1913 // In the East, enlightenment is described as a state of ultimate=62

```

```

1914 //int Patternlen; // Make it global in sake of QSORT
1915 long LinesEncountered=0;
1916 long BruteForceDummyhits=0;
1917 long KarpRabinKazehits=0;
1918 long KarpRabinKaze_BOOSTEDhits=0;
1919 long Karp_Rabin_Kaze_4_OCTETShits=0;
1920 long KarpRabinhits=0;
1921 long HORSPoolhits=0;
1922 long HORSPool_Kazehits=0;
1923 long strstrMicrosofthits=0;
1924 long strstrGNUCLibraryhits=0;
1925
1926 //int i,j;
1927 char *DumboBox[8][2] = { "an\0", "to\0",
1928 "TDK\0", "the\0",
1929 "fast\0", "easy\0",
1930 "grmb\0", "email\0",
1931 "pasting\0", "amazing\0",
1932 "underdog\0", "superdog\0",
1933 "participants\0", "skilllessness\0",
1934 "I should have known\0", "human consciousness\0"
1935 };
1936
1937 long FoundIn;
1938 char *FoundInPTR;
1939 //char l1ToaDigits[27]; // 9,223,372,036,854,775,807: 1(sign or carry)+19(digits)+1('\0')+6(,)
1940 char BuildingBlocksBox[10] = "BB---.txt\0";
1941
1942
1943 //SWAP [
1944
1945 unsigned long long ThunderwithL64;
1946 unsigned long long Strnglen64;
1947 unsigned long long size_in64, size_in2;
1948 unsigned long long Over4billionLines, j_Over4billion;
1949 char OneChar_ieByte;
1950 unsigned long long SeekPosition;
1951 unsigned long long *PointerToSeekPosition;
1952
1953 char FourGram[LongestLineInclusive+2]; // 31 longest 4-gram + CR + LF
1954 char *Auberge[4] = {"|\0", "\0", "-\0", "\\0"};
1955 int Melnitchka;
1956 char l1ToaDigits[27]; // 9,223,372,036,854,775,807: 1(sign or carry)+19(digits)+1('\0')+6(,)
1957
1958 char *PoolPhysical;
1959 char *ARG_CLf = "/fast\0";
1960 char *ARG_CLs = "/slow\0";
1961 unsigned long long fsetpos_ZERO=0;
1962
1963 //QuickSort:
1964 unsigned long long Nbr2Sort;
1965 unsigned long long QSleft[512]; // Stack arrays for the left/right
1966 unsigned long long QSright[512]; // ends of subgroups within QuickSort()
1967 unsigned long long i, j, k, StackPtr;
1968 unsigned long long LeftEnd, RightEnd, LeftPtr, RightPtr, MidPtr, MinGroup;
1969 unsigned long long Pvalue, temp;
1970 long clocks1, clocks2;
1971 long clocks0, clocks3;
1972 unsigned long long NumberOfSplittings=0;
1973
1974 //RG:
1975 unsigned long long jFirst, jPrevLine, j_RG;
1976 unsigned long long linecounterNotRL, linecounterRL, nlines1;
1977 unsigned long long TimesOverItem;
1978
1979 //SWAP ]
1980
1981 int nlines = 0;
1982 string basep, cur, next;
1983 string *ASbackup;
1984 string *ASbackup2;
1985
1986 //unsigned long nlines1, linecounterNotRL, linecounterRL, jPrevLine, jFirst;
1987
1988 printf("QuickSortExternal_4+GB r.2+, written by Kaze.\n");
1989 if (argc != 3) {
1990 printf("Usage: QuickSortExternal_4+GB wordlistfile {/fast|/slow}\n");
1991 printf("Note1: The lines in wordlistfile must be up to 2048 chars/bytes and end with CRLF i.e. maximum 2050bytes long.\n");
1992 printf("Note2: The goal is to sort a file unfittable in physical memory, and with 32bit code.\n");
1993 printf("Note3: when /fast is specified wordlistfile is uploaded into internal memory if it is possible.\n");
1994 printf("Note4: when /slow is specified wordlistfile is NOT uploaded into internal memory even if it is possible.\n");
1995
1996 //printf("Note1: wordlistfile1's lines encountered in wordlistfile2's lines go to 'Overlapped.txt' file.\n");
1997 //printf("Note2: wordlistfile1's lines blended (no repetitions allowed) with wordlistfile2's lines go to 'Blended.txt' file.\n");
1998 //printf("Note3: wordlistfile1's lines not encountered in wordlistfile2 go to 'Unfamiliar.txt' file.\n");
1999 //printf("Note1a: (wordlistfile1 logical_AND wordlistfile2) = 'Overlapped.txt' file.\n");
2000 //printf("Note2a: (wordlistfile1 logical_OR wordlistfile2) = 'Blended.txt' file.\n");
2001 //printf("Note3a: wordlistfile1 - (wordlistfile1 logical_AND wordlistfile2) = 'Unfamiliar.txt' file.\n");

```

```

2002 //printf("Note4: If you need only one file to be sorted-and-deduplicated then use this:\n");
2003 //printf("D:\\_KAZE_new-stuff\\Overlapper-Blender_r1>copy con Empty\n");
2004 //printf("AZ\n");
2005 //printf("      1 file(s) copied.\n");
2006 //printf("\n");
2007 //printf("D:\\_KAZE_new-stuff\\Overlapper-Blender_r1>dir Empty\n");
2008 //printf("03/02/2011  08:59 PM          0 Empty\n");
2009 //printf("\n");
2010 //printf("D:\\_KAZE_new-stuff\\Overlapper-Blender_r1>Overlapper-Blender wordlistfile1 Empty\n");
2011 //printf("Note5: Current pool(due to 32bit address limitation) for incoming strings is 1GB.\n");
2012 exit (1);
2013 }
2014
2015 if( strcmp(argv[2], ARG_CLs) != 0 && strcmp(argv[2], ARG_CLf) != 0 ) exit (1);
2016
2017  clocks0 = clock();
2018
2019 if( ( fp_in = fopen( argv[1], "rb" ) ) == NULL )
2020 { printf( "QuickSortExternal_4+GB: Can't open file %s \n", argv[1] ); return( 1 ); }
2021
2022 #if defined(_WIN32_ENVIRONMENT_)
2023 // 64bit:
2024 _lseeki64( fileno(fp_in), 0L, SEEK_END );
2025 size_in64 = _telli64( fileno(fp_in) );
2026 _lseeki64( fileno(fp_in), 0L, SEEK_SET );
2027 #else
2028 // 64bit:
2029 fseeko( fp_in, 0L, SEEK_END );
2030 size_in64 = ftello( fp_in );
2031 fseeko( fp_in, 0L, SEEK_SET );
2032 #endif /* defined(_WIN32_ENVIRONMENT_) */
2033
2034 //fseek( fp_in, 0L, SEEK_END );
2035 //size_in64 = ftell( fp_in );
2036 //fseek( fp_in, 0L, SEEK_SET );
2037 //printf( "Size of 1st input file: %llu\n", size_in64 );
2038 //printf( "Size of input file: %llu\n", size_in64 ); // ?! doesn't report the correct size whne 4+GB ?!
2039 printf( "Size of input file: %s\n", _ui64toaKAZEcomma(size_in64, llToaDigits, 10));
2040
2041 // if( ( fp_in2 = fopen( argv[2], "rb" ) ) == NULL )
2042 // { printf( "Overlapper-Blender: Can't open file %s \n", argv[2] ); return( 1 ); }
2043 //
2044 // #if defined(_WIN32_ENVIRONMENT_)
2045 // // 64bit:
2046 // _lseeki64( fileno(fp_in2), 0L, SEEK_END );
2047 // size_in2 = _telli64( fileno(fp_in2) );
2048 // _lseeki64( fileno(fp_in2), 0L, SEEK_SET );
2049 // #else
2050 // // 64bit:
2051 // fseeko( fp_in2, 0L, SEEK_END );
2052 // size_in2 = ftello( fp_in2 );
2053 // fseeko( fp_in2, 0L, SEEK_SET );
2054 // #endif /* defined(_WIN32_ENVIRONMENT_) */
2055 //
2056 // //fseek( fp_in2, 0L, SEEK_END );
2057 // //size_in2 = ftell( fp_in2 );
2058 // //fseek( fp_in2, 0L, SEEK_SET );
2059 // printf( "Size of 2nd input file: %llu\n", size_in2 );
2060 //
2061 // if( ( fp_out2 = fopen( "Overlapped.txt", "wb+" ) ) == NULL )
2062 // { printf( "Overlapper-Blender: Can't create file 'Overlapped.txt'.\n" ); return( 1 ); }
2063 // if( ( fp_out = fopen( "Blended.txt", "wb+" ) ) == NULL )
2064 // { printf( "Overlapper-Blender: Can't create file 'Blended.txt'.\n" ); return( 1 ); }
2065 // if( ( fp_out3 = fopen( "Unfamiliar.txt", "wb+" ) ) == NULL )
2066 // { printf( "Overlapper-Blender: Can't create file 'Unfamiliar.txt'.\n" ); return( 1 ); }
2067
2068 if( ( fp_out = fopen( "QuickSortExternal_4+GB.txt", "wb+" ) ) == NULL )
2069 { printf( "QuickSortExternal_4+GB: Can't create file 'QuickSortExternal_4+GB.txt'.\n" ); return( 1 ); }
2070
2071 if( ( fp_outRG = fopen( "QuickSortExternal_4+GB.distinct.txt", "wb+" ) ) == NULL )
2072 { printf( "QuickSortExternal_4+GB: Can't create file 'QuickSortExternal_4+GB.distinct.txt'.\n" ); return( 1 ); }
2073
2074 //if( ( fp_out = fopen( "Left-File_Lines_Found_In_Right-File.txt", "wb+" ) ) == NULL )
2075 //{ printf( "Compare_Two_Wordlists: Can't create file 'Left-File_Lines_Found_In_Right-File.txt' \n" ); return( 1 ); }
2076 //if( ( fp_out2 = fopen( "Left-File_Lines_Not-Found_In_Right-File.txt", "wb+" ) ) == NULL )
2077 //{ printf( "Compare_Two_Wordlists: Can't create file 'Left-File_Lines_Not-Found_In_Right-File.txt' \n" ); return( 1 ); }
2078
2079 printf( "Counting lines ...\n" );
2080 fsetpos(fp_in, &fsetpos_ZERO); // SOMETHING ROTTEN with lseeki64/fseeko and fsetpos ???! So DO-IT-OVER.
2082 // 1++_NON_Physical: [ =====
2083 Strnglen64 = size_in64;
2084
2085 Over4billionLines = 0;
2086 for (ThunderwithL64=0; ThunderwithL64<Strnglen64; ThunderwithL64++)
2087 { fread(&OneChar_ieByte, 1, 1, fp_in);
2088   if (OneChar_ieByte == 13) {Over4billionLines++;}
2089 }

```

```

2090
2091 //fseek(fp_in, 0, SEEK_SET);
2092 // _CRTIMP __int64 __cdecl _lseeki64(int, __int64, int);
2093 // int fseeko64 (FILE *stream, off64_t offset, int whence)
2094
2095 #if defined(_WIN32_ENVIRONMENT_)
2096 _lseeki64( fileno(fp_in), 0L, SEEK_SET );
2097 #else
2098 fseeko( fp_in, 0L, SEEK_SET );
2099 #endif /* defined(_WIN32_ENVIRONMENT_) */
2100 fsetpos(fp_in, &fsetpos_ZERO); // SOMETHING ROTTEN with lseeki64/fseeko and fsetpos ???! So DO-IT-OVER.
2101
2102 PointerToSeekPosition = (unsigned long long *)malloc( (Over4billionLines + 1)*sizeof(unsigned long long) ); // sizeof(string) is 4 ; +1 for QS
sentinel
2103 if( PointerToSeekPosition == NULL )
2104 { puts( "QuickSortExternal_4+GB: Needed memory allocation denied!\n" ); return( 1 ); }
2105 printf( "Allocated memory for pointers-to-words in MB: %lu\n", (((Over4billionLines + 1)*sizeof(unsigned long long))>>20)+1 ); // 8bytes per
element
2106
2107 printf( "Assigning pointers ...\n" );
2108 // ASSIGNING POINTERS ... [
2109 SeekPosition = 0;
2110 for( j_Over4billion = 1; j_Over4billion <= Over4billionLines; j_Over4billion++ )
2111 {
2112     PointerToSeekPosition[j_Over4billion] = SeekPosition;
2113 //printf("%llu\n",PointerToSeekPosition[j_Over4billion]);
2114 //printf("%llu\n",SeekPosition);
2115     for ( ;; ) {
2116         fread(&OneChar_ieByte, 1, 1, fp_in);
2117         if (OneChar_ieByte == 10) break;
2118         SeekPosition++;
2119     }
2120     SeekPosition++;
2121 }
2122 // ASSIGNING POINTERS ... ]
2123
2124 #if defined(_WIN32_ENVIRONMENT_)
2125 _lseeki64( fileno(fp_in), 0L, SEEK_SET );
2126 #else
2127 fseeko( fp_in, 0L, SEEK_SET );
2128 #endif /* defined(_WIN32_ENVIRONMENT_) */
2129
2130 if( strcmp(argv[2], ARG_CLF) == 0 )
2131 printf( "Trying to allocate memory for the file itself in MB: %lu ... ", ((size_in64 + 1)>>20)+1 ); // 8bytes per element
2132 if( strcmp(argv[2], ARG_CLS) == 0 )
2133 PoolPhysical = NULL;
2134 else
2135 PoolPhysical = (char *)malloc( (size_in64 + 1) ); // sizeof(string) is 4 ; +1 for QS sentinel
2136 if( PoolPhysical == NULL )
2137 { printf( "UNSUCCESSFUL! Get on with slow external accesses.\n" ); }
2138 else
2139 { printf( "OK! Get on with fast internal accesses.\n" );
2140     printf( "Uploading ...\n" );
2141     fread(PoolPhysical, size_in64, 1, fp_in);
2142 }
2143
2144 //printf( "Sorting %llu Pointers ...\n", Over4billionLines); // %llu %lu %d where is the law?
2145 printf( "Sorting %s Pointers ...\n", _ui64toaKAZEcomma(Over4billionLines, l1TOaDigits, 10));
2146
2147 // SORTING POINTERS ... [
2148 //void mkqsort_main(unsigned char **a, int n) { mkqsort(a, n, 0); }
2149 //mkqsort_main(ASbackup, nlines);
2150 //inssort_NON_Physical(ASbackup, nlines,0);
2151 // SORTING POINTERS ... ]
2152
2153 //printf ("%d\n",CompareStringsEndingWith13(PointerToSeekPosition[1],PointerToSeekPosition[2])); // Returns 1 if 1>2
2154 //printf ("%d\n",CompareStringsEndingWith13(PointerToSeekPosition[2],PointerToSeekPosition[1])); // Returns -1 if 1<2
2155 //printf ("%d\n",CompareStringsEndingWith13(PointerToSeekPosition[1],PointerToSeekPosition[1])); // Returns 0 if 1==2
2156
2157 //exit(1);
2158
2159
2160 // QuickSort Standalone [ -----
2161 //unsigned long Nbr2Sort = 100000000; // Number of items to sort.
2162 //unsigned char RandNbrs[100000001]; // Must be + 1 because of using [0] as a sentinel.
2163 Nbr2Sort = Over4billionLines;
2164
2165 //*****
2166 //
2167 // QuickSort
2168 //
2169 // QuickSort has long had a reputation for being the fastest general purpose
2170 // sort algorithm. It is also perhaps the most difficult to code, and is
2171 // subject to sharply adverse execution time if the "pivot values" are picked
2172 // poorly - which can happen if the data to be sorted is already partially
2173 // sorted.
2174 //
2175 // The algorithm works by picking one of the elements to be sorted as a "pivot

```

```

2176 // value". The list of items to be sorted is then partitioned so that all
2177 // elements that have a value less than the pivot end up in the front portion
2178 // of the array while all elements that are greater than the pivot value end
2179 // up in the other end. (Elements that are equal to the pivot could end up in
2180 // either section.) After the first pass, the array of items to be sorted
2181 // looks like:
2182 //
2183 //
2184 //          Low elements are here      Pivot      Higher elements are here
2185 //          -----
2186 // RandNbrs | | | | | | | | | | | | | | | |
2187 //          -----
2188 //          1  2  3  4                                Nbr2Sort
2189 //
2190 // After round 1, the QuickSort process is applied to both of the 2 subgroups.
2191 // whichever subgroup was smaller is processed immediately while the location
2192 // of the left and right ends of the larger subgroup are placed on a stack
2193 // for later processing. This processing order will guarantee that the stack
2194 // will never exceed Log2(Nbr2Sort) items.
2195 //
2196 // The repetitive processing of subgroups continues until the size of a
2197 // subgroup falls below a size defined by "MinGroup". Once a subgroup is
2198 // smaller than this, it is not sorted further by QuickSort. Small groups can
2199 // be processed faster by Insertion Sort. When QuickSort has reduced all
2200 // subgroups to < "MinGroup" size, control passes to "Insertion Sort" for a
2201 // final pass through the entire array.
2202 //
2203 // In the "old days", the optimal size for "MinGroup" was about 18. The cache
2204 // memory on current processor chips reduces the time to access anything in
2205 // the cache - which includes the part of the array that is currently residing
2206 // in the cache. This greatly increases the efficiency of the final "Insertion
2207 // Sort" relative to the quicksort portion. Thus, significantly larger values
2208 // for "MinGroup" work better when a cache is being used. (You can experiment
2209 // with the value that is assigned to "MinGroup".)
2210 //
2211 // Selection of the "pivot value" is crucial to the efficiency of Quicksort.
2212 // If the pivot value is selected so that it evenly partitions a subgroup,
2213 // then Quicksort is very efficient. On the other hand, if the value of the
2214 // "Pivot item" is near either the lowest or highest values that are going to
2215 // be partitioned within any subgroup, that particular round of Quicksort will
2216 // not do its job of quickly splitting the subgroups into ever smaller sizes.
2217 //
2218 // The "median of three" portion of the routine is an effort to pick a good
2219 // "pivot value". If a "pivot value" can be picked so that it exactly splits a
2220 // subgroup into 2 equal portions, then Quicksort will be as efficient as
2221 // possible. An effort is made to do this by trying to find a value which is
2222 // close to the median of the subgroup. This is done by checking the values at
2223 // the second, last, and middle positions within a subgroup. The middle value
2224 // of these three is used as the "pivot value" while the two extremes are
2225 // placed at the two ends of the subgroup.
2226 //
2227 // The code given here is based on a flyer that Robert Sedgewick (author of
2228 // "Algorithms") handed out "a few years ago" during a 2-semester sequence of
2229 // "Analysis of Algorithms". (Professor Sedgewick is 2nd from the left in the
2230 // center photo at http://groups.yahoo.com/group/CSAtrium/)
2231 //
2232 //*****
2233 //
2234 //void QuickSort(void) {
2235 //
2236 //unsigned long long QSleft[512];           // Stack arrays for the left/right
2237 //unsigned long long QSright[512];          // ends of subgroups within QuickSort()
2238 //
2239 // unsigned long long i, j, k, StackPtr;
2240 // unsigned long long LeftEnd, RightEnd, LeftPtr, RightPtr, MidPtr, MinGroup;
2241 // unsigned long long Pvalue, temp;
2242 // long clocks1, clocks2;
2243 //
2244 printf( "Pass #1: Quicksort started ...\n");
2245 //
2246 clocks1 = clock();
2247 //
2248 // RandNbrs[0] = 0;                               // Sentinel for sort - used
2249 //                                           // by the Insertion Sort
2250 //                                           // portion.
2251 PointerToSeekPosition[0] = 0; // In fact not needed: look the comment below at insertionsort.
2252 //
2253 // Initialize left end, right end, stack pointer,
2254 // and minimum size for subgroups.
2255 //
2256 LeftEnd = 1;                                     // For the first round, the 2
2257 RightEnd = Nbr2Sort;                             // ends will be the whole array
2258 MinGroup = 16384/32;                             // Years ago this would be ~18
2259 MinGroup = 16; //12 gives NumberOfComparisons: 1,585,788,132
2260 // 3 gives NumberOfComparisons: 1,613,651,713
2261 //
2262 //printf("Name:          AMD Athlon XP\n");
2263 //printf("Code Name:      Barton\n");

```

```

2264 //printf("Specification:      AMD Athlon(tm) XP 2600+\n");
2265 //printf("Technology:         0.13 microns\n");
2266 //printf("CPU Clock Speed:     1919.6 MHz\n");
2267 //printf("L1 Data Cache:       64 KBytes, 2-way set associative, 64 Bytes line size\n");
2268 //printf("L1 Instruction Cache: 64 KBytes, 2-way set associative, 64 Bytes line size\n");
2269 //printf("L2 Cache:            512 KBytes, 16-way set associative, 64 Bytes line size\n");
2270 //printf("L2 Speed:             1919.6 MHz (Full)\n");
2271 //printf("MinGroup =             16*1byte, gives 8984 clocks.\n");
2272 //printf("MinGroup =             64*1byte, gives 8469 clocks.\n");
2273 //printf("MinGroup =            256*1byte, gives 7954 clocks.\n");
2274 //printf("MinGroup =           1024*1byte, gives 7390 clocks.\n");
2275 //printf("MinGroup =           4096*1byte, gives 6937 clocks.\n");
2276 //printf("MinGroup =          16384*1byte, gives 6719 clocks.\n");
2277 //printf("MinGroup =         65536*1byte, gives 11500 clocks.\n");
2278 //printf("MinGroup =        262144*1byte, gives 73781 clocks.\n");
2279 //printf("MinGroup =       1048576*1byte, gives 1249734 clocks.\n");
2280
2281
2282 if (Nbr2Sort > MinGroup)                // Run quicksort until no
2283     StackPtr = 1;                        // subgroup remains larger
2284 else StackPtr = 0;                       // than "MinGroup" elements.
2285
2286
2287 // Start quicksort. First, set the pivot value equal to the median of the
2288 // array values at RandNbrs[LeftEnd+1], RandNbrs[(LeftEnd+RightEnd)/2],
2289 // and RandNbrs[RightEnd]. The minimum of these 3 is placed at
2290 // RandNbrs[LeftEnd+1] while the maximum is placed at RandNbrs[RightEnd].
2291 // The value at RandNbrs[LeftEnd] is moved to
2292 // RandNbrs[(LeftEnd+RightEnd)/2].
2293
2294 while (StackPtr) {                      // Loop until all subgroups
2295                                         // are partitioned down to
2296                                         // <= "MinGroup" size.
2297     LeftPtr = LeftEnd + 1;              // Ptr to left end.
2298     RightPtr = RightEnd;                // Ptr to right end.
2299     MidPtr = (LeftEnd + RightEnd)/2;    // Point to middle
2300
2301     NumberOfSplittings++;
2302
2303     if (NumberOfSplittings % (1*256) == 1) {
2304         Melnitchka = Melnitchka & 3; // 0 1 2 3: 00 01 10 11
2305         printf( "%s RightEnd-LeftEnd: %s; ", Auberge[Melnitchka++], _ui64toaKAZEzerocomma(RightEnd-LeftEnd, 11ToADigits, 10)+(26-15));
2306         printf( "NumberOfSplittings: %s ...\r", _ui64toaKAZEzerocomma(NumberOfSplittings, 11ToADigits, 10)+(26-13));
2307     }
2308
2309     // Start sort of these 3
2310     if (RandNbrs[LeftPtr] > RandNbrs[RightPtr]) {
2311         if (CompareStringsEndingWith13(PointerToSeekPosition[LeftPtr], PointerToSeekPosition[RightPtr], PoolPhysical) == 1) {
2312             temp = RandNbrs[LeftPtr]; // elements
2313             temp = PointerToSeekPosition[LeftPtr];
2314             RandNbrs[LeftPtr] = RandNbrs[RightPtr];
2315             PointerToSeekPosition[LeftPtr] = PointerToSeekPosition[RightPtr];
2316             RandNbrs[RightPtr] = temp;
2317             PointerToSeekPosition[RightPtr] = temp;
2318         }
2319
2320         if (RandNbrs[MidPtr] > RandNbrs[RightPtr]) {
2321             if (CompareStringsEndingWith13(PointerToSeekPosition[MidPtr], PointerToSeekPosition[RightPtr], PoolPhysical) == 1) {
2322                 Pvalue = RandNbrs[RightPtr];
2323                 Pvalue = PointerToSeekPosition[RightPtr];
2324                 RandNbrs[RightPtr] = RandNbrs[MidPtr];
2325                 PointerToSeekPosition[RightPtr] = PointerToSeekPosition[MidPtr];
2326             }
2327             else if (RandNbrs[MidPtr] < RandNbrs[LeftPtr]) {
2328                 else if (CompareStringsEndingWith13(PointerToSeekPosition[MidPtr], PointerToSeekPosition[LeftPtr], PoolPhysical) == -1) {
2329                     Pvalue = RandNbrs[LeftPtr];
2330                     Pvalue = PointerToSeekPosition[LeftPtr];
2331                     RandNbrs[LeftPtr] = RandNbrs[MidPtr];
2332                     PointerToSeekPosition[LeftPtr] = PointerToSeekPosition[MidPtr];
2333                 }
2334                 else Pvalue = RandNbrs[MidPtr];
2335             else Pvalue = PointerToSeekPosition[MidPtr];
2336             // The 3 values are sorted and
2337             // and the median is in Pvalue
2338             RandNbrs[MidPtr] = RandNbrs[LeftEnd]; // Fill in hole with LeftEnd
2339             PointerToSeekPosition[MidPtr] = PointerToSeekPosition[LeftEnd];
2340
2341             // Start the main loop. Move pointers inward until
2342             // we find 2 elements that have to be exchanged.
2343
2344             while (RandNbrs[++LeftPtr] < Pvalue); // Set up pointers
2345             while (CompareStringsEndingWith13(PointerToSeekPosition[++LeftPtr], Pvalue, PoolPhysical) == -1);
2346             while (RandNbrs[--RightPtr] > Pvalue); // for 1st exchange
2347             while (CompareStringsEndingWith13(PointerToSeekPosition[--RightPtr], Pvalue, PoolPhysical) == 1);
2348             while (LeftPtr < RightPtr) { // Make these
2349                 temp = RandNbrs[LeftPtr]; // statements as
2350                 temp = PointerToSeekPosition[LeftPtr];
2351                 RandNbrs[LeftPtr] = RandNbrs[RightPtr]; // efficient as

```



```

2352 PointerToSeekPosition[LeftPtr] = PointerToSeekPosition[RightPtr];
2353 // RandNbrs[RightPtr] = temp; // possible.
2354 PointerToSeekPosition[RightPtr] = temp;
2355 // while (RandNbrs[++LeftPtr] < Pvalue); // Continue this loop until
2356 while (CompareStringsEndingWith13(PointerToSeekPosition[++LeftPtr],Pvalue, PoolPhysical) == -1);
2357 // while (RandNbrs[--RightPtr] > Pvalue); // the pointers cross.
2358 while (CompareStringsEndingWith13(PointerToSeekPosition[--RightPtr],Pvalue, PoolPhysical) == 1);
2359 }
2360
2361 // RandNbrs[LeftEnd] = RandNbrs[RightPtr]; // After pointers cross, fill
2362 PointerToSeekPosition[LeftEnd] = PointerToSeekPosition[RightPtr];
2363 // RandNbrs[RightPtr] = Pvalue; // left end and middle hole.
2364 PointerToSeekPosition[RightPtr] = Pvalue;
2365
2366 // All values to the left of RandNbrs[RightPtr] are <= Pvalue while all to
2367 // the right are >= Pvalue. Next, test the 2 subgroups on either side to
2368 // see if they are still larger than the minimum efficient size. If both
2369 // are still too large, then place the larger one on the stack and
2370 // partition the smaller. If only one needs partitioning, then partition
2371 // it, otherwise get the left and right ends of a subgroup stored on the
2372 // stack in an earlier operation.
2373
2374 // Move RightPtr into
2375 RightPtr--; // unsorted left subgroup
2376
2377 if (RightPtr < MidPtr) { // If left SubGroup is smaller
2378     if (RightPtr - LeftEnd > MinGroup) { // If both are large then put
2379         Qsleft[StackPtr] = LeftPtr; // right side on the stack
2380         Qsright[StackPtr] = RightEnd; // and sort the left side.
2381         RightEnd = RightPtr;
2382         ++StackPtr; // Ready for next subgroup
2383     }
2384     else if (RightEnd - LeftPtr > MinGroup) // Else if just have to
2385         LeftEnd = LeftPtr; // sort the right side
2386     else { // Else neither gets sorted. Get a
2387         if (--StackPtr!=0) {
2388             LeftEnd = Qsleft[StackPtr]; // prior subgroup from the stack.
2389             RightEnd = Qsright[StackPtr]; // (will be garbage if all
2390         }
2391     } // subgroups are sorted)
2392 } // End of "if left is smaller"
2393
2394 else { // Else left side is larger
2395     if (RightEnd - LeftPtr > MinGroup) { // If both sides are large
2396         Qsleft[StackPtr] = LeftEnd; // then put left side on
2397         Qsright[StackPtr] = RightPtr; // the stack
2398         LeftEnd = LeftPtr; // and sort the right side
2399         ++StackPtr; // Ready for next subgroup
2400     }
2401     else if (RightPtr - LeftEnd > MinGroup) // else if left side is
2402         RightEnd = RightPtr; // too large, then sort it.
2403     else { // Else neither gets sorted. Get a
2404         if (--StackPtr!=0) {
2405             LeftEnd = Qsleft[StackPtr]; // prior subgroup from the stack
2406             RightEnd = Qsright[StackPtr]; // (will be garbage if all
2407         }
2408     } // subgroups are sorted).
2409 } // End of "if left is larger"
2410
2411 if (StackPtr==0)
2412 {
2413     Melnitckha = Melnitckha & 3; // 0 1 2 3: 00 01 10 11
2414     printf( "%s RightEnd-LeftEnd: %s; ", Auberge[Melnitckha++], _ui64toaKAZEzerocomma(RightEnd-LeftEnd, 11ToaDigits, 10)+(26-15));
2415     printf( "NumberOfSplittings: %s ...\n", _ui64toaKAZEzerocomma(NumberOfSplittings, 11ToaDigits, 10)+(26-13));
2416 }
2417
2418 } // Repeat until all subgroups are
2419 // small.
2420
2421 // Finish up with "Insertion Sort"
2422
2423 //goto SkipInsertion;
2424 printf( "Pass #2: Insertionsort started ...\n");
2425
2426 //void insort_NON_Physical(unsigned char **a, int n, int d)
2427 //{ unsigned char **pi, **pj, *s, *t;
2428 // for (pi = a + 1; --n > 0; pi++)
2429 // for (pj = pi; pj > a; pj--) {
2430 // /* Inline strcmp: break if *(pj-1) <= *pj */
2431 // for (s=(pj-1)+d, t=*pj+d; *s==*t && *s!=0; s++, t++)
2432 // ;
2433 // if (*s <= *t)
2434 // break;
2435 // swap2(pj, pj-1);
2436 // }
2437 //}
2438
2439 //void StraightInsertion(TElem M[], int N)

```

```

2440 // {
2441 //   for (int i = 2; i <= N; i++)
2442 //   {
2443 //       TElem X = M[i]; int j = i-1; M[0] = X;
2444 //       while (X.Key < M[j].Key) M[j+1] = M[j--];
2445 //       M[j+1] = X;
2446 //   }
2447 // }
2448
2449 //void BinaryInsertion(TElem M[], int N)
2450 // {
2451 //   for (int i = 2; i <= N; i++)
2452 //   { TElem X = M[i];
2453 //     int L = 1,
2454 //       R = i-1;
2455 //     while (L <= R)
2456 //     { int Med = (L+R) / 2;
2457 //       if (X.Key < M[Med].Key) R = Med-1; else L = Med+1; }
2458 //     for (int j = i-1; j >= L; j--) M[j+1] = M[j];
2459 //     M[L] = X;
2460 //   }
2461 // }
2462
2463   for (i = 2; i <= Nbr2Sort; i++) {
2464   if (i % (16*1024) == 1) {
2465   Melnitchka = Melnitchka & 3; // 0 1 2 3: 00 01 10 11
2466   printf( "%s i: %s ...\r", Auberge[Melnitchka++], _ui64toaKAZEzerocomma(i, lToaDigits, 10)+(26-15));
2467   }
2468
2469   j = i - 1;
2470   //   temp = RandNbrs[k];
2471   temp = PointerToSeekPosition[i];
2472   PointerToSeekPosition[0] = temp;
2473   //   while (RandNbrs[j] > temp) {
2474   while (CompareStringsEndingWith13(PointerToSeekPosition[j],temp, PoolPhysical) == 1) {
2475   //       RandNbrs[k] = RandNbrs[j];
2476   //       PointerToSeekPosition[j+1] = PointerToSeekPosition[j];
2477   //       j--;
2478   //   }
2479   //   RandNbrs[k] = temp;
2480   PointerToSeekPosition[j+1] = temp;
2481   }
2482
2483   Melnitchka = Melnitchka & 3; // 0 1 2 3: 00 01 10 11
2484   printf( "%s i: %s ...\n", Auberge[Melnitchka++], _ui64toaKAZEzerocomma(i-1, lToaDigits, 10)+(26-15));
2485
2486   //SkipInsertion:
2487
2488   // Insertionsort(above fragment) follows:
2489   /*
2490   mov eax, DWORD PTR _Nbr2Sort
2491   $L1481:
2492   ; Line 431
2493   cmp eax, 2
2494   jb  SHORT $L1514
2495   mov ecx, OFFSET FLAT:_RandNbrs+1
2496   mov ebp, 2
2497   sub ebp, ecx
2498   $L1512:
2499   ; Line 434
2500   mov dl, BYTE PTR [ecx+1]
2501   ; Line 435
2502   cmp BYTE PTR [ecx], dl
2503   lea edi, DWORD PTR [ecx+ebp]
2504   jbe SHORT $L1517
2505   mov esi, ecx
2506   mov eax, ecx
2507   $L1516:
2508   ; Line 436
2509   mov bl, BYTE PTR [esi]
2510   mov BYTE PTR [eax+1], bl
2511   mov bl, BYTE PTR [eax-1]
2512   ; Line 438
2513   dec edi
2514   dec eax
2515   cmp bl, dl
2516   mov esi, eax
2517   ja  SHORT $L1516
2518   ; Line 435
2519   mov eax, DWORD PTR _Nbr2Sort
2520   $L1517:
2521   ; Line 434
2522   inc ecx
2523   ; Line 440
2524   mov BYTE PTR _RandNbrs[edi], dl
2525   lea edx, DWORD PTR [ecx+ebp]
2526   cmp edx, eax
2527   jbe SHORT $L1512

```

```

2528 $L1514:
2529 */
2530
2531 /*
2532 // Bubblesort
2533     for(i=1;i<((Nbr2Sort+1)-1);i++)
2534         for(j=1;j<((Nbr2Sort+1)-(i+1));j++)
2535             if(RandNbrs[j] > RandNbrs[j+1])
2536                 { temp = RandNbrs[j+1];
2537                   RandNbrs[j+1] = RandNbrs[j];
2538                   RandNbrs[j] = temp;
2539                 }
2540 */
2541
2542 clocks2 = clock();
2543 //printf("The time to sort %llu items via Quicksort was %d clocks.\n", Nbr2Sort, clocks2 - clocks1);
2544 //Caramba again: why above line doesn't print the second argument?!?!
2545 //printf("The time to sort %llu items via Quicksort was ", Nbr2Sort);
2546
2547 printf( "NumberOfComparisons: %s\n", _ui64toaKAZEcomma(NumberOfComparisons, 11ToaDigits, 10));
2548
2549 printf( "The time to sort %s items via Quicksort+Insertionsort was ", _ui64toaKAZEcomma(Nbr2Sort, 11ToaDigits, 10));
2550 //printf("%d clocks.\n", clocks2 - clocks1);
2551 printf( "%s clocks.\n", _ui64toaKAZEcomma(clocks2 - clocks1, 11ToaDigits, 10));
2552
2553 //}
2554
2555 // QuickSort Standalone ] -----
2556
2557 printf( "Dumping the sorted data ...\n");
2558     for( j_Over4billion = 1; j_Over4billion <= Over4billionLines; j_Over4billion++ )
2559     {
2560 if( PoolPhysical == NULL ) {
2561 fsetpos(fp_in, &PointerToSeekPosition[j_Over4billion]);
2562 for (i=0; i<(LongestLineInclusive+1)+1; i++) {fread(&FourGram[i], 1, 1, fp_in); fwrite(&FourGram[i], 1, 1, fp_out); if (FourGram[i]==10)
break;}
2563 } else {
2564 for (i=0; i<(LongestLineInclusive+1)+1; i++) {
2565 FourGram[i] = *(char *) (PoolPhysical + PointerToSeekPosition[j_Over4billion] + i);
2566 fwrite(&FourGram[i], 1, 1, fp_out);
2567 if (FourGram[i]==10) break;
2568 }
2569 }
2570
2571 //for (i=0; i<32+1; i++) {printf("%c",FourGram[i]); if (FourGram[i]==10) break;}
2572 //printf("\n");
2573 }
2574 printf( "Dumped %s lines.\n", _ui64toaKAZEcomma(j_Over4billion-1, 11ToaDigits, 10));
2575
2576 fclose(fp_out);
2577 if( ( fp_out = fopen( "QuickSortExternal_4+GB.txt", "rb" ) ) == NULL )
2578 { printf( "QuickSortExternal_4+GB: Can't open file 'QuickSortExternal_4+GB.txt'.\n" ); return( 1 ); }
2579
2580 #if defined(_WIN32_ENVIRONMENT_)
2581 // 64bit:
2582 _lseeki64( fileno(fp_out), 0L, SEEK_END );
2583 size_in2 = _telli64( fileno(fp_out) );
2584 _lseeki64( fileno(fp_out), 0L, SEEK_SET );
2585 #else
2586 // 64bit:
2587 fseeko( fp_out, 0L, SEEK_END );
2588 size_in2 = ftello( fp_out );
2589 fseeko( fp_out, 0L, SEEK_SET );
2590 #endif /* defined(_WIN32_ENVIRONMENT_) */
2591
2592 fclose(fp_out);
2593 if (size_in2 == size_in64)
2594 printf( "OK! Incoming and resultant file's sizes match.\n");
2595 else
2596 { printf( "Failure! Incoming and resultant file's sizes mismatch?!\n"); exit(13);}
2597
2598 printf( "Dumping the sorted data [deduplicated] ...\n");
2599
2600 // Caution: the difference from above fragment is the first array element: 1 above 0 below!
2601 nlines1 = Over4billionLines;
2602 linecounterNotRL = Over4billionLines;
2603 linecounterRL = 0;
2604 // Railgunning ... [
2605 //jFirst = 0;
2606 jFirst = 0+1;
2607
2608 if (linecounterNotRL == 1) {
2609 //fprintf(fp_outRG, "%s", backupPAT1[jFirst]);
2610 //fwrite(CRdLFa, 2, 1, fp_outRG );
2611 // fprintf(fp_outRG, "%lu\n", 1);
2612 fprintf( fp_outRG, "%s\t", _ui64toaKAZEzerocomma(1, 11ToaDigits, 10)+(26-9) );
2613
2614 if( PoolPhysical == NULL ) {
2615 fsetpos(fp_in, &PointerToSeekPosition[jFirst]);
2616 for (i=0; i<(LongestLineInclusive+1)+1; i++) {fread(&FourGram[i], 1, 1, fp_in); fwrite(&FourGram[i],

```

```

1, 1, fp_outRG); if (FourGram[i]==10) break;}
2615 } else {
2616 for (i=0; i<(LongestLineInclusive+1)+1; i++) {
2617 FourGram[i] = *(char *) (PoolPhysical + PointerToSeekPosition[jFirst] + i);
2618 fwrite(&FourGram[i], 1, 1, fp_outRG);
2619 if (FourGram[i]==10) break;
2620 }
2621 }
2622 linecounterRL++;
2623 }
2624 jPrevLine = jFirst;
2625 linecounterNotRL--;
2626 TimesOverItem = 1;
2627 for( j_RG = jFirst + 1; j_RG < nlinesl+1; j_RG++ )
2628 {
2629 if (linecounterNotRL != 0) {
2630 linecounterNotRL--;
2631
2632 //if ( strcmpKAZE(backupPAT1[jPrevLine], backupPAT1[j_RG]) != 0 ) {
2633 if ( CompareStringsEndingWith13(PointerToSeekPosition[jPrevLine],PointerToSeekPosition[j_RG], PoolPhysical) != 0 ) {
2634 //fprintf(fp_outRG, "%s", backupPAT1[jPrevLine]);
2635 //fwrite(CRdLFa, 2, 1, fp_outRG );
2636 // fprintf(fp_outRG, "%lu\n", TimesOverItem);
2637 fprintf( fp_outRG, "%s\t", _ui64toaKAZEzerocomma(TimesOverItem, 11ToaDigits, 10)+(26-9) );
2638
2639 if( PoolPhysical == NULL ) {
2640 fsetpos(fp_in, &PointerToSeekPosition[jPrevLine]);
2641 for (i=0; i<(LongestLineInclusive+1)+1; i++) {fread(&FourGram[i], 1, 1, fp_in); fwrite(&FourGram[i],
2642 1, 1, fp_outRG); if (FourGram[i]==10) break;}
2643 } else {
2644 for (i=0; i<(LongestLineInclusive+1)+1; i++) {
2645 FourGram[i] = *(char *) (PoolPhysical + PointerToSeekPosition[jPrevLine] + i);
2646 fwrite(&FourGram[i], 1, 1, fp_outRG);
2647 if (FourGram[i]==10) break;
2648 }
2649 }
2650 TimesOverItem = 0;
2651 linecounterRL++;
2652 }
2653 TimesOverItem++;
2654
2655 if (linecounterNotRL == 0) {
2656 //fprintf(fp_outRG, "%s", backupPAT1[j_RG]);
2657 //fwrite(CRdLFa, 2, 1, fp_outRG );
2658 // fprintf(fp_outRG, "%lu\n", TimesOverItem);
2659 fprintf( fp_outRG, "%s\t", _ui64toaKAZEzerocomma(TimesOverItem, 11ToaDigits, 10)+(26-9) );
2660 if( PoolPhysical == NULL ) {
2661 fsetpos(fp_in, &PointerToSeekPosition[j_RG]);
2662 for (i=0; i<(LongestLineInclusive+1)+1; i++) {fread(&FourGram[i], 1, 1, fp_in); fwrite(&FourGram[i],
2663 1, 1, fp_outRG); if (FourGram[i]==10) break;}
2664 } else {
2665 for (i=0; i<(LongestLineInclusive+1)+1; i++) {
2666 FourGram[i] = *(char *) (PoolPhysical + PointerToSeekPosition[j_RG] + i);
2667 fwrite(&FourGram[i], 1, 1, fp_outRG);
2668 if (FourGram[i]==10) break;
2669 }
2670 }
2671 linecounterRL++;
2672 }
2673 jPrevLine = j_RG;
2674 }
2675 // Railgunning ... ]
2676
2677 printf( "Dumped %s distinct lines.\n", _ui64toaKAZEcomma(linecounterRL, 11ToaDigits, 10));
2678
2679 free(PointerToSeekPosition); free(PoolPhysical);
2680 fclose(fp_outRG);
2681
2682 clocks3 = clock();
2683
2684 printf( "Total time: %s clocks.\n", _ui64toaKAZEcomma(clocks3 - clocks0, 11ToaDigits, 10));
2685
2686 #if defined(_WIN32_ENVIRONMENT_)
2687 printf( "Performance: %s KB/s.\n", _ui64toaKAZEcomma((size_in64>>10)/((clocks3 - clocks0 + 1)/1000 + 1), 11ToaDigits, 10));
2688 #else
2689 printf( "Performance: %s KB/s.\n", _ui64toaKAZEcomma((size_in64>>10)/((clocks3 - clocks0 + 1)/1000000 + 1), 11ToaDigits, 10));
2690 #endif /* defined(_WIN32_ENVIRONMENT_) */
2691
2692 printf( "Done successfully.\n");
2693 exit(0);
2694
2695 // 1+_NON_Physical: ] =====
2696 }
2697
2698 }
2699

```

```

2700 /*
2701 ;DEFNG A-Z
2702 ;SUB FoxQuickSort (Quantity&, HIMEMhandle%, EMBOffsetOfDATA&, MotherStringL%, Strings%, StringL%, EMBOffsetForStack&)
2703 ;SHARED VideoBuffer%, ZeroIfT5100%, c0%, c3%, c4%
2704 ;DIM XMS AS EMMSructure
2705 ;   Left = 1: Right = Quantity&: StackPtr = 0
2706 ;   ' PUSH Left
2707 ;       StackPtr = StackPtr + 1
2708 ;       XMS.SourceHandle = 0
2709 ;       CALL ToLong(XMS.SourceOffset, VARPTR(Left), VARSEG(Left))
2710 ;       XMS.NumberOfBytes = 4
2711 ;       XMS.TargetHandle = HIMEMhandle%
2712 ;       XMS.TargetOffset = EMBOffsetForStack& + (StackPtr - 1) * 4
2713 ;       CALL HIMEMMove(ErrorBL%, VARSEG(XMS), VARPTR(XMS), ErrorStatus%)
2714 ;   ' PUSH Right
2715 ;       StackPtr = StackPtr + 1
2716 ;       XMS.SourceHandle = 0
2717 ;       CALL ToLong(XMS.SourceOffset, VARPTR(Right), VARSEG(Right))
2718 ;       XMS.NumberOfBytes = 4
2719 ;       XMS.TargetHandle = HIMEMhandle%
2720 ;       XMS.TargetOffset = EMBOffsetForStack& + (StackPtr - 1) * 4
2721 ;       CALL HIMEMMove(ErrorBL%, VARSEG(XMS), VARPTR(XMS), ErrorStatus%)
2722 ;DO
2723 ;   ' POP Right
2724 ;       XMS.SourceHandle = HIMEMhandle%
2725 ;       XMS.SourceOffset = EMBOffsetForStack& + (StackPtr - 1) * 4
2726 ;       XMS.NumberOfBytes = 4
2727 ;       XMS.TargetHandle = 0
2728 ;       CALL ToLong(XMS.TargetOffset, VARPTR(Right), VARSEG(Right))
2729 ;       CALL HIMEMMove(ErrorBL%, VARSEG(XMS), VARPTR(XMS), ErrorStatus%)
2730 ;       StackPtr = StackPtr - 1
2731 ;   ' POP Left
2732 ;       XMS.SourceHandle = HIMEMhandle%
2733 ;       XMS.SourceOffset = EMBOffsetForStack& + (StackPtr - 1) * 4
2734 ;       XMS.NumberOfBytes = 4
2735 ;       XMS.TargetHandle = 0
2736 ;       CALL ToLong(XMS.TargetOffset, VARPTR(Left), VARSEG(Left))
2737 ;       CALL HIMEMMove(ErrorBL%, VARSEG(XMS), VARPTR(XMS), ErrorStatus%)
2738 ;       StackPtr = StackPtr - 1
2739 ;DO
2740 ;       Pivot$ = MID$(FoxDATA$((Left + Right) \ 2, HIMEMhandle%, EMBOffsetOfDATA&, MotherStringL%), Strings%, StringL%)
2741 ;       Indx = Left
2742 ;       Jndx = Right
2743 ;DO
2744 ;DO WHILE MID$(FoxDATA$(Indx, HIMEMhandle%, EMBOffsetOfDATA&, MotherStringL%), Strings%, StringL%) < Pivot$
2745 ;       Indx = Indx + 1
2746 ;LOOP
2747 ;DO WHILE MID$(FoxDATA$(Jndx, HIMEMhandle%, EMBOffsetOfDATA&, MotherStringL%), Strings%, StringL%) > Pivot$
2748 ;       Jndx = Jndx - 1
2749 ;LOOP
2750 ;   IF Indx <= Jndx THEN
2751 ;       IF Indx < Jndx THEN
2752 ;           ' Jndx -> Temp
2753 ;           Temp$ = FoxDATA$(Jndx, HIMEMhandle%, EMBOffsetOfDATA&, MotherStringL%)
2754 ;           ' Indx -> Jndx
2755 ;           XMS.SourceHandle = HIMEMhandle%
2756 ;           XMS.SourceOffset = EMBOffsetOfDATA& + (Indx - 1) * MotherStringL%
2757 ;           XMS.NumberOfBytes = MotherStringL%
2758 ;           XMS.TargetHandle = HIMEMhandle%
2759 ;           XMS.TargetOffset = EMBOffsetOfDATA& + (Jndx - 1) * MotherStringL%
2760 ;           CALL HIMEMMove(ErrorBL%, VARSEG(XMS), VARPTR(XMS), ErrorStatus%)
2761 ;           ' Temp -> Indx
2762 ;           XMS.SourceHandle = 0
2763 ;           CALL ToLong(XMS.SourceOffset, SADD(Temp$), SSEG(Temp$))
2764 ;           XMS.NumberOfBytes = MotherStringL%
2765 ;           XMS.TargetHandle = HIMEMhandle%
2766 ;           XMS.TargetOffset = EMBOffsetOfDATA& + (Indx - 1) * MotherStringL%
2767 ;           CALL HIMEMMove(ErrorBL%, VARSEG(XMS), VARPTR(XMS), ErrorStatus%)
2768 ;       END IF
2769 ;       Indx = Indx + 1
2770 ;       Jndx = Jndx - 1
2771 ;   END IF
2772 ;LOOP UNTIL Indx > Jndx
2773 ;
2774 ;IF Indx < Right THEN
2775 ;   ' PUSH Indx
2776 ;       StackPtr = StackPtr + 1
2777 ;       XMS.SourceHandle = 0
2778 ;       CALL ToLong(XMS.SourceOffset, VARPTR(Indx), VARSEG(Indx))
2779 ;       XMS.NumberOfBytes = 4
2780 ;       XMS.TargetHandle = HIMEMhandle%
2781 ;       XMS.TargetOffset = EMBOffsetForStack& + (StackPtr - 1) * 4
2782 ;       CALL HIMEMMove(ErrorBL%, VARSEG(XMS), VARPTR(XMS), ErrorStatus%)
2783 ;   ' PUSH Right
2784 ;       StackPtr = StackPtr + 1
2785 ;       XMS.SourceHandle = 0
2786 ;       CALL ToLong(XMS.SourceOffset, VARPTR(Right), VARSEG(Right))
2787 ;       XMS.NumberOfBytes = 4

```

```

2788 ;           XMS.TargetHandle = HIMEMhandle%
2789 ;           XMS.TargetOffset = EMBoffsetForStack& + (StackPtr - 1) * 4
2790 ;           CALL HIMEMMove(ErrorBL%, VARSEG(XMS), VARPTR(XMS), ErrorStatus%)
2791 ;END IF
2792 ;           Right = Jndx
2793 ;
2794 ;LOOP UNTIL Left >= Right
2795 ;LOOP UNTIL StackPtr = 0
2796 ;END SUB
2797 */
2798
2799 /*
2800 'DEFINT A-Z
2801 'SUB Qsdouble (SortArray#())
2802 'left = LBOUND(SortArray#)
2803 'right = UBOUND(SortArray#)
2804 'LeftMargin = left
2805 'REDIM Stack%(left TO right)
2806 'CONST LocalFalse = 0, LocalTrue = NOT LocalFalse
2807 'Tiller = LocalFalse
2808 'DO
2809 'IF left < right THEN
2810 '    Position = (left + right) \ 2
2811 '    SWAP SortArray#(Position), SortArray#(right)
2812 '    Pivot# = SortArray#(right)
2813 '    Indx = left
2814 '    Jndx = right
2815 'DO WHILE (Indx < Jndx) AND (SortArray#(Indx) <= Pivot#)
2816 '    Indx = Indx + 1
2817 'LOOP
2818 'DO WHILE (Jndx > Indx) AND (SortArray#(Jndx) >= Pivot#)
2819 '    Jndx = Jndx - 1
2820 'LOOP
2821 '    SWAP SortArray#(Indx), SortArray#(Jndx)
2822 'LOOP UNTIL Jndx = Indx
2823 '    SWAP SortArray#(Indx), SortArray#(right)
2824 'IF Indx - left > right - Indx THEN
2825 '    StackPtr = StackPtr + 1
2826 '    Stack%(StackPtr + LeftMargin) = left
2827 '    StackPtr = StackPtr + 1
2828 '    Stack%(StackPtr + LeftMargin) = Indx - 1
2829 '    left = Indx + 1
2830 'ELSE
2831 '    StackPtr = StackPtr + 1
2832 '    Stack%(StackPtr + LeftMargin) = Indx + 1
2833 '    StackPtr = StackPtr + 1
2834 '    Stack%(StackPtr + LeftMargin) = right
2835 '    right = Indx - 1
2836 'END IF
2837 'ELSE
2838 'IF StackPtr = 0 THEN
2839 '    Tiller = LocalTrue
2840 'ELSE
2841 '    right = Stack%(StackPtr + LeftMargin)
2842 '    StackPtr = StackPtr - 1
2843 '    left = Stack%(StackPtr + LeftMargin)
2844 '    StackPtr = StackPtr - 1
2845 'END IF
2846 'END IF
2847 'LOOP UNTIL Tiller
2848 'END SUB
2849 */
2850
2851 /*
2852 void quickSort(int l, int r) {
2853     int i = l, j = r, x = M[(i+j) / 2].Key;
2854     do {
2855         while (x > M[i].Key) i++;
2856         while (x < M[j].Key) j--;
2857         if (i <= j) {
2858             swap(M, i, j); i++; j--;
2859         }
2860     } while (j >= i);
2861     if (j > l) quickSort(l, j);
2862     if (i < r) quickSort(i, r);
2863 }
2864 */
2865
2866 /*
2867 // FILE LOADING & RAILGUNNING [ Country-Region-City
2868 printf("FINAL-SHOWDOWN ...\n");
2869 if ((linecounterNotRL = readlines("Column1.txt", &backupPAT1, 0)) >= 0)
2870 { printf("Number of Triads uploaded: %lu\n", linecounterNotRL);
2871   nlines1 = linecounterNotRL;
2872   printf("Flushing sorted & railguned Triads ...\n");
2873   if( ( fp_out = fopen( "Column1.txt", "wb+" ) ) == NULL )

```

```

2876 { printf( "IP_ELF: Can't create file %s \n", "Column1.txt" ); return( 1 ); }
2877 linecounterRL = 0;
2878
2879 printf("Sorting(with 'MultikeyQuickSortSort' by J. Bentley and R. Sedgewick) ...\n");
2880 mkqsort_main(backupPAT1, linecounterNotRL); // backup[0..nlines-1]
2881 // Railgunning ... [
2882 jFirst = 0;
2883
2884         if (linecounterNotRL == 1) {
2885             fprintf(fp_out, "%s", backupPAT1[jFirst]);
2886             fwrite(CRDLFa, 2, 1, fp_out );
2887             fprintf(fp_outC2, "%lu\n", 1);
2888             linecounterRL++;
2889         }
2890 jPrevLine = jFirst;
2891 linecounterNotRL--;
2892 TimesOverItem = 1;
2893 for( j = jFirst + 1; j < nlines1; j++ )
2894 {
2895     if (linecounterNotRL != 0) {
2896         linecounterNotRL--;
2897
2898         if ( strcmpKAZE(backupPAT1[jPrevLine], backupPAT1[j]) != 0 ) {
2899             fprintf(fp_out, "%s", backupPAT1[jPrevLine]);
2900             fwrite(CRDLFa, 2, 1, fp_out );
2901             fprintf(fp_outC2, "%lu\n", TimesOverItem);
2902             TimesOverItem = 0;
2903             linecounterRL++;
2904         }
2905         TimesOverItem++;
2906
2907         if (linecounterNotRL == 0) {
2908             fprintf(fp_out, "%s", backupPAT1[j]);
2909             fwrite(CRDLFa, 2, 1, fp_out );
2910             fprintf(fp_outC2, "%lu\n", TimesOverItem);
2911             linecounterRL++;
2912         }
2913         jPrevLine = j;
2914     }
2915 }
2916 // Railgunning ... ]
2917 }
2918 else
2919 { printf("IP_ELF: Input file too large, IPs remain unsorted!\n");
2920   return 1;
2921 }
2922 // FILE LOADING & RAILGUNNING ]
2923 */

```