```
// This is source of Leprechaun revision 16FIX: Leprechaun_x-leton.c, copyleft Sanmayce, 2012-Dec-16.
// How embarrassing! A stupid bug was fixed, namely one missed 'if ( REUSE == 0 ) {}' holding the TRAVERSE segment - this segment nullifies
   LEAF addresses thus making W/W unable to retraverse.

// This is source of Leprechaun revision 16: Leprechaun_x-leton.c, copyleft Sanmayce, 2012-Dec-13.
// The new feature is the ability to reuse the external hash-tree structure.
// The option is W/W similar to Z/Z. This way the latency i.e. the response time is <1s.

// This is source of Leprechaun revision 15FIXFIX+: Leprechaun_x-leton.c, copyleft Sanmayce, 2012-Dec-11.
// The new feature is the ability to command Leprechaun (from inside the list file with 2 metacommands) to enter/exit INSERT mode.
// This allows to control whether new (to current hash-tree structure) x-grams are to be counted [and] INSERTed.

// Usage:
// E:\_Gamera_r15_12348>type ON.txt
// Leprechaun says x-gram inserting disabled for next files: ON
//
// E:\_Gamera_r15_12348>type OFF.txt
// Leprechaun says x-gram inserting disabled for next files: OFF
//
// E:\_Gamera_r15_12348>dir Your_textual_folders/b/s/a-d>go.lst
// E:\_Gamera_r15_12348>copy go.lst+on.txt+_Gamera.tar.3.sorted.4andabove.lst MetaLep.lst /b
// E:\_Gamera_r15_12348>type MetaLep.lst
// E:\_Gamera_r15_12348\Your_textual_folders\Example.txt
// Leprechaun says x-gram inserting disabled for next files: ON
// _Gamera.tar.3.sorted.4andabove.txt
//
// E:\_Gamera_r15_12348>Leprechaun_x-leton_META_32bit_03_01p.exe MetaLep.lst MetaLep.3.wrd 1234567 Y

// All lines new to r15FIXFIX are with commented part //15FIXFIX+

/*
This is source of Leprechaun revision 15FIXFIX: Leprechaun_x-leton.c, copyleft Sanmayce, 2011-Dec-14. 2011-Mar-07: Fixed a small command line
    parsing bug.

The 15FIXFIX differs from 15fix with:
[a bug fixed(REALLY FIXED!)]: Fixed a nasty bug causing very restrictive way of forming x-grams.]
The 15fix differs from 15 with:
[a bug fixed: division by zero when finishing-starting time is under 1 second
Fixed a nasty bug causing very restrictive way of forming x-grams.]
The 15 differs from 14++++FIXFIX with:
[
Only some more stats at the end.
]
The 14++++FIXFIX differs from 14+++FIX with:
[
Bugs in LOG stats in r.14+++FIX:
Not nullified variables during passes - must be nullified.
Number of Trees(GREATER THE BETTER): 195,939
Number of LEAFs(littler THE BETTER) not counting ROOT LEAFs: 654,428
Total Attempts to Find/Put WORDs into B-trees order 3: 33,042,828
Highest Tree not counting ROOT Level i.e. CORONA Level(littler THE BETTER): 3
]
The 14+++FIX differs from 14+++ with:
[
1)Fixed occurrencies bug due to not NULLifying the field housing the occurrencies, a nasty thing: all the revisions 14??? were buggy, how
  stupid from my side, grrrr.
2)Ability to rip in passes:
#define HashChunkSizeInBITS 26 // Defines the number of passes. Should be smaller or equal to HashInBITS. If HashInBITS == HashChunkSizeInBITS
  then 2^(HashInBITS-HashChunkSizeInBITS)=2^0=1 passe(s).
]
The 14++ differs from 14+ with:
[
//Only one must be uncommented:
//#define singleton
//#define doubleton
//#define tripleton
//#define quadrupleton
//#define quintupleton
//#define sextupleton
//#define septupleton
//#define octupleton
//#define nonupleton
//#define decupleton
One     single,   singlet,   singleton
Two     double,   doublet,   doubleton
Three   triple,   triplet,   tripleton
Four    quadruple, quadruplet, quadrupleton
Five    quintuple, quintuplet, quintupleton
Six     sextuple,  sextuplet,  sextupleton
Seven   septuple,  septuplet,  septupleton
Eight   octuple,   octuplet,   octupleton
Nine    nonuple,   nonuplet,   nonupleton
Ten,    decuple,   decuplet,   decupleton
One     ace, single, singleton, unary, unit, unity
Two     binary, brace, couple, couplet, distich, deuce, double, doubleton, duad, duality, duet, duo, dyad, pair, snake eyes, span,
  twain, twosome, yoke
Three   deuce-ace, leash, set, tercet, ternary, ternion, terzetto, threesome, tierce, trey, triad, trine, trinity, trio, triplet,
  troika, hat-trick
```

Leprechaun_x-leton revision **16FIX**; Both Physical/INTERNAL and Virtual/EXTERNAL modes pure 64bit (32bit code along with 64bit) + Multi-Pass Mode + REUSE; Last update: 2012-Dec-16

```
4    Four     foursome, quadruplet, quatern, quaternary, quaternion, quaternity, quartet, tetrad
5    Five     cinque, fin, fivesome, pentad, quint, quintet, quintuplet
6    Six      half dozen, hexad, sestet, sextet, sextuplet, sise
7    Seven    heptad, septet, septuplet
8    Eight    octad, octave, octet, octonary, octuplet, ogdoad
Also, in addition to 'Y' and 'Z', 'y' and 'z' were added in order to be able to dump only n-grams without occurrencies.
]

A lazy approach is applied in order to add occurrences of each 4-gram:
- just reserve the last 4bytes in 'wrd' for counter as follows:
  'LongestLineInclusive' has to be greater than 31 (51 looks good enough) in order not to miss longer 4-grams like:
encourage_innovative_approaches_to
  char FourGram[LongestLineInclusive+1+4]; // 31bytes longest 4-gram + 1byte NULL + 4bytes COUNTER
- the laziness lies here:
  no need to make the four bytes to house the value 1 when a new 'wrd' is being inserted (either in step 1 or step 3) just add 1 at final
traverse dump,
  in step 1 when a 'wrd' is found then add 1 to the counter only if it is not 9,999,999 already (limitation enforced on counter).
- when dumping the format has to be:
  0,000,001\ta_b_c_d
  in order to sort the whole lines later with external Qsort and have easy screening for rare/wrong/useless 4-grams.

Comment/Uncomment accordingly in order to compile:
//#define _WIN32_ENVIRONMENT_
#define _POSIX_ENVIRONMENT_

Windows compile(uncomment #include <io.h> line, ignore warnings):
For Microsoft (R) 32-bit C/C++ Optimizing Compiler Version 13.10.3077 for 80x86 use:
cl /Ox /Wp64 /TcLeprechaun.c /FaLeprechaun

Windows compile(comment #include <io.h> line, ignore warnings):
For Intel(R) C++ Compiler Professional for applications running on IA-32, Version 11.1 use:
icl /Ox /Wp64 /TcLeprechaun.c /FaLeprechaun /w /QxHOST

Linux compile(ignore warnings):
gcc -D_FILE_OFFSET_BITS=64 -m64 -static -O3 -mtune=generic Leprechaun_quadrupleton.c -o Leprechaun_quadrupleton_r14_generic_64bits.elf

!!! For some reason a nasty bug (some UFO/wrong occurrences before phrases in the resultant file) occurs when 32bit (supposedly the opposite
of the expected) code is generated:
gcc -D_FILE_OFFSET_BITS=64 -m32 -static -O3 -mtune=generic Leprechaun_quadrupleton.c -o Leprechaun_quadrupleton_r14_generic_32bits.elf

[It's a little weird(Intel boosts the sort while falls behind in parsing, tested on T3400):]

Leprechaun_r13_7pluses_Microsoft_32-bit_16.00.30319.01.exe _vs_ Wikipedia_22,202,980_LATIN-Words:
Words per second performance: 1,679,585W/s
Time for making unsorted wordlist: 30 second(s)
Time for sorting unsorted wordlist: 25 second(s)

Leprechaun_r13_7pluses_Intel_IA-32_11.1.exe _vs_ Wikipedia_22,202,980_LATIN-Words:
Words per second performance: 1,603,240W/s
Time for making unsorted wordlist: 31 second(s)
Time for sorting unsorted wordlist: 19 second(s)

Due to my ignorance(calderas in my C knowledge): 64bit code cannot be generated, for now.
Any improvement is welcome.
Enjoy!
*/

// C:\WorkTemp\Leprechaun_r13++\Visual C++ Toolkit 2003\Leprechaun_r13++++++_C_EXE>cl /Ox /Wp64 /TcLeprechaun.c /FaLeprechaun
// Microsoft (R) 32-bit C/C++ Optimizing Compiler Version 13.10.3077 for 80x86
// Copyright (C) Microsoft Corporation 1984-2002. All rights reserved.
//
// Leprechaun.c
// Leprechaun.c(829) : warning C4312: 'type cast' : conversion from 'int' to 'string' of greater size
// Leprechaun.c(849) : warning C4312: 'type cast' : conversion from 'int' to 'string * ' of greater size
// Leprechaun.c(2048) : warning C4312: 'type cast' : conversion from 'int' to 'char *' of greater size
// Leprechaun.c(2063) : warning C4311: 'type cast' : pointer truncation from 'char *' to 'unsigned long'
// Leprechaun.c(2068) : warning C4311: 'type cast' : pointer truncation from 'char *' to 'unsigned long'
// Leprechaun.c(2371) : warning C4312: 'type cast' : conversion from 'unsigned long' to 'char *' of greater size
// Leprechaun.c(2570) : warning C4312: 'type cast' : conversion from 'unsigned long' to 'char *' of greater size
// Leprechaun.c(2626) : warning C4312: 'type cast' : conversion from 'unsigned long' to 'char *' of greater size
// Leprechaun.c(2657) : warning C4312: 'type cast' : conversion from 'unsigned long' to 'char *' of greater size
// Leprechaun.c(2663) : warning C4312: 'type cast' : conversion from 'unsigned long' to 'char *' of greater size
// Leprechaun.c(2668) : warning C4312: 'type cast' : conversion from 'unsigned long' to 'char *' of greater size
// Leprechaun.c(2696) : warning C4312: 'type cast' : conversion from 'unsigned long' to 'char *' of greater size
// Leprechaun.c(2729) : warning C4312: 'type cast' : conversion from 'unsigned long' to 'char *' of greater size
// Leprechaun.c(2743) : warning C4312: 'type cast' : conversion from 'unsigned long' to 'char *' of greater size
// Leprechaun.c(2755) : warning C4312: 'type cast' : conversion from 'unsigned long' to 'char *' of greater size
// Microsoft (R) Incremental Linker Version 7.10.3077
// Copyright (C) Microsoft Corporation.  All rights reserved.
//
// /out:Leprechaun.exe
// Leprechaun.obj
//
// C:\WorkTemp\Leprechaun_r13++\Visual C++ Toolkit 2003\Leprechaun_r13++++++_C_EXE>

/*
Below is the gain in 13++ and 13+++:

Words per second performance: 5,974,513W/s
```



```
Word count: 4,582,451,898 of them 9,177,221 distinct
Number Of Trees(GREATER THE BETTER): 2855919
Number Of Hash Collisions(Distinct WORDs - Number Of Trees): 6321302

Words per second performance: 6,329,353W/s
Word count: 4,582,451,898 of them 9,177,221 distinct
Number Of Trees(GREATER THE BETTER): 2958681
Number Of Hash Collisions(Distinct WORDs - Number Of Trees): 6218540

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.
Thanks to Fowler/Noll/Vo hash inventors.
*/

/*
Let's see the supplementary-clash on Intel Pentium T3400 Merom-1M 2166MHz:
Binary-Search-Trees vs B-Trees of order 3

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
Leprechaun(Fast Greedy Word-Ripper), revision 13++++++, written by Svalqyatchx.
Leprechaun: 'Oh, well, didn't you hear? Bigger is good, but jumbo is dear.'
Kaze: Let's see what a 3-way hash + 6,602,752 Binary-Search-Trees can give us,
      also the performance of a 3-way hash + 6,602,752 B-Trees of order 3.
Size of input file with files for Leprechauning: 27
Allocating memory 1863MB ... OK
Size of Input TEXTual file: 146,973,879
\; Word count: 12,561,874 of them 12,561,874 distinct; Done: 64/64
Bytes per second performance: 14,697,387B/s
Words per second performance: 1,256,187W/s
Flushing unsorted words ...
Time for making unsorted wordlist: 15 second(s)
Deallocated memory in MB: 1863
Allocated memory for words in MB: 141
Allocated memory for pointers-to-words in MB: 48
Sorting(with 'MultiKeyQuickSortX26Sort' by J. Bentley and R. Sedgewick) ...
Sort pass 26/26 ...
Flushing sorted words ...
Time for sorting unsorted wordlist: 14 second(s)
Leprechaun: Done.

[An excerpt of Leprechaun.LOG:]
Number Of Trees(GREATER THE BETTER): 2786806
Total Attempts to Find/Put WORDs into Binary-Search-Trees: 58,935,172
Total Number of LEAFs in Binary-Search-Trees(GREATER THE BETTER): 2,786,806

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
Leprechaun(Fast Greedy Word-Ripper), revision 13++++++, written by Svalqyatchx.
Leprechaun: 'Oh, well, didn't you hear? Bigger is good, but jumbo is dear.'
Kaze: Let's see what a 3-way hash + 6,602,752 Binary-Search-Trees can give us,
      also the performance of a 3-way hash + 6,602,752 B-Trees of order 3.
Size of input file with files for Leprechauning: 27
Allocating memory 1863MB ... OK
Size of Input TEXTual file: 146,973,879
\; Word count: 12,561,874 of them 12,561,874 distinct; Done: 64/64
Bytes per second performance: 24,495,646B/s
Words per second performance: 2,093,645W/s
Flushing unsorted words ...
Time for making unsorted wordlist: 12 second(s)
Deallocated memory in MB: 1863
Allocated memory for words in MB: 141
Allocated memory for pointers-to-words in MB: 48
Sorting(with 'MultiKeyQuickSortX26Sort' by J. Bentley and R. Sedgewick) ...
Sort pass 26/26 ...
Flushing sorted words ...
Time for sorting unsorted wordlist: 14 second(s)
Leprechaun: Done.

[An excerpt of Leprechaun.LOG:]
Number Of Trees(GREATER THE BETTER): 2786806
Total Attempts to Find/Put WORDs into B-trees order 3: 18,534,910

C:\WorkTemp\Leprechaun_r13++\Visual C++ Toolkit 2003\Leprechaun_step_1_PAIR-QUEST>type Leprechaun_vs_Wikipedia_en-WORDS.lst
wikipedia-en-html.tar.wrd

C:\WorkTemp\Leprechaun_r13++\Visual C++ Toolkit 2003\Leprechaun_step_1_PAIR-QUEST>dir Leprechaun_vs_Wikipedia_en-WORDS.*
 Volume in drive C is H320_Vol2
 Volume Serial Number is A094-FAE2

 Directory of C:\WorkTemp\Leprechaun_r13++\Visual C++ Toolkit 2003\Leprechaun_step_1_PAIR-QUEST

09/14/2010  06:04 AM                27 Leprechaun_vs_Wikipedia_en-WORDS.lst
09/15/2010  02:51 AM       146,973,879 Leprechaun_vs_Wikipedia_en-WORDS.wrd
               2 File(s)    146,973,906 bytes
               0 Dir(s)     965,787,648 bytes free

Conclusion:
```

```
18,534,910/12,561,874=1.475 Average Attempts to Find/Put WORDs into B-trees order 3, not bad at all.
*/

// To do: must learn how to align, at last.
/*
Matt Mahoney ZPAQ fragment:
  T *data;  // allocated memory
  int offset;
  ...
  offset=64-int((long)data&63);
  data=(T*)((char*)data+offset);  // adjust to 64 byte boundary

quicklz.c fragment:
#define QLZ_ALIGNMENT_PADD 8
unsigned char *scratch_aligned = (unsigned char *)scratch_compress + QLZ_ALIGNMENT_PADD - (((size_t)scratch_compress) % QLZ_ALIGNMENT_PADD);
size_t *buffersize = (size_t *)scratch_aligned;

minilzo.c fragment:
#define lzo_uintptr_t       unsigned long
#define PTR(a)              ((lzo_uintptr_t) (a))
#define PTR_LINEAR(a)       PTR(a)
#define PTR_ALIGNED_4(a)    ((PTR_LINEAR(a) & 3) == 0)
*/

//__declspec(align(64)) int BigArray[1024]; // Windows syntax
//or
//int BigArray[1024] __attribute__((aligned(64))); // Linux syntax

#if defined(_WIN32_ENVIRONMENT_)
__declspec(align(64))
#else
//__attribute__((aligned(64)));
#endif /* defined(_WIN32_ENVIRONMENT_)  */

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:
//; 792 : hash32 = FNV_32A_OP32(hash32, *(UINT*)p&0xFFFF);

typedef unsigned int UINT;
typedef unsigned int DWORD;

/*
Enter-the-BESTer or an alchemical clash of pairs of primes.

When an x-bit hash where x < 16 and is not a power of 2 is needed,
here comes 'FNV1A_Hash_4_OCTETS': a slightly tuned FNV1A hash for a huge(22,202,980) wordlist of latin-letters-words.

Two improvements for the generic(base) FNV1A hash:
- first, better speed: by reducing 'imul' instructions when string is 4++ chars
- second, better dispersion: by experimenting(superficially-lite test done, so far) with 'FNV1_32_PRIME'

Or more concretely:
- For FNV1_32_INIT = 2166136261
- Giving to 'FNV1_32_PRIME' all primes between 2 and 11987
- Shifting by 16bits instead of 13bits, when 8192 slots are used

C code:
typedef unsigned char u_int8_t;
typedef unsigned long u_int32_t;

#define FNV1_32_INIT ((u_int32_t)2166136261)
#define FNV1_32_PRIME ((u_int32_t)1607)

#define FNV_32A_OP(hash, octet) \
    (((u_int32_t)(hash) ^ (u_int8_t)(octet)) * FNV1_32_PRIME)

#define FNV_32A_OP32(hash, octet) \
    (((u_int32_t)(hash) ^ (u_int32_t)(octet)) * FNV1_32_PRIME)

0800 // Invoking: FNV1A_Hash_4_OCTETS(wrd, wrdlen>>2) // = 0,1,2,3,4,5,6,7 [1..31]
0801 int FNV1A_Hash_4_OCTETS(char *str, int wrdlen_QUADRUPLETS)
0802 {
0803 u_int32_t hash;
0804 char *p;
0805
0806 hash = FNV1_32_INIT;
0807 p=str;
0808
0809 // The goal of stage #1: to reduce number of 'imul's.
0810
0811 // Stage #1:
0812 for (; wrdlen_QUADRUPLETS != 0; --wrdlen_QUADRUPLETS) {
0813     hash = FNV_32A_OP32(hash, (unsigned long)*(long *)p); // mov edi, DWORD PTR [eax]
0814     p=p+4; // add eax, 4
0815 }
0816
0817 // Stage #2:
0818 for (; *p; ++p) {
```



```
0819     hash = FNV_32A_OP(hash, *p); // mov dl, BYTE PTR [eax]
0820 }
0821
0822   //return ((hash>>13) ^ hash) & 8191; // (((u_int32_t)1<<(x))-1) where x=13
0823   return ((hash>>16) ^ hash) & 8191; // 00..8191 i.e. 2^13=8192
0824 }

Assembler code:
_FNV1A_Hash_4_OCTETS PROC NEAR
; Line 812
mov  edx, DWORD PTR _wrdlen_QUADRUPLETS$[esp-4]
test edx, edx
mov  eax, DWORD PTR _str$[esp-4]
push esi
mov  esi, DWORD PTR _FNV1_32_PRIME
mov  ecx, -2128831035
je   SHORT $L1612
push edi
npad 7
$L1610:
; Line 813
mov  edi, DWORD PTR [eax]
xor  edi, ecx
imul edi, esi
; Line 814
add  eax, 4
dec  edx
mov  ecx, edi
jne  SHORT $L1610
pop  edi
$L1612:
; Line 818
mov  dl, BYTE PTR [eax]
test dl, dl
je   SHORT $L1619
$L1617:
; Line 819
movzx        edx, dl
xor  edx, ecx
imul edx, esi
inc  eax
mov  ecx, edx
mov  dl, BYTE PTR [eax]
test dl, dl
jne  SHORT $L1617
$L1619:
; Line 823
mov  eax, ecx
shr  eax, 16
xor  eax, ecx
and  eax, 8191
pop  esi
; Line 824
ret  0
_FNV1A_Hash_4_OCTETS ENDP


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.

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'.

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.
This is critical for speed performance e.g. when 30,974,750,142 words, the case of wikipedia-en-html.tar, must be hashed.
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.
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.

Between 1..1166136247 there are 58,834,113 primes (inclusive).
Between 1..16777619 there are 1,077,891 primes (inclusive).
Or 58834113*1077891 = 63,416,760,895,683 pairs or 2,010,932 years needed at one-pair-per-second rate.

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.
Since the magnum corpus of words is static-enough, THE pair is worthy to be found.

It doesn't take a think-tank to see the superiority of FNV, Fowler/Noll/Vo did reveal a thing of beauty.

Performance of 'FNV1A_Hash_4_OCTETS': 10236 words/clock or 105 MB/s|3,549,448 used slots (best)

CASE #1: with 'if (strlen(backup[j]) != 0)' before each execution
Performance of 'KuxHash3plus' aka '2in1': 8076 words/clock or 82 MB/s|3,410,463 used slots (worst)
Performance of 'FNV1A_Hash': 8079 words/clock or 83 MB/s|3,527,916 used slots
Performance of 'FNV1A_Hash_SHIFTless_XORless': 8109 words/clock or 83 MB/s|3,540,323 used slots

CASE #2: without 'if (strlen(backup[j]) != 0)' before each execution
Performance of 'KuxHash3plus' aka '2in1': 11673 words/clock or 119 MB/s|3,410,463 used slots (worst)
Performance of 'FNV1A_Hash': 11558 words/clock or 118 MB/s|3,527,916 used slots
Performance of 'FNV1A_Hash_SHIFTless_XORless': 11570 words/clock or 118 MB/s|3,540,323 used slots
```

```
0426
0427 Note:
0428 The 'strlen' overhead(CASE #1) is necessary due to priorly(before hash invocation) needed len-of-string for 'FNV1A_Hash_4_OCTETS'.
0429 Almost always, that is the case, since parsing of incoming text must know length of words/lines/files.
0430 In case of not knowing this length: ((119-105)/105)*100% = 13% degradation is unacceptable.
0431 The 'strlen' is an awful brake.
0432 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.
0433 */
0434
0435 /*
0436 FNV1_32_PRIME: //?: 16777619
0437
0438 Above Binary-Search-Tree with MaxPEAK = 61 has NODEs = 61 and LEAFs = 1
0439 Words per second performance: 1,046,822W/s
0440 Input File with a list of TEXTual Files: Leprechaun_vs_Wikipedia_en-WORDS.lst
0441 Size of all TEXTual Files: 146,973,879
0442 Word count: 12,561,874 of them 12,561,874 distinct
0443 Number Of Trees(GREATER THE BETTER): 2775839
0444
0445 Above Binary-Search-Tree with MaxPEAK = 39 has NODEs = 72 and LEAFs = 15
0446 Words per second performance: 1,356,588W/s
0447 Input File with a list of TEXTual Files: Leprechaun_vs_Wikipedia_LATIN-WORDS.lst
0448 Size of all TEXTual Files: 415,982,896
0449 Word count: 35,271,297 of them 22,202,980 distinct
0450 Number Of Trees(GREATER THE BETTER): 3539690
0451
0452 FNV1_32_PRIME: //3549448: 1607
0453
0454 Above Binary-Search-Tree with MaxPEAK = 61 has NODEs = 61 and LEAFs = 1
0455 Words per second performance: 1,046,822W/s
0456 Input File with a list of TEXTual Files: Leprechaun_vs_Wikipedia_en-WORDS.lst
0457 Size of all TEXTual Files: 146,973,879
0458 Word count: 12,561,874 of them 12,561,874 distinct
0459 Number Of Trees(GREATER THE BETTER): 2783970
0460
0461 Above Binary-Search-Tree with MaxPEAK = 38 has NODEs = 50 and LEAFs = 11
0462 Words per second performance: 1,410,851W/s
0463 Input File with a list of TEXTual Files: Leprechaun_vs_Wikipedia_LATIN-WORDS.lst
0464 Size of all TEXTual Files: 415,982,896
0465 Word count: 35,271,297 of them 22,202,980 distinct
0466 Number Of Trees(GREATER THE BETTER): 3549395
0467
0468 FNV1_32_PRIME: //3550132: 175757909
0469
0470 Above Binary-Search-Tree with MaxPEAK = 60 has NODEs = 60 and LEAFs = 1
0471 Words per second performance: 966,298W/s
0472 Input File with a list of TEXTual Files: Leprechaun_vs_Wikipedia_en-WORDS.lst
0473 Size of all TEXTual Files: 146,973,879
0474 Word count: 12,561,874 of them 12,561,874 distinct
0475 Number Of Trees(GREATER THE BETTER): 2784479
0476
0477 Above Binary-Search-Tree with MaxPEAK = 39 has NODEs = 64 and LEAFs = 12
0478 Words per second performance: 1,410,851W/s
0479 Input File with a list of TEXTual Files: Leprechaun_vs_Wikipedia_LATIN-WORDS.lst
0480 Size of all TEXTual Files: 415,982,896
0481 Word count: 35,271,297 of them 22,202,980 distinct
0482 Number Of Trees(GREATER THE BETTER): 3550115
0483
0484 FNV1_32_PRIME: //3550687: 201887489
0485
0486 Above Binary-Search-Tree with MaxPEAK = 60 has NODEs = 60 and LEAFs = 1
0487 Words per second performance: 966,298W/s
0488 Input File with a list of TEXTual Files: Leprechaun_vs_Wikipedia_en-WORDS.lst
0489 Size of all TEXTual Files: 146,973,879
0490 Word count: 12,561,874 of them 12,561,874 distinct
0491 Number Of Trees(GREATER THE BETTER): 2784377
0492
0493 Above Binary-Search-Tree with MaxPEAK = 40 has NODEs = 55 and LEAFs = 11
0494 Words per second performance: 1,356,588W/s
0495 Input File with a list of TEXTual Files: Leprechaun_vs_Wikipedia_LATIN-WORDS.lst
0496 Size of all TEXTual Files: 415,982,896
0497 Word count: 35,271,297 of them 22,202,980 distinct
0498 Number Of Trees(GREATER THE BETTER): 3550528
0499
0500 FNV1_32_PRIME: //3550733: 172783361
0501
0502 Above Binary-Search-Tree with MaxPEAK = 59 has NODEs = 59 and LEAFs = 1
0503 Words per second performance: 1,046,822W/s
0504 Input File with a list of TEXTual Files: Leprechaun_vs_Wikipedia_en-WORDS.lst
0505 Size of all TEXTual Files: 146,973,879
0506 Word count: 12,561,874 of them 12,561,874 distinct
0507 Number Of Trees(GREATER THE BETTER): 2786362
0508
0509 Above Binary-Search-Tree with MaxPEAK = 38 has NODEs = 70 and LEAFs = 17
0510 Words per second performance: 1,410,851W/s
0511 Input File with a list of TEXTual Files: Leprechaun_vs_Wikipedia_LATIN-WORDS.lst
0512 Size of all TEXTual Files: 415,982,896
```



```
0513 Word count: 35,271,297 of them 22,202,980 distinct
0514 Number Of Trees(GREATER THE BETTER): 3550746
0515
0516 FNV1_32_PRIME: //3550929: 204312319
0517
0518 Above Binary-Search-Tree with MaxPEAK = 61 has NODEs = 61 and LEAFs = 1
0519 Words per second performance: 966,298W/s
0520 Input File with a list of TEXTual Files: Leprechaun_vs_Wikipedia_en-WORDS.lst
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): 2785581
0524
0525 Above Binary-Search-Tree with MaxPEAK = 37 has NODEs = 55 and LEAFs = 12
0526 Words per second performance: 1,356,588W/s
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): 3550886
0531
0532 Leprechaun_Microsoft.exe: FNV1_32_PRIME: //3551736: 107712257
0533
0534 Above Binary-Search-Tree with MaxPEAK = 61 has NODEs = 61 and LEAFs = 1
0535 Words per second performance: 1,046,822W/s
0536 Input File with a list of TEXTual Files: Leprechaun_vs_Wikipedia_en-WORDS.lst
0537 Size of all TEXTual Files: 146,973,879
0538 Word count: 12,561,874 of them 12,561,874 distinct
0539 Number Of Trees(GREATER THE BETTER): 2786515
0540
0541 Above Binary-Search-Tree with MaxPEAK = 36 has NODEs = 64 and LEAFs = 15
0542 Words per second performance: 1,356,588W/s
0543 Input File with a list of TEXTual Files: Leprechaun_vs_Wikipedia_LATIN-WORDS.lst
0544 Size of all TEXTual Files: 415,982,896
0545 Word count: 35,271,297 of them 22,202,980 distinct
0546 Number Of Trees(GREATER THE BETTER): 3551744
0547
0548 Leprechaun_Intel.exe: FNV1_32_PRIME: //3551736: 107712257
0549
0550 Above Binary-Search-Tree with MaxPEAK = 61 has NODEs = 61 and LEAFs = 1
0551 Words per second performance: 1,256,187W/s
0552 Input File with a list of TEXTual Files: Leprechaun_vs_Wikipedia_en-WORDS.lst
0553 Size of all TEXTual Files: 146,973,879
0554 Word count: 12,561,874 of them 12,561,874 distinct
0555 Number Of Trees(GREATER THE BETTER): 2786515
0556
0557 Above Binary-Search-Tree with MaxPEAK = 36 has NODEs = 64 and LEAFs = 15
0558 Words per second performance: 1,603,240W/s
0559 Input File with a list of TEXTual Files: Leprechaun_vs_Wikipedia_LATIN-WORDS.lst
0560 Size of all TEXTual Files: 415,982,896
0561 Word count: 35,271,297 of them 22,202,980 distinct
0562 Number Of Trees(GREATER THE BETTER): 3551744
0563
0564 Wow: 1,603,240W/s vs 1,356,588W/s respectively Leprechaun_Intel.exe vs Leprechaun_Microsoft.exe, i.e. 18% betterment, no joke!
0565
0566 Alchemical search for best PRIME-PAIR revision uses next line:
0567 Slot = FNV1A_Hash_4_OCTETS(wrd, wrdlen>>2)<<2; //13++++
0568 This revision uses next lines:
0569 if (wrdlen<=19) // 4x4+3=19 i.e. last contains 7 clashes
0570               Slot = FNV1A_Hash_Granularity(wrd, wrdlen>>2, 2)<<2; //13+++++
0571 else            // 2x8+4=20 i.e. first contains 6 clashes
0572               Slot = FNV1A_Hash_Granularity(wrd, wrdlen>>3, 3)<<2; //13+++++
0573
0574 ! 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.
0575
0576 Leprechaun.exe: FNV1_32_PRIME: //3551961: 428904191
0577
0578 Above Binary-Search-Tree with MaxPEAK = 60 has NODEs = 60 and LEAFs = 1
0579 Words per second performance: 966,298W/s
0580 Input File with a list of TEXTual Files: Leprechaun_vs_Wikipedia_en-WORDS.lst
0581 Size of all TEXTual Files: 146,973,879
0582 Word count: 12,561,874 of them 12,561,874 distinct
0583 Number Of Trees(GREATER THE BETTER): 2786383
0584
0585 Above Binary-Search-Tree with MaxPEAK = 39 has NODEs = 71 and LEAFs = 16
0586 Words per second performance: 1,410,851W/s
0587 Input File with a list of TEXTual Files: Leprechaun_vs_Wikipedia_LATIN-WORDS.lst
0588 Size of all TEXTual Files: 415,982,896
0589 Word count: 35,271,297 of them 22,202,980 distinct
0590 Number Of Trees(GREATER THE BETTER): 3551503
0591
0592 Leprechaun.exe: FNV1_32_PRIME: //3552103: 588411137
0593
0594 Above Binary-Search-Tree with MaxPEAK = 6 has NODEs = 6 and LEAFs = 1
0595 Size of all TEXTual Files: 4,067,439
0596 Word count: 358,798 of them 351,116 distinct
0597 Number Of Trees(GREATER THE BETTER): 310622
0598 Total Number of LEAFs in Binary-Search-Trees(GREATER THE BETTER): 310,622
0599
```

```
0600 Above Binary-Search-Tree with MaxPEAK = 60 has NODEs = 60 and LEAFs = 1
0601 Size of all TEXTual Files: 146,973,879
0602 Word count: 12,561,874 of them 12,561,874 distinct
0603 Number Of Trees(GREATER THE BETTER): 2786485
0604 Total Number of LEAFs in Binary-Search-Trees(GREATER THE BETTER): 2,786,485
0605
0606 Above Binary-Search-Tree with MaxPEAK = 39 has NODEs = 62 and LEAFs = 15
0607 Size of all TEXTual Files: 415,982,896
0608 Word count: 35,271,297 of them 22,202,980 distinct
0609 Number Of Trees(GREATER THE BETTER): 3551956
0610 Total Number of LEAFs in Binary-Search-Trees(GREATER THE BETTER): 8,072,131
0611
0612 Leprechaun.exe: FNV1_32_PRIME: //3552039: 602173697 !!!GOODEST so far!!!
0613
0614 Above Binary-Search-Tree with MaxPEAK = 6 has NODEs = 6 and LEAFs = 1
0615 Size of all TEXTual Files: 4,067,439
0616 Word count: 358,798 of them 351,116 distinct
0617 Number Of Trees(GREATER THE BETTER): 310948
0618 Total Number of LEAFs in Binary-Search-Trees(GREATER THE BETTER): 310,948
0619
0620 Above Binary-Search-Tree with MaxPEAK = 63 has NODEs = 63 and LEAFs = 1
0621 Size of all TEXTual Files: 146,973,879
0622 Word count: 12,561,874 of them 12,561,874 distinct
0623 Number Of Trees(GREATER THE BETTER): 2786806
0624 Total Number of LEAFs in Binary-Search-Trees(GREATER THE BETTER): 2,786,806
0625
0626 Above Binary-Search-Tree with MaxPEAK = 36 has NODEs = 52 and LEAFs = 9
0627 Input File with a list of TEXTual Files: Leprechaun_vs_Wikipedia_LATIN-WORDS.lst
0628 Size of all TEXTual Files: 415,982,896
0629 Word count: 35,271,297 of them 22,202,980 distinct
0630 Number Of Trees(GREATER THE BETTER): 3552296
0631 Total Number of LEAFs in Binary-Search-Trees(GREATER THE BETTER): 8,072,899
0632
0633 Between 1 and 602392027 at step 100 following FNV1_32_PRIMEs(for FNV1_32_INIT=2166136261) give(FNV1A_Hash_4_OCTETS) dispersion:
0634 3550022: 423779327
0635 3550028: 513793537
0636 3550053: 434840321
0637 3550067: 437062229
0638 3550080: 420344321
0639 3550090: 304777471
0640 3550097: 496547839
0641 3550129: 390809599
0642 3550132: 175757909
0643 3550163: 353712127
0644 3550231: 334434817
0645 3550237: 272789761
0646 3550247: 590341121
0647 3550255: 358814207
0648 3550277: 437182721
0649 3550326: 521795327
0650 3550347: 311867393
0651 3550447: 456137729
0652 3550458: 418208767
0653 3550516: 602048767
0654 3550525: 513597697
0655 3550526: 347283199
0656 3550528: 598773503
0657 3550592: 598139137
0658 3550598: 242448127
0659 3550611: 571481087
0660 3550628: 457012993
0661 3550664: 482822143
0662 3550666: 249098753
0663 3550687: 201887489
0664 3550702: 489976063
0665 3550710: 272961023
0666 3550733: 172783361
0667 3550734: 431562497
0668 3550929: 204312319
0669 3550984: 562853633
0670 3550991: 551362303
0671 3551359: 332820737
0672 3551484: 354126079
0673 3551514: 407138561
0674 3551523: 442058753
0675 3551701: 449230849
0676 3551736: 107712257
0677 3551961: 428904191
0678 3552039: 602173697
0679 3552103: 588411137
0680 */
0681
0682 // Windows: ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
0683 //  _CRTIMP size_t __cdecl fread(void *, size_t, size_t, FILE *);
0684 //  _CRTIMP size_t __cdecl fwrite(const void *, size_t, size_t, FILE *);
0685 //  _CRTIMP int __cdecl fgetpos(FILE *, fpos_t *);
0686 //  _CRTIMP int __cdecl fsetpos(FILE *, const fpos_t *);
0687
```



```
0688 //  _CRTIMP __int64 __cdecl _lseeki64(int, __int64, int);
0689 //  _CRTIMP __int64 __cdecl _telli64(int);
0690 //  _CRTIMP __int64 __cdecl _filelengthi64(int);
0691 //  above 3 are in 'io.h'
0692
0693 //  _CRTIMP int __cdecl fseek(FILE *, long, int);
0694 //  _CRTIMP long __cdecl ftell(FILE *);
0695 //  _CRTIMP int __cdecl fclose(FILE *);
0696
0697 //  #ifndef _SIZE_T_DEFINED
0698 //  #ifdef  _WIN64
0699 //  typedef unsigned __int64    size_t;
0700 //  #else
0701 //  typedef _W64 unsigned int   size_t;
0702 //  #endif
0703 //  #define _SIZE_T_DEFINED
0704 //  #endif
0705
0706 //  typedef __int64 fpos_t;
0707
0708 // Linux: ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
0709 //  size_t fread (void *data, size_t size, size_t count, FILE *stream)
0710 //  size_t fwrite (const void *data, size_t size, size_t count, FILE *stream)
0711 //  int fgetpos (FILE *stream, fpos_t *position)
0712 //  int fsetpos (FILE *stream, const fpos_t *position)
0713
0714 //  FILE * fopen64 (const char *filename, const char *opentype)
0715 //  int fseeko64 (FILE *stream, off64_t offset, int whence)
0716 //  off64_t ftello64 (FILE *stream)
0717 //  int fclose (FILE *stream)
0718
0719 //  off_t lseek (int filedes, off_t offset, int whence)
0720 //  above 1 is in 'unistd.h'
0721
0722
0723 // ============ MUST work both for Windows and Linux =============
0724 //Only one must be uncommented:
0725 #define _WIN32_ENVIRONMENT_
0726 //#define _POSIX_ENVIRONMENT_
0727
0728 //Only one must be uncommented:
0729 //#define singleton
0730 //#define doubleton
0731 #define tripleton
0732 //#define quadrupleton
0733 //#define quintupleton
0734 //#define sextupleton
0735 //#define septupleton
0736 //#define octupleton
0737 //#define nonupleton
0738 //#define decupleton
0739
0740 #ifdef singleton
0741 #define _ngram_ 1
0742 #endif
0743 #ifdef doubleton
0744 #define _ngram_ 2
0745 #endif
0746 #ifdef tripleton
0747 #define _ngram_ 3
0748 #endif
0749 #ifdef quadrupleton
0750 #define _ngram_ 4
0751 #endif
0752 #ifdef quintupleton
0753 #define _ngram_ 5
0754 #endif
0755 #ifdef sextupleton
0756 #define _ngram_ 6
0757 #endif
0758 #ifdef septupleton
0759 #define _ngram_ 7
0760 #endif
0761 #ifdef octupleton
0762 #define _ngram_ 8
0763 #endif
0764 #ifdef nonupleton
0765 #define _ngram_ 9
0766 #endif
0767 #ifdef decupleton
0768 #define _ngram_ 10
0769 #endif
0770
0771 #ifndef NULL
0772 #ifdef __cplusplus
0773 #define NULL 0
0774 #else
0775 #define NULL ((void*)0)
```

```
0776 #endif
0777 #endif
0778
0779 #define HashInBITS 24 // default 26 i.e. 2^26 i.e. 64MS(Mega Slots); slots contain 8bytes pointers or 512MB, because many netbooks have 512MB free (1GB in total)!
0780 #define HashChunkSizeInBITS 24 // Defines the number of passes. Should be smaller or equal to HashInBITS. If HashInBITS == HashChunkSizeInBITS then 2^(HashInBITS-HashChunkSizeInBITS)=2^0=1 pass(es).
0781 /*
0782 Tests done on super-speed-ramdisk 1800MB:
0783 Leprechaun_quadrupleton rev. 14+ in fact differs from r.14 only with optimized(LEAFwise) fragment 1] and 2]. Fragmet 3] and dump are not still optimized. The goal is to track how this partial tweak will affect 64KS(Kilo Slots) or 512KB hash or 1000 times smaller hash variant.
0784
0785 [Variant (HashInBITS 26 - 0) with 512MB hash:]
0786
0787 Leprechaun_quadrupleton (Fast Greedy Phrase-Ripper), rev. 14+, written by Svalqyatchx.
0788 Purpose: Rips all distinct 4-grams (4-word phrases) with length 12..51 chars from incoming texts.
0789 Feature1: In this revision 512MB 1-way hash is used which results in 67,108,864 external B-Trees of order 3.
0790 Feature2: The bottleneck is seek-time, if the external memory has latency 100+microseconds then look further.
0791 Size of input file with files for Leprechauning: 19
0792 Allocating HASH memory 536,870,977 bytes ... OK
0793 Allocating/ZEROing 1,292,478,478 bytes swap file ... OK
0794 Size of Input TEXTual file: 206,908,949
0795 |; 0,065,139P/s; Phrase count: 18,760,213 of them 10,165,640 distinct; Done: 64/64
0796 Bytes per second performance: 718,433B/s
0797 Phrases per second performance: 65,139P/s
0798 Time for putting phrases into trees: 288 second(s)
0799 Flushing UNsorted phrases: 100%; Shaking trees performance: 0,014,439P/s
0800 Time for shaking phrases from trees: 704 second(s)
0801 Dump LEAFwise also [
0802 Bytes per second performance: 736,330B/s
0803 Phrases per second performance: 66,762P/s
0804 Time for putting phrases into trees: 281 second(s)
0805 Flushing UNsorted phrases: 100%; Shaking trees performance: 0,023,807P/s
0806 Time for shaking phrases from trees: 427 second(s)
0807 Dump LEAFwise also ]
0808 Leprechaun: Done.
0809
0810 Leprechaun report:
0811 Number Of Hash Collisions(Distinct WORDs - Number Of Trees): 731,746
0812 Number Of Trees(GREATER THE BETTER): 9,433,894
0813 Number Of LEAFs(littler THE BETTER) not counting ROOT LEAFs: 69,623
0814 Highest Tree not counting ROOT Level i.e. CORONA levels(littler THE BETTER): 1
0815 Used value for third parameter in KB: 1,262,186
0816 Use next time as third parameter: 1,262,186
0817 Total Attempts to Find/Put WORDs into B-trees order 3: 365,283
0818
0819 [Variant (HashInBITS 26 - 10) with 512KB hash:]
0820
0821 Leprechaun_quadrupleton (Fast Greedy Phrase-Ripper), rev. 14+, written by Svalqyatchx.
0822 Purpose: Rips all distinct 4-grams (4-word phrases) with length 12..51 chars from incoming texts.
0823 Feature1: In this revision 512MB 1-way hash is used which results in 67,108,864 external B-Trees of order 3.
0824 Feature2: The bottleneck is seek-time, if the external memory has latency 100+microseconds then look further.
0825 Size of input file with files for Leprechauning: 19
0826 Allocating HASH memory 524,353 bytes ... OK
0827 Allocating/ZEROing 1,292,478,478 bytes swap file ... OK
0828 Size of Input TEXTual file: 206,908,949
0829 |; 0,014,331P/s; Phrase count: 18,760,213 of them 10,165,640 distinct; Done: 64/64
0830 Bytes per second performance: 158,066B/s
0831 Phrases per second performance: 14,331P/s
0832 Time for putting phrases into trees: 1309 second(s)
0833 Flushing UNsorted phrases: 100%; Shaking trees performance: 0,019,739P/s
0834 Time for shaking phrases from trees: 515 second(s)
0835 Dump LEAFwise also [
0836 Bytes per second performance: 158,429B/s
0837 Phrases per second performance: 14,364P/s
0838 Time for putting phrases into trees: 1306 second(s)
0839 Flushing UNsorted phrases: 100%; Shaking trees performance: 0,041,492P/s
0840 Time for shaking phrases from trees: 245 second(s)
0841 Dump LEAFwise also ]
0842 Dump & Insert LEAFwise also [
0843 Bytes per second performance: 174,459B/s
0844 Phrases per second performance: 15,818P/s
0845 Time for putting phrases into trees: 1186 second(s)
0846 Flushing UNsorted phrases: 100%; Shaking trees performance: 0,041,323P/s
0847 Time for shaking phrases from trees: 246 second(s)
0848 Dump & Insert LEAFwise also ]
0849 Leprechaun: Done.
0850
0851 Leprechaun report:
0852 Number Of Hash Collisions(Distinct WORDs - Number Of Trees): 10,100,104
0853 Number Of Trees(GREATER THE BETTER): 65,536
0854 Number Of LEAFs(littler THE BETTER) not counting ROOT LEAFs: 7,522,788
0855 Highest Tree not counting ROOT Level i.e. CORONA levels(littler THE BETTER): 6
0856 Used value for third parameter in KB: 1,262,186
0857 Use next time as third parameter: 1,007,825
0858 Total Attempts to Find/Put WORDs into B-trees order 3: 84,868,241
0859
0860 [Variant (HashInBITS 26 - 20) with 512 hash:]
```



```
0861
0862 Leprechaun_quadrupleton (Fast Greedy Phrase-Ripper), rev. 14+, written by Svalqyatchx.
0863 Purpose: Rips all distinct 4-grams (4-word phrases) with length 12..51 chars from incoming texts.
0864 Feature1: In this revision 512MB 1-way hash is used which results in 67,108,864 external B-Trees of order 3.
0865 Feature2: The bottleneck is seek-time, if the external memory has latency 100+microseconds then look further.
0866 Size of input file with files for Leprechauning: 19
0867 Allocating HASH memory 577 bytes ... OK
0868 Allocating/ZEROing 1,292,478,478 bytes swap file ... OK
0869 Size of Input TEXTual file: 206,908,949
0870 |; 0,007,717P/s; Phrase count: 18,760,213 of them 10,165,640 distinct; Done: 64/64
0871 Bytes per second performance: 85,112B/s
0872 Phrases per second performance: 7,717P/s
0873 Time for putting phrases into trees: 2431 second(s)
0874 Flushing UNsorted phrases: 100%; Shaking trees performance: 0,019,777P/s
0875 Time for shaking phrases from trees: 514 second(s)
0876 Leprechaun: Done.
0877
0878 Leprechaun report:
0879 Number Of Hash Collisions(Distinct WORDs - Number Of Trees): 10,165,576
0880 Number Of Trees(GREATER THE BETTER): 64
0881 Number Of LEAFs(littler THE BETTER) not counting ROOT LEAFs: 7,592,585
0882 Highest Tree not counting ROOT Level i.e. CORONA levels(littler THE BETTER): 14
0883 Used value for third parameter in KB: 1,262,186
0884 Use next time as third parameter: 1,008,399
0885 Total Attempts to Find/Put WORDs into B-trees order 3: 271,393,689
0886
0887 r.14++ physical memory test [Variant (HashInBITS 26 - 0) with 512MB hash:]:
0888 D:\_KAZE_new-stuff\Leprechaun_quadrupleton_r14++_64bit_Physical-n-Virtual>OSHO-TEST_INTERNAL.BAT
0889 Leprechaun_quadrupleton (Fast-In-Future Greedy Phrase-Ripper), rev. 14++, written by Svalqyatchx.
0890 Purpose: Rips all distinct 4-grams (4-word phrases) with length 12..51 chars from incoming texts.
0891 Feature1: In this revision 512MB 1-way hash is used which results in 67,108,864 external B-Trees of order 3.
0892 Feature2: If the external memory has latency 99+microseconds then !(look no further), IOPS(seek-time) rules.
0893 Size of input file with files for Leprechauning: 19
0894 Allocating HASH memory 536,870,977 bytes ... OK
0895 Allocating memory 1233MB ... OK
0896 Size of Input TEXTual file: 206,908,949
0897 |; 1,042,234P/s; Phrase count: 18,760,213 of them 10,165,640 distinct; Done: 64/64
0898 Bytes per second performance: 11,494,941B/s
0899 Phrases per second performance: 1,042,234P/s
0900 Time for putting phrases into trees: 18 second(s)
0901 Flushing UNsorted phrases: 100%; Shaking trees performance: 0,597,978P/s
0902 Time for shaking phrases from trees: 17 second(s)
0903 Leprechaun: Done.
0904
0905 D:\_KAZE_new-stuff\Leprechaun_quadrupleton_r14++_64bit_Physical-n-Virtual>type Leprechaun.LOG
0906 Leprechaun report:
0907 Number Of Hash Collisions(Distinct WORDs - Number Of Trees): 731,746
0908 Number Of Trees(GREATER THE BETTER): 9,433,894
0909 Number Of LEAFs(littler THE BETTER) not counting ROOT LEAFs: 69,623
0910 Highest Tree not counting ROOT Level i.e. CORONA levels(littler THE BETTER): 1
0911 Used value for third parameter in KB: 1,262,186
0912 Use next time as third parameter: 1,262,186
0913 Total Attempts to Find/Put WORDs into B-trees order 3: 365,283
0914
0915 D:\_KAZE_new-stuff\Leprechaun_quadrupleton_r14++_64bit_Physical-n-Virtual>
0916 */
0917
0918 // To do #1: Put this 31 in MAXwl: 'int MAXwl = 31;'
0919 // To do #2: No need of flushing unsorted words to file: make backup[] array
0920 //           instead of writing. And mostly sort 26 times!
0921 // HEAVY BUG in r.7: unsigned long Hill(unsigned long n)
0922 //                   is NOT identical with
0923 //                   unsigned long GRMBLhill[32]; // 00 not used, only 01..31
0924 //                   BECAUSE DUMBEST DUMB Array GRMBLhill expects 'int' not
0925 //                   'unsigned long' !!!
0926
0927 #include <stdio.h>
0928 #include <ctype.h>
0929 #include <time.h>
0930 #if defined(_WIN32_ENVIRONMENT_)
0931 #include <io.h> // needed for Windows' 'lseeki64' and 'telli64'
0932 //Above line must be commented in order to compile with Intel C compiler: an error "can't find io.h" occurs.
0933 #else
0934 #endif /* defined(_WIN32_ENVIRONMENT_)  */
0935
0936 typedef unsigned char char_t;
0937 typedef char_t *string;
0938
0939 clock_t clocks1, clocks2;
0940 int Bozan;
0941
0942 typedef unsigned char u_int8_t; //FNV only
0943 typedef unsigned long u_int32_t; //FNV only
0944 typedef unsigned long long u_int64_t; //FNV only
0945
0946 // SINHA fragment[
0947
0948 #define swapKAZE(a, b) { t = *(a); *(a) = *(b); *(b) = t; }
```

```
0949
0950 static void InsertSortKAZE(string *a, int n, int d) //void inssort(unsigned char **a, int n, int d)
0951 {   string *pi, *pj, s, t;                         //unsigned char **pi, **pj, *s, *t;
0952     for (pi = a + 1; --n > 0; pi++)
0953         for (pj = pi; pj > a; pj--) {
0954             /* Inline strcmp: break if *(pj-1) <= *pj */
0955             for (s=*(pj-1)+d, t=*pj+d; *s==*t && *s!=0; s++, t++)
0956                 ;
0957             if (*s <= *t)
0958                 break;
0959             swapKAZE(pj, pj-1);
0960     }
0961 }
0962
0963 //int cmpit(unsigned char **h1, unsigned char **h2)
0964 //{
0965 //    return( scmp(*h1, *h2) );
0966 //}
0967
0968 //int scmp( unsigned char *s1, unsigned char *s2 )
0969 //{
0970 //    while( *s1 != '\0' && *s1 == *s2 )
0971 //    {
0972 //        s1++;
0973 //        s2++;
0974 //    }
0975 //    return( *s1-*s2 );
0976 //}
0977
0978 //static void simplesort(string a[], int n, int b)
0979 //{
0980 //   int i, j;
0981 //   string tmp;
0982 //
0983 //   for (i = 1; i < n; i++)
0984 //      for (j = i; j > 0 && scmp(a[j-1]+b, a[j]+b) > 0; j--)
0985 //         { tmp = a[j]; a[j] = a[j-1]; a[j-1] = tmp; }
0986 //}
0987
0988 // SINHA fragment]
0989
0990 // mkqsort.c BEGIN ************************************************************
0991 /*
0992    Multikey quicksort, a radix sort algorithm for arrays of character
0993    strings by Bentley and Sedgewick.
0994
0995    J. Bentley and R. Sedgewick. Fast algorithms for sorting and
0996    searching strings. In Proceedings of 8th Annual ACM-SIAM Symposium
0997    on Discrete Algorithms, 1997.
0998
0999    http://www.CS.Princeton.EDU/~rs/strings/index.html
1000
1001    The code presented in this file has been tested with care but is
1002    not guaranteed for any purpose. The writer does not offer any
1003    warranties nor does he accept any liabilities with respect to
1004    the code.
1005
1006    Ranjan Sinha, 1 jan 2003.
1007
1008    School of Computer Science and Information Technology,
1009    RMIT University, Melbourne, Australia
1010    rsinha@cs.rmit.edu.au
1011
1012 */
1013
1014 //#include "sortstring.h"
1015
1016 /* MULTIKEY QUICKSORT */
1017
1018 #ifndef min
1019 #define min(a, b) ((a)<=(b) ? (a) : (b))
1020 #endif
1021
1022
1023 // ------------------------------ BTREE [
1024 #define false -1
1025 #define true 0
1026
1027 struct nodeBTREE {
1028 int data;
1029 struct nodeBTREE* left;
1030 struct nodeBTREE* right;
1031 };
1032
1033 // ------------------------------ BTREE ]
1034
1035
1036 /* ssort2 -- Faster Version of Multikey Quicksort */
```



```
1037
1038 void vecswap2(unsigned char **a, unsigned char **b, int n)
1039 {   while (n-- > 0) {
1040         unsigned char *t = *a;
1041         *a++ = *b;
1042         *b++ = t;
1043     }
1044 }
1045
1046 #define swap2(a, b) { t = *(a); *(a) = *(b); *(b) = t; }
1047 #define ptr2char(i) (*(*(i) + depth))
1048
1049 unsigned char **med3func(unsigned char **a, unsigned char **b, unsigned char **c, int depth)
1050 {   int va, vb, vc;
1051     if ((va=ptr2char(a)) == (vb=ptr2char(b)))
1052         return a;
1053     if ((vc=ptr2char(c)) == va || vc == vb)
1054         return c;
1055     return va < vb ?
1056           (vb < vc ? b : (va < vc ? c : a ) )
1057         : (vb > vc ? b : (va < vc ? a : c ) );
1058 }
1059 #define med3(a, b, c) med3func(a, b, c, depth)
1060
1061 void inssort(unsigned char **a, int n, int d)
1062 {   unsigned char **pi, **pj, *s, *t;
1063     for (pi = a + 1; --n > 0; pi++)
1064         for (pj = pi; pj > a; pj--) {
1065             /* Inline strcmp: break if *(pj-1) <= *pj */
1066             for (s=*(pj-1)+d, t=*pj+d; *s==*t && *s!=0; s++, t++)
1067                 ;
1068             if (*s <= *t)
1069                 break;
1070             swap2(pj, pj-1);
1071     }
1072 }
1073
1074 void mkqsort(unsigned char **a, int n, int depth)
1075 {   int d, r, partval;
1076     unsigned char **pa, **pb, **pc, **pd, **pl, **pm, **pn, *t;
1077     if (n < 20) {
1078         inssort(a, n, depth);
1079         return;
1080     }
1081     pl = a;
1082     pm = a + (n/2);
1083     pn = a + (n-1);
1084     if (n > 30) { /* On big arrays, pseudomedian of 9 */
1085         d = (n/8);
1086         pl = med3(pl, pl+d, pl+2*d);
1087         pm = med3(pm-d, pm, pm+d);
1088         pn = med3(pn-2*d, pn-d, pn);
1089     }
1090     pm = med3(pl, pm, pn);
1091     swap2(a, pm);
1092     partval = ptr2char(a);
1093     pa = pb = a + 1;
1094     pc = pd = a + n-1;
1095     for (;;) {
1096         while (pb <= pc && (r = ptr2char(pb)-partval) <= 0) {
1097             if (r == 0) { swap2(pa, pb); pa++; }
1098             pb++;
1099         }
1100         while (pb <= pc && (r = ptr2char(pc)-partval) >= 0) {
1101             if (r == 0) { swap2(pc, pd); pd--; }
1102             pc--;
1103        }
1104         if (pb > pc) break;
1105         swap2(pb, pc);
1106         pb++;
1107         pc--;
1108     }
1109     pn = a + n;
1110     r = min(pa-a, pb-pa);    vecswap2(a,  pb-r, r);
1111     r = min(pd-pc, pn-pd-1); vecswap2(pb, pn-r, r);
1112     if ((r = pb-pa) > 1)
1113         mkqsort(a, r, depth);
1114     if (ptr2char(a + r) != 0)
1115         mkqsort(a + r, pa-a + pn-pd-1, depth+1);
1116     if ((r = pd-pc) > 1)
1117         mkqsort(a + n-r, r, depth);
1118 }
1119
1120 void mkqsort_main(unsigned char **a, int n) { mkqsort(a, n, 0); }
1121 // mkqsort.c END ************************************************************
1122
1123 // Why Sinha uses int instead of long??!!
1124 static int readlines(char *file_name, string **lines)
```

```
{
   int nlines = 0;
   size_t size;
   FILE *in_file;
   string basep, cur, next;
   string *ASbackup;

   if (!(in_file = fopen(file_name, "rb"))) {
      printf( "Leprechaun: Can't open file %s \n", file_name );
      exit(-1);
   }
   fseek(in_file, 0, SEEK_END);
   size = ftell(in_file);
   fseek(in_file, 0, SEEK_SET);
   if (!(basep = (string) malloc(size*sizeof(char_t)))) return -1;
   printf( "Allocated memory for words in MB: %lu\n", ((size*sizeof(char_t))>>20)+1 );
   if (fread(basep, 1, size, in_file) < size) {
      printf( "Leprechaun: Can't read file %s \n", file_name );
      exit(-1);
   }
   fclose(in_file);

// GET nlines:
   cur = basep;
   while (cur < basep + size) {
      next = cur;
      while ((next < basep + size) && (*next != '\n')) {next++;}
      *--next = '\0';          // This is ala DOS i.e. Windows
                               // 1310 not 10(\n=10)
      cur = next + 2;
      nlines++;
   }

// printf("%lu\n",(unsigned long)*lines); -> backup = *lines = 0
ASbackup = (string *)malloc( nlines*sizeof(string) ); // sizeof(string) is 4
if( ASbackup == NULL )
{ puts( "Leprechaun: Needed memory allocation denied!\n" ); return( 1 ); }
printf( "Allocated memory for pointers-to-words in MB: %lu\n", ((nlines*sizeof(string))>>20)+1 );
*lines = ASbackup;
//printf("%lu\n",(unsigned long)*lines); -> backup = *lines = ASbackup = 6946888

// Upload nlines times:
   nlines = 0;
   cur = basep;
   while (cur < basep + size) {
      next = cur;
      while ((next < basep + size) && (*next != '\n')) {next++;}
      *--next = '\0';          // This is ala DOS i.e. Windows
                               // 1310 not 10(\n=10)
      ASbackup[nlines] = cur;
      cur = next + 2;
      nlines++;
   }
   return nlines;
}

void x64toaKAZE (      /* stdcall is faster and smaller... Might as well use it for the helper. */
        unsigned long long val,
        char *buf,
        unsigned radix,
        int is_neg
        )
{
        char *p;                /* pointer to traverse string */
        char *firstdig;         /* pointer to first digit */
        char temp;              /* temp char */
        unsigned digval;        /* value of digit */

        p = buf;

        if ( is_neg )
        {
            *p++ = '-';         /* negative, so output '-' and negate */
            val = (unsigned long long)(-(long long)val);
        }

        firstdig = p;           /* save pointer to first digit */

        do {
            digval = (unsigned) (val % radix);
            val /= radix;       /* get next digit */

            /* convert to ascii and store */
            if (digval > 9)
                *p++ = (char) (digval - 10 + 'a');  /* a letter */
            else
                *p++ = (char) (digval + '0');       /* a digit */
        } while (val > 0);
```



```

        /* We now have the digit of the number in the buffer, but in reverse
           order.  Thus we reverse them now. */

        *p-- = '\0';            /* terminate string; p points to last digit */

        do {
            temp = *p;
            *p = *firstdig;
            *firstdig = temp;   /* swap *p and *firstdig */
            --p;
            ++firstdig;         /* advance to next two digits */
        } while (firstdig < p); /* repeat until halfway */
}

/* Actual functions just call conversion helper with neg flag set correctly,
   and return pointer to buffer. */

char * _i64toaKAZE (
        long long val,
        char *buf,
        int radix
        )
{
        x64toaKAZE((unsigned long long)val, buf, radix, (radix == 10 && val < 0));
        return buf;
}

char * _ui64toaKAZE (
        unsigned long long val,
        char *buf,
        int radix
        )
{
        x64toaKAZE(val, buf, radix, 0);
        return buf;
}

char * _ui64toaKAZEzerocomma (
        unsigned long long val,
        char *buf,
        int radix
        )
{
                        char *p;
                        char temp;
                        int txpman;
                        int pxnman;
        x64toaKAZE(val, buf, radix, 0);
                        p = buf;
                        do {
                        } while (*++p != '\0');
                        p--; // p points to last digit
                             // buf points to first digit
                        buf[26] = 0;
                        txpman = 1;
                        pxnman = 0;
                        do
                        { if (buf <= p)
                          { temp = *p;
                            buf[26-txpman] = temp; pxnman++;
                            p--;
                            if (pxnman % 3 == 0)
                            { txpman++;
                              buf[26-txpman] = (char) (',');
                            }
                          }
                          else
                          { buf[26-txpman] = (char) ('0'); pxnman++;
                            if (pxnman % 3 == 0)
                            { txpman++;
                              buf[26-txpman] = (char) (',');
                            }
                          }
                          txpman++;
                        } while (txpman <= 26);
        return buf;
}

char * _ui64toaKAZEcomma (
        unsigned long long val,
        char *buf,
        int radix
        )
{
                        char *p;
                        char temp;
                        int txpman;
```

```
                        int pxnman;
        x64toaKAZE(val, buf, radix, 0);
                        p = buf;
                        do {
                        } while (*++p != '\0');
                        p--; // p points to last digit
                             // buf points to first digit
                        buf[26] = 0;
                        txpman = 1;
                        pxnman = 0;
                        while (buf <= p)
                        { temp = *p;
                          buf[26-txpman] = temp; pxnman++;
                          p--;
                          if (pxnman % 3 == 0 && buf <= p)
                          { txpman++;
                            buf[26-txpman] = (char) (',');
                          }
                          txpman++;
                        }
        return buf+26-(txpman-1);
}

unsigned char KuxHash(char *str)
{ unsigned char h = 0;
  int max31 = 0;
  //while (*str)
  while (str[max31])
  { h = h ^ str[max31++];
    //h = h ^ *str++; // I am not sure 'str' is returned changed after return?!
  }
  return h; // 00..255 i.e. 2^8=256
}

int KuxHash2(char *str)
{ int h = 0;
  unsigned long h2 = 0; // must be long: 31*'z'=31*122
  int max31 = 0;
  while (str[max31])
  { h = h ^ str[max31]; // 00..255 i.e. 2^8=256
    //h2 = h2 + str[max31++];       // [113s]
    h2 = h2 + max31 * str[max31++];
  }
  h=h<<4; // 00..15 i.e. 2^4=16
  //h = h|( str[0] ^ str[max31-1] ); // [111s] a..z: each XOR each gives 00..31
  h = h|( h2%((1<<4)-1) );
  return h; // 00..4095 i.e. 2^12=4096
}

//OSHO test - Attempts to Find/Put a WORD into linked list count: 32,011,937
int KuxHash3(char *str)
{ int h = 0;
  unsigned long h2 = 0; // must be long: 31*'z'=31*122
  int max31 = 0;
  while (str[max31])
  { h = h ^ str[max31]; // 00..255 i.e. 2^8=256
    //h2 = h2 + str[max31++];       // [113s]
    h2 = h2 + str[max31++] * (max31+1);
  }
// Result is: 7bits in 'h' and 32bits in 'h2'.

                        //printf("%s:\n ",str);
                        //printf("%d ",h);
  h=h<<6; // 00..15 i.e. 00-05+7bits=13bits
                        //printf("%d ",h);
                        //printf("%d ",h2);
  //h = h|( str[0] ^ str[max31-1] ); // [111s] a..z: each XOR each gives 00..31
  h = h|( h2%((1<<6)-1) ); // 64-1=63=9*7; 61 is prime
                        //printf("%d \n",h);
  return h; // 00..8191 i.e. 2^13=8192
}

//OSHO test - Attempts to Find/Put a WORD into a linked list count: 31,927,285
int KuxHash3plus(char *str)
{ int h = 0;
  unsigned long h2 = 0; // must be long: 31*'z'=31*122
  int max31 = 0;
  while (str[max31])
  { h = h ^ str[max31]; // 00..255 i.e. 2^8=256
    //h2 = h2 + str[max31++];       // [113s]
    h2 = h2 + str[max31++] * (max31+1);
  }
// Result is: 7bits in 'h' and 32bits in 'h2'.

                        //printf("%s:\n ",str);
                        //printf("%d ",h);
// a in ASCII is 097 = 0110 0001
// z in ASCII is 122 = 0111 1010
```



```
// Above two lines show that bits 8-7-6 are always 0-1-1 so need for low 5 bits.
  //h=h<<8; // 00..15 i.e. 5bits + 00-07bits=13bits
                        //printf("%d ",h);
                        //printf("%d ",h2);
  //h = h|( str[0] ^ str[max31-1] ); // [111s] a..z: each XOR each gives 00..31
  h = (( h<<8 )|( h2%(251) ))&8191; // 251 prime
                        //printf("%d \n",h);
  return h; // 00..8191 i.e. 2^13=8192
}

/*
PUBLIC        _KuxHash3plus
; Function compile flags: /Ogty
_TEXT         SEGMENT
_str$ = 8                                               ; size = 4
_KuxHash3plus PROC NEAR
; Line 511
mov  ecx, DWORD PTR _str$[esp-4]
mov  dl, BYTE PTR [ecx]
push esi
xor  esi, esi
xor  eax, eax
test dl, dl
je   SHORT $L1561
push ebx
push edi
mov  edi, 1
sub  edi, ecx
npad 8
$L1560:
; Line 512
movsx         edx, BYTE PTR [ecx]
; Line 514
lea  ebx, DWORD PTR [edi+ecx]
imul ebx, edx
xor  esi, edx
mov  dl, BYTE PTR [ecx+1]
add  eax, ebx
inc  ecx
test dl, dl
jne  SHORT $L1560
pop  edi
pop  ebx
$L1561:
; Line 527
xor  edx, edx
mov  ecx, 251                               ; 000000fbH
div  ecx
shl  esi, 8
mov  eax, edx
; Line 529
or   eax, esi
and  eax, 8191                              ; 00001fffH
pop  esi
; Line 530
ret  0
_KuxHash3plus ENDP
_TEXT         ENDS
*/

//OSHO test - Attempts to Find/Put a WORD into a linked list count: 32,021,975
int KuxHash4(char *str)
{
  int h2 = 0;
    for (; *str != 0; str++) {
      //h2 = (127*h2 + *str) % (8192-1); // 2^13-1 = 8191 is Mersenne prime
      h2 = ((h2<<7) + *str) % (8192-1); // 2^13-1 = 8191 is Mersenne prime
    }

  return h2; // 00..8191 i.e. 2^13=8192
}

/*
int hash(char *v, int M)
  { int h = 0, a = 127;
    for (; *v != 0; v++)
      h = (a*h + *v) % M;
    return h;
  }

int hashU(char *v, int M)
  { int h, a = 31415, b = 27183;
    for (h = 0; *v != 0; v++, a = a*b % (M-1))
        h = (a*h + *v) % M;
    return (h < 0) ? (h + M) : h;
  }
*/
```

```
// Kaze: My appreciation of FNV is far beyound C code optimization, it is alchemical, and why not, magical.
/*
FNV hash history
    The basis of the FNV hash algorithm was taken from an idea sent as reviewer comments to the IEEE POSIX P1003.2 committee
by Glenn Fowler and Phong Vo back in 1991. In a subsequent ballot round: Landon Curt Noll improved on their algorithm.
Some people tried this hash and found that it worked rather well. In an EMail message to Landon, they named it
the ``Fowler/Noll/Vo'' or FNV hash.
    FNV hashes are designed to be fast while maintaining a low collision rate. The FNV speed allows one to quickly hash
lots of data while maintaining a reasonable collision rate. The high dispersion of the FNV hashes makes them well suited
for hashing nearly identical strings such as URLs, hostnames, filenames, text, IP addresses, etc.
*/

/* NOTE: u_int64_t is a 64 bit unsigned type */
/* NOTE: u_int32_t is a 32 bit unsigned type */
/* NOTE: u_int16_t is a 16 bit unsigned type */
/* NOTE: u_int8_t is a 8 bit unsigned type */

//typedef unsigned char u_int8_t; //FNV only
//typedef unsigned long u_int32_t; //FNV only
//typedef unsigned long long u_int64_t; //FNV only

// 32 bit FNV_prime = 2^24 + 2^8 + 0x93 = 16777619
// 64 bit FNV_prime = 2^40 + 2^8 + 0xb3 = 1099511628211

// 32 bit offset_basis = 2166136261
// 64 bit offset_basis = 14695981039346656037

#define FNV1_64_INIT ((u_int64_t)14695981039346656037)
#define FNV1_64_PRIME ((u_int64_t)1099511628211)
#define FNV1_32_INIT ((u_int32_t)2166136261)
#define FNV1_32_PRIME ((u_int32_t)602173697)
// FNV1A_Hash_4_OCTETS gives dispersion as follows:
//3549448: 1607
//3549669: 171072511
//3550710: 272961023
//3550733: 172783361
//3550734: 431562497
//3550929: 204312319
//3550984: 562853633
//3550991: 551362303
//3551359: 332820737
//3551484: 354126079
//3551514: 407138561
//3551523: 442058753
//3551701: 449230849
//3551736: 107712257
//3551961: 428904191
//3552039: 602173697
//3552103: 588411137

#define FNV_64A_OP(hash, octet) \
    (((u_int64_t)(hash) ^ (u_int8_t)(octet)) * FNV1_64_PRIME)

#define FNV_64A_OP64(hash, octet) \
    (((u_int64_t)(hash) ^ (u_int64_t)(octet)) * FNV1_64_PRIME)

#define FNV_32A_OP_GENERIC(hash, octet) \
    (((u_int32_t)(hash) ^ (u_int8_t)(octet)) * 16777619)

#define FNV_32A_OP(hash, octet) \
    (((u_int32_t)(hash) ^ (u_int8_t)(octet)) * FNV1_32_PRIME)

#define FNV_32A_OP_MULless_core(hash, octet) \
    ( (u_int32_t)(hash) ^ (u_int8_t)(octet) )

#define FNV_32A_OP_MULless(hash, octet) \
    ( (FNV_32A_OP_MULless_core(hash, octet)<<5) - FNV_32A_OP_MULless_core(hash, octet) )

#define FNV_32A_OP32(hash, octet) \
    (((u_int32_t)(hash) ^ (u_int32_t)(octet)) * FNV1_32_PRIME)

#define FNV_32A_OP64(hash, octet) \
    (((u_int64_t)(hash) ^ (u_int64_t)(octet)) * FNV1_32_PRIME)

#define FNV_32A_OP32_MULless_core(hash, octet) \
    ( (u_int32_t)(hash) ^ (u_int32_t)(octet) )

#define FNV_32A_OP32_MULless(hash, octet) \
    ( (FNV_32A_OP32_MULless_core(hash, octet)<<5) - FNV_32A_OP32_MULless_core(hash, octet) )


// Invoking: FNV1A_Hash_4_OCTETS_31(wrd, wrdlen>>2) // = 0,1,2,3,4,5,6,7 [1..31]
int FNV1A_Hash_4_OCTETS_31(char *str, int wrdlen_QUADRUPLETS)
{
u_int32_t hash;
char *p;

```



```
hash = FNV1_32_INIT;
p=str;

// The goal of stage #1: to reduce number of 'imul's in fact to reduce loops.

// Stage #1:
for (; wrdlen_QUADRUPLETS != 0; --wrdlen_QUADRUPLETS) {
    hash = FNV_32A_OP32_MULless(hash, (unsigned long)*(long *)p); // mov edi, DWORD PTR [eax]
    p=p+4; // add eax, 4
}

// Stage #2:
for (; *p; ++p) {
    hash = FNV_32A_OP_MULless(hash, *p); // mov dl, BYTE PTR [ecx]
}

  //return ((hash>>13) ^ hash) & 8191; // (((u_int32_t)1<<(x))-1) where x=13
  return ((hash>>16) ^ hash) & 8191; // 00..8191 i.e. 2^13=8192
}


// Invoking: FNV1A_Hash_4_OCTETS(wrd, wrdlen>>2) // = 0,1,2,3,4,5,6,7 [1..31]
int FNV1A_Hash_4_OCTETS(char *str, int wrdlen_QUADRUPLETS)
{
u_int32_t hash;
char *p;

hash = FNV1_32_INIT;
p=str;

// The goal of stage #1: to reduce number of 'imul's.

// Stage #1:
for (; wrdlen_QUADRUPLETS != 0; --wrdlen_QUADRUPLETS) {
    hash = FNV_32A_OP32(hash, (unsigned long)*(long *)p); // mov edi, DWORD PTR [eax]
    p=p+4; // add eax, 4
}

// Stage #2:
for (; *p; ++p) {
    hash = FNV_32A_OP(hash, *p); // mov dl, BYTE PTR [ecx]
}

  //return ((hash>>13) ^ hash) & 8191; // (((u_int32_t)1<<(x))-1) where x=13
  return ((hash>>16) ^ hash) & 8191; // 00..8191 i.e. 2^13=8192
}

/*
Results for 'FNV1A_Hash_8_OCTETS':
Bytes per second performance: 23,110,160B/s
Words per second performance: 1,959,516W/s
Input File with a list of TEXTual Files: Leprechaun_vs_Wikipedia_LATIN-WORDS.lst
Size of all TEXTual Files: 415,982,896
Word count: 35,271,297 of them 22,202,980 distinct
Number Of Files: 8
Number Of Lines: 35271297
Allocated memory in MB: 1950
Number Of Trees(GREATER THE BETTER): 3419429
Forest population(Hash Function Quality regarding Collisions i.e. Hash Table Utilization): 51%
Number Of Hash Collisions(Distinct WORDs - Number Of Trees): 18783551
Maximum Attempts to Find/Put a WORD into a Binary-Search-Tree: '1,119'
Total Attempts to Find/Put WORDs into Binary-Search-Trees: 268,085,505
Total Number of LEAFs in Binary-Search-Trees(GREATER THE BETTER): 7,690,615
Perfectly-Balanced-Binary-Search-Tree for MaxNODEs = 2,622 must have PEAK = 12 = rounding down of integer (1+lb(2,622))
Binary-Search-Tree(1st out of 1) with MaxNODEs = 2,622 has PEAK = 592 and LEAFs = 689
Binary-Search-Tree(1st out of 1) with MaxPEAK = '1,119' has NODEs = 1,537 and LEAFs = 287
Binary-Search-Tree(1st out of 1) with MaxLEAFs = 731 has NODEs = 2,517 and PEAK = 448
*/
// Invoking: FNV1A_Hash_8_OCTETS(wrd, wrdlen>>3) // = 0,1,2,3 [1..31]
int FNV1A_Hash_8_OCTETS(char *str, int wrdlen_OCTETS)
{
u_int32_t hash;
char *p;

hash = FNV1_32_INIT;
p=str;

// The goal of stage #1: to reduce number of 'imul's.

// Stage #1:
for (; wrdlen_OCTETS != 0; --wrdlen_OCTETS) {
    hash = FNV_32A_OP64(hash, (unsigned long long)*(long *)p); // mov edi, DWORD PTR [eax]
    p=p+8; // add eax, 4
}

// Stage #2:
for (; *p; ++p) {
    hash = FNV_32A_OP(hash, *p); // mov dl, BYTE PTR [ecx]
```

```
}

  //return ((hash>>13) ^ hash) & 8191; // (((u_int32_t)1<<(x))-1) where x=13
  return ((hash>>16) ^ hash) & 8191; // 00..8191 i.e. 2^13=8192
}


// Invoking: FNV1A_Hash_Granularity(wrd, wrdlen>>0|2|3, 0|2|3)
int FNV1A_Hash_Granularity(char *str, int wrdlen_granulated, int Granularity) // wrdlen>>0=wrdlen
{
u_int32_t hash;
u_int64_t hash64;
char *p;

hash = FNV1_32_INIT;
p=str;

// The goal of stage #1: to reduce number of 'imul's and mainly: the number of loops.

// Stage #1:
        if (Granularity == 2) {
for (; wrdlen_granulated != 0; --wrdlen_granulated) {
    hash = FNV_32A_OP32(hash, (u_int32_t)*(u_int32_t *)p);
    p=p+4; // (1<<Granularity): 1<<0=1, 1<<2=4, 1<<3=8
}
        }
        if (Granularity == 3) {
hash64 = FNV1_64_INIT;
for (; wrdlen_granulated != 0; --wrdlen_granulated) {
    hash64 = FNV_64A_OP64(hash64, (u_int64_t)*(u_int64_t *)p);
    p=p+8; // (1<<Granularity): 1<<0=1, 1<<2=4, 1<<3=8
}
for (; *p; ++p) {
    hash64 = FNV_64A_OP(hash64, (u_int8_t)*(u_int8_t *)p);
}

  //return ((hash>>13) ^ hash) & 8191; // (((u_int32_t)1<<(x))-1) where x=13
  return ((hash64>>51) ^ hash64) & 8191; // 00..8191 i.e. 2^13=8192
  // probably better shifting is not by 16 bits but ...
  //hash64>>16: 3,544,160 just bad
  //hash64>>33: 3,547,854
  //hash64>>34: 3,547,266
  //hash64>>35: 3,547,453
  //hash64>>36: 3,547,242
  //hash64>>40: 3,548,263
  //hash64>>44: 3,548,242
  //hash64>>45: 3,549,056
  //hash64>>46: 3,549,207
  //hash64>>47: 3,549,094
  //hash64>>50: 3,549,392
  //hash64>>51: 3,549,395 i.e. maximum shift: the 13 most significant bits i.e. (64-13); closest to 3,549,448

// Above results are obtained for following set:
//if (wrdlen<=19) // 4x4+3=19 i.e. last contains 7 clashes
//              Slot = FNV1A_Hash_Granularity(wrd, wrdlen>>2, 2)<<2; //13+++++
//else            // 2x8+4=20 i.e. first contains 6 clashes
//              Slot = FNV1A_Hash_Granularity(wrd, wrdlen>>3, 3)<<2; //13+++++
        }

        //if (Granularity != 3) {
// Stage #2:
for (; *p; ++p) {
    hash = FNV_32A_OP(hash, (u_int8_t)*(u_int8_t *)p);
}

  //return ((hash>>13) ^ hash) & 8191; // (((u_int32_t)1<<(x))-1) where x=13
  return ((hash>>16) ^ hash) & 8191; // 00..8191 i.e. 2^13=8192
        //}
}


// char *string;                /* the string to 64 bit FNV-1a hash */
// u_int64_t hash;              /* will hold the final value of the hash */
// char *p;
//
// hash = FNV1_64_INIT;
// for (p=string; *p; ++p) {
//     hash = FNV_64A_OP(hash, *p);
// }


// If you need an x-bit hash where x is not a power of 2,
// 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.
// By xor-folding we mean shift the excess high order bits down and xor them with the lower x-bits.
// For tiny x < 16 bit values, we recommend using a 32 bit FNV-1 hash as follows:

//    /* NOTE: for 0 < x < 16 ONLY!!! */
//    #define TINY_MASK(x) (((u_int32_t)1<<(x))-1)
```



```
//    #define FNV1_32_INIT ((u_int32_t)2166136261)
//    u_int32_t hash;
//    void *data;
//    size_t data_len;
//
//    hash = fnv_32_buf(data, data_len, FNV1_32_INIT);
//    hash = (((hash>>x) ^ hash) & TINY_MASK(x));


int FNV1A_Hash_SHIFTless_XORless(char *str)
{
u_int32_t hash;                 /* will hold the final value of the hash */
char *p;

hash = FNV1_32_INIT;
for (p=str; *p; ++p) {
    hash = FNV_32A_OP(hash, *p);
}
//hash = ((hash>>13) ^ hash) & 8191; // (((u_int32_t)1<<(x))-1) where x=13

  return hash & 8191; // 00..8191 i.e. 2^13=8192
}

/*
_FNV1A_Hash_SHIFTless_XORless PROC NEAR
; Line 721
mov edx, DWORD PTR _str$[esp-4]
mov cl, BYTE PTR [edx]
test cl, cl
mov eax, -2128831035                    ; 811c9dc5H
je  SHORT $L1582
npad 1
$L1580:
; Line 722
movzx       ecx, cl
xor ecx, eax
imul ecx, 16777619                              ; 01000193H
inc edx
mov eax, ecx
mov cl, BYTE PTR [edx]
test cl, cl
jne SHORT $L1580
$L1582:
; Line 726
and eax, 8191                           ; 00001fffH
; Line 727
ret 0
_FNV1A_Hash_SHIFTless_XORless ENDP
*/


int FNV1A_Hash(char *str)
{
u_int32_t hash;                 /* will hold the final value of the hash */
char *p;

hash = FNV1_32_INIT;
for (p=str; *p; ++p) {
    hash = FNV_32A_OP(hash, *p);
}
//hash = ((hash>>13) ^ hash) & 8191; // (((u_int32_t)1<<(x))-1) where x=13

  return ((hash>>13) ^ hash) & 8191; // 00..8191 i.e. 2^13=8192
}

/*
_FNV1A_Hash PROC NEAR
; Line 722
mov edx, DWORD PTR _str$[esp-4]
mov al, BYTE PTR [edx]
test al, al
mov ecx, -2128831035                    ; 811c9dc5H
je  SHORT $L1582
npad 1
$L1580:
; Line 723
movzx       eax, al
xor eax, ecx
imul eax, 16777619                              ; 01000193H
inc edx
mov ecx, eax
mov al, BYTE PTR [edx]
test al, al
jne SHORT $L1580
$L1582:
; Line 727
mov eax, ecx
shr eax, 13                             ; 0000000dH
```

```
xor eax, ecx
and eax, 8191                           ; 00001fffH
; Line 728
ret 0
_FNV1A_Hash ENDP
*/

/*
Wayne Diamond implemented 32-bit FNV algorithm in PowerBASIC inline x86 assembly:


    FUNCTION FNV32(BYVAL dwOffset AS DWORD, BYVAL dwLen AS DWORD, BYVAL offset_basis AS DWORD) AS DWORD
    #REGISTER NONE
    ! mov esi, dwOffset      ;esi = ptr to buffer
    ! mov ecx, dwLen         ;ecx = length of buffer (counter)
    ! mov eax, offset_basis  ;set to 2166136261 for FNV-1
    ! mov edi, &h01000193    ;FNV_32_PRIME = 16777619
    ! xor ebx, ebx           ;ebx = 0
    nextbyte:
    ! mul edi                ;eax = eax * FNV_32_PRIME
    ! mov bl, [esi]          ;bl = byte from esi
    ! xor eax, ebx           ;al = al xor bl
    ! inc esi                ;esi = esi + 1 (buffer pos)
    ! dec ecx                ;ecx = ecx - 1 (counter)
    ! jnz nextbyte           ;if ecx is 0, jmp to NextByte
    ! mov FUNCTION, eax      ;else, function = eax
    END FUNCTION

Wayne said:

    ''Just thought I should let you know that I've ported the 32-bit FNV algorithm over to inline assembly.
It's actually in PowerBASIC (www.powerbasic.com) format - a compiler I use, but the main function is all assembly.
It could be optimized further in terms of saving a couple of clock cycles,
but it's fairly optimized al ready - only 6 instructions in the main loop, plus 5 setup instructions,
and compiles to just 33 bytes.''

M.S.Schulte sent us these 32-bit FNV-1 and FNV-1a x86 assembler implementations (written in flat assembler),
half of which were optimized for speed, the other half were optimized for size:

    small_fnv32:    ;FNV1 32bit (size: 31 bytes)
    ;                Intel Core 2 Duo E6600: 354.20 mb/s
       push    esi
       push    edi
       mov     esi, [esp + 0ch] ;buffer
       mov     ecx, [esp + 10h] ;length
       mov     eax, [esp + 14h] ;basis
       mov     edi, 01000193h   ;fnv_32_prime
     next:
       mul     edi
       xor     al, [esi]
       inc     esi
       loop    snext
       pop     edi
       pop     esi
       retn    0ch

     small_fnv32a:   ;FNV1a 32bit (size: 31 bytes)
    ;                 Intel Core 2 Duo E6600: 327.68 mb/s
       push    esi
       push    edi
       mov     esi, [esp + 0ch] ;buffer
       mov     ecx, [esp + 10h] ;length
       mov     eax, [esp + 14h] ;basis
       mov     edi, 01000193h   ;fnv_32_prime
     nexta:
       xor     al, [esi]
       mul     edi
       inc     esi
       loop    nexta
       pop     edi
       pop     esi
       retn    0ch

    fast_fnv32:    ;FNV1 32bit (size: 36 bytes)
    ;               Intel Core 2 Duo E6600: 565.12 mb/s
       push    ebx
       push    esi
       push    edi
       mov     esi, [esp + 10h] ;buffer
       mov     ecx, [esp + 14h] ;length
       mov     eax, [esp + 18h] ;basis
       mov     edi, 01000193h   ;fnv_32_prime
       xor     ebx, ebx
     next:
       mul     edi
       mov     bl, [esi]
       xor     eax, ebx
       inc     esi
```



```
       dec     ecx
       jnz     next
       pop     edi
       pop     esi
       pop     ebx
       retn    0ch

     fast_fnv32a:   ;FNV1a 32bit (size: 36 bytes)
    ;                Intel Core 2 Duo E6600: 574.95 mb/s
       push    ebx
       push    esi
       push    edi
       mov     esi, [esp + 10h] ;buffer
       mov     ecx, [esp + 14h] ;length
       mov     eax, [esp + 18h] ;basis
       mov     edi, 01000193h   ;fnv_32_prime
       xor     ebx, ebx
     nexta:
       mov     bl, [esi]
       xor     eax, ebx
       mul     edi
       inc     esi
       dec     ecx
       jnz     nexta
       pop     edi
       pop     esi
       pop     ebx
       retn    0ch
*/

//Number Of Trees(GREATER THE BETTER): 3525737
//Forest population(Hash Function Quality regarding Collisions i.e. Hash Table Utilization): 53%
//Number Of Hash Collisions(Distinct WORDs - Number Of Trees): 18677243
int Hash17_unrolled(const char *key, int wrdlen)
{
int hash = 1;
        int i;
for(i = 0; i < (wrdlen & -2); i += 2) {
    hash = (17) * hash + (key[i] - ' ');
    hash = (17) * hash + (key[i+1] - ' ');
}
if(wrdlen & 1)
    hash = (17) * hash + (key[wrdlen-1] - ' ');
return ( hash ^ (hash >> 16) ) & 8191;
}

//hash = 1:
//Number Of Trees(GREATER THE BETTER): 3556516
//Forest population(Hash Function Quality regarding Collisions i.e. Hash Table Utilization): 53%
//Number Of Hash Collisions(Distinct WORDs - Number Of Trees): 18646464
//hash = 13:
//Number Of Trees(GREATER THE BETTER): 3556755
//Forest population(Hash Function Quality regarding Collisions i.e. Hash Table Utilization): 53%
//Number Of Hash Collisions(Distinct WORDs - Number Of Trees): 18646225
//hash = 11:
//Number Of Trees(GREATER THE BETTER): 3557011
//Forest population(Hash Function Quality regarding Collisions i.e. Hash Table Utilization): 53%
//Number Of Hash Collisions(Distinct WORDs - Number Of Trees): 18645969
//hash = 7:
//Number Of Trees(GREATER THE BETTER): 3557181
//Forest population(Hash Function Quality regarding Collisions i.e. Hash Table Utilization): 53%
//Number Of Hash Collisions(Distinct WORDs - Number Of Trees): 18645799
int Alfalfa(const char *key, int wrdlen)
{
int hash = 7;
        int i;
for(i = 0; i < (wrdlen & -2); i += 2) {
    hash = (17+9) * ((17+9) * hash + (key[i])) + (key[i+1]);
}
if(wrdlen & 1)
    hash = (17+9) * hash + (key[wrdlen-1]);
return ( hash ^ (hash >> 16) ) & 8191;
}

/*
[FNV1A 'shift-less-&-xor-less' hash used in Leprechaun r.13+++:]

int FNV1A_Hash_SHIFTless_XORless(char *str)
{
u_int32_t hash;
char *p;

hash = FNV1_32_INIT;
for (p=str; *p; ++p) {
    hash = FNV_32A_OP(hash, *p);
}
//hash = ((hash>>13) ^ hash) & 8191; // (((u_int32_t)1<<(x))-1) where x=13

```

```
  return hash & 8191; // 00..8191 i.e. 2^13=8192
}

Words per second performance: 837,458W/s
Input File with a list of TEXTual Files: wikipedia-en-html.tar.wrd.lst
Word count: 12,561,874 of them 12,561,874 distinct
Number Of Trees(GREATER THE BETTER): 2772875
Forest population(Hash Function Quality regarding Collisions i.e. Hash Table Utilization): 41%
Number Of Hash Collisions(Distinct WORDs - Number Of Trees): 9788999

Words per second performance: 1,007,751W/s
Input File with a list of TEXTual Files: Leprechaun_vs_Wikipedia_LATIN-WORDS.lst
Word count: 35,271,297 of them 22,202,980 distinct
Number Of Trees(GREATER THE BETTER): 3537061
Forest population(Hash Function Quality regarding Collisions i.e. Hash Table Utilization): 53%
Number Of Hash Collisions(Distinct WORDs - Number Of Trees): 18665919

[My '2in1' hash used in Leprechaun r.13++:]

int KuxHash3plus(char *str)
{ int h = 0;
  unsigned long h2 = 0; // must be long: 31*'z'=31*122
  int max31 = 0;
  while (str[max31])
  { h = h ^ str[max31]; // 00..255 i.e. 2^8=256
    //h2 = h2 + str[max31++];       // [113s]
    h2 = h2 + str[max31++] * (max31+1);
  }
// Result is: 7bits in 'h' and 32bits in 'h2'.

                        //printf("%s:\n ",str);
                        //printf("%d ",h);
// a in ASCII is 097 = 0110 0001
// z in ASCII is 122 = 0111 1010
// Above two lines show that bits 8-7-6 are always 0-1-1 so need for low 5 bits.
  //h=h<<8; // 00..15 i.e. 5bits + 00-07bits=13bits
                        //printf("%d ",h);
                        //printf("%d ",h2);
  //h = h|( str[0] ^ str[max31-1] ); // [111s] a..z: each XOR each gives 00..31
  h = (( h<<8 )|( h2%(251) ))&8191; // 251 prime
                        //printf("%d \n",h);
  return h; // 00..8191 i.e. 2^13=8192
}

Words per second performance: 785,117W/s
Input File with a list of TEXTual Files: wikipedia-en-html.tar.wrd.lst
Word count: 12,561,874 of them 12,561,874 distinct
Number Of Trees(GREATER THE BETTER): 2663566
Forest population(Hash Function Quality regarding Collisions i.e. Hash Table Utilization): 40%
Number Of Hash Collisions(Distinct WORDs - Number Of Trees): 9898308

Words per second performance: 979,758W/s
Input File with a list of TEXTual Files: Leprechaun_vs_Wikipedia_LATIN-WORDS.lst
Word count: 35,271,297 of them 22,202,980 distinct
Number Of Trees(GREATER THE BETTER): 3410463
Forest population(Hash Function Quality regarding Collisions i.e. Hash Table Utilization): 51%
Number Of Hash Collisions(Distinct WORDs - Number Of Trees): 18792517

[Last standing for English(en)-Wikipedia's wordlist:]
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%
chongo's hash has better distribution than Kaze's hash by ((9898308 - 9788999)/9788999)*100% = 1.1%

[Last standing for LATIN(de,en,es,fr,it,nl,pt,ro)-Wikipedia's wordlist:]
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%
chongo's hash has better distribution than Kaze's hash by ((18792517 - 18665919)/18665919)*100% = 0.6%

Bottomline is:
Your hash thrash, my hash for trash, he-he.
Thanks a lot, again, Mr. Noll.

Yummy little file: http://www.isthe.com/chongo/src/fnv/fnv-5.0.2.tar.gz
*/

/*
// Paul Larson (http://research.microsoft.com/~PALARSON/)
UINT HashLarson(const CHAR *key, SIZE_T len) {
UINT hash = 0;
for(UINT i = 0; i < len; ++i)
    hash = 101 * hash + key[i];
return hash ^ (hash >> 16);
}

// Kernighan & Ritchie, "The C programming Language", 3rd edition.
UINT HashKernighanRitchie(const CHAR *key, SIZE_T len) {
UINT hash = 0;
for(UINT i = 0; i < len; ++i)
    hash = 31 * hash + key[i];
return hash;
```



```
}

// A hash function with multiplier 65599 (from Red Dragon book)
UINT Hash65599(const CHAR *key, SIZE_T len) {
UINT hash = 0;
for(UINT i = 0; i < len; ++i)
    hash = 65599 * hash + key[i];
return hash ^ (hash >> 16);
}

// FNV hash, http://isthe.com/chongo/tech/comp/fnv/
UINT HashFNV1a(const CHAR *key, SIZE_T len) {
UINT hash = 2166136261;
for(UINT i = 0; i < len; ++i)
    hash = 16777619 * (hash ^ key[i]);
return hash ^ (hash >> 16);
}

// Ramakrishna hash
UINT HashRamakrishna(const CHAR *key, SIZE_T len) {
UINT h = 0;
for(UINT i = 0; i < len; ++i) {
    h ^= (h << 5) + (h >> 2) + key[i];
}
return h;
}
*/

/*
Results for 'Hash_Alfalfa':
Bytes per second performance: 19,808,709B/s
Words per second performance: 1,679,585W/s
Input File with a list of TEXTual Files: Leprechaun_vs_Wikipedia_LATIN-WORDS.lst
Size of all TEXTual Files: 415,982,896
Word count: 35,271,297 of them 22,202,980 distinct
Number Of Files: 8
Number Of Lines: 35271297
Allocated memory in MB: 1950
Number Of Trees(GREATER THE BETTER): 3549079
Forest population(Hash Function Quality regarding Collisions i.e. Hash Table Utilization): 53%
Number Of Hash Collisions(Distinct WORDs - Number Of Trees): 18653901
Maximum Attempts to Find/Put a WORD into a Binary-Search-Tree: '37'
Total Attempts to Find/Put WORDs into Binary-Search-Trees: 117,063,824
Total Number of LEAFs in Binary-Search-Trees(GREATER THE BETTER): 8,072,279
Perfectly-Balanced-Binary-Search-Tree for MaxNODEs = 84 must have PEAK = 7 = rounding down of integer (1+lb(84))
Binary-Search-Tree(1st out of 2) with MaxNODEs = 84 has PEAK = 20 and LEAFs = 24
Binary-Search-Tree(1st out of 3) with MaxPEAK = '37' has NODEs = 67 and LEAFs = 17
Binary-Search-Tree(1st out of 1) with MaxLEAFs = 28 has NODEs = 78 and PEAK = 22
*/
UINT Hash_Alfalfa(const char *key, unsigned int wrdlen)
{
UINT hash = 7;
unsigned int i;
for (i = 0; i < (wrdlen & -2); i += 2) {
    hash = (53) * ((53) * hash + (key[i])) + (key[i+1]);
}
if (wrdlen & 1)
    hash = (53) * hash + (key[wrdlen-1]);
return ((hash>>16) ^ hash) & 8191;
}

/*
Results for 'HashAlfalfa_HALF':
Bytes per second performance: 19,808,709B/s
Words per second performance: 1,679,585W/s
Input File with a list of TEXTual Files: Leprechaun_vs_Wikipedia_LATIN-WORDS.lst
Size of all TEXTual Files: 415,982,896
Word count: 35,271,297 of them 22,202,980 distinct
Number Of Files: 8
Number Of Lines: 35271297
Allocated memory in MB: 1950
Number Of Trees(GREATER THE BETTER): 3550665
Forest population(Hash Function Quality regarding Collisions i.e. Hash Table Utilization): 53%
Number Of Hash Collisions(Distinct WORDs - Number Of Trees): 18652315
Maximum Attempts to Find/Put a WORD into a Binary-Search-Tree: '39'
Total Attempts to Find/Put WORDs into Binary-Search-Trees: 117,053,918
Total Number of LEAFs in Binary-Search-Trees(GREATER THE BETTER): 8,072,259
Perfectly-Balanced-Binary-Search-Tree for MaxNODEs = 87 must have PEAK = 7 = rounding down of integer (1+lb(87))
Binary-Search-Tree(1st out of 1) with MaxNODEs = 87 has PEAK = 21 and LEAFs = 27
Binary-Search-Tree(1st out of 2) with MaxPEAK = '39' has NODEs = 65 and LEAFs = 18
Binary-Search-Tree(1st out of 4) with MaxLEAFs = 27 has NODEs = 77 and PEAK = 23
*/
UINT HashAlfalfa_HALF(const char *key, unsigned int wrdlen)
{
UINT hash = 12;
UINT hashBUFFER;
unsigned int i,j;
for(i = 0; i < (wrdlen & -4); i += 4) {
```

```
    //hash = (( ((hash<<5)-hash) + key[i] )<<5) - ( ((hash<<5)-hash) + key[i] ) + (key[i+1]);
    hashBUFFER = ((hash<<5)-hash) + key[i];
    hash = (( hashBUFFER )<<5) - ( hashBUFFER ) + (key[i+1]);
    //hash = (( ((hash<<5)-hash) + key[i+2] )<<5) - ( ((hash<<5)-hash) + key[i+2] ) + (key[i+3]);
    hashBUFFER = ((hash<<5)-hash) + key[i+2];
    hash = (( hashBUFFER )<<5) - ( hashBUFFER ) + (key[i+3]);
}
for(j = 0; j < (wrdlen & 3); j += 1) {
    hash = ((hash<<5)-hash) + key[i+j];
}
return ((hash>>16) ^ hash) & 8191;
}

/*
Results for 'HashFNV1A_unrolled_Final':
Bytes per second performance: 19,808,709B/s
Words per second performance: 1,679,585W/s
Input File with a list of TEXTual Files: Leprechaun_vs_Wikipedia_LATIN-WORDS.lst
Size of all TEXTual Files: 415,982,896
Word count: 35,271,297 of them 22,202,980 distinct
Number Of Files: 8
Number Of Lines: 35271297
Allocated memory in MB: 1950
Number Of Trees(GREATER THE BETTER): 3445337
Forest population(Hash Function Quality regarding Collisions i.e. Hash Table Utilization): 52%
Number Of Hash Collisions(Distinct WORDs - Number Of Trees): 18757643
Maximum Attempts to Find/Put a WORD into a Binary-Search-Tree: '43'
Total Attempts to Find/Put WORDs into Binary-Search-Trees: 118,349,998
Total Number of LEAFs in Binary-Search-Trees(GREATER THE BETTER): 7,997,033
Perfectly-Balanced-Binary-Search-Tree for MaxNODEs = 89 must have PEAK = 7 = rounding down of integer (1+lb(89))
Binary-Search-Tree(1st out of 1) with MaxNODEs = 89 has PEAK = 28 and LEAFs = 28
Binary-Search-Tree(1st out of 1) with MaxPEAK = '43' has NODEs = 65 and LEAFs = 11
Binary-Search-Tree(1st out of 2) with MaxLEAFs = 28 has NODEs = 78 and PEAK = 24
*/
UINT HashFNV1A_unrolled_Final(char *str, unsigned int wrdlen)
{
//const UINT PRIME = 31;
unsigned int hash = 2166136261;
char * p = str;

/*
// Reduce the number of multiplications by unrolling the loop
for (SIZE_T ndwords = wrdlen / sizeof(DWORD); ndwords; --ndwords) {
    //hash = (hash ^ *(DWORD*)p) * PRIME;
    hash = ((hash ^ *(DWORD*)p)<<5) - (hash ^ *(DWORD*)p);

    p += sizeof(DWORD);
}
*/
for(; wrdlen >= 4; wrdlen -= 4, p += 4) {
    hash = ((hash ^ *(unsigned int*)p)<<5) - (hash ^ *(unsigned int*)p);
}

// Process the remaining bytes
/*
for (SIZE_T i = 0; i < (wrdlen & (sizeof(DWORD) - 1)); i++) {
    //hash = (hash ^ *p++) * PRIME;
    hash = ((hash ^ *p)<<5) - (hash ^ *p);
    p++;
}
*/
if (wrdlen & -2) {
    hash = ((hash ^ (*(unsigned int*)p&0xFFFF))<<5) - (hash ^ (*(unsigned int*)p&0xFFFF));
    p++;p++;
}
if (wrdlen & 1)
    hash = ((hash ^ *p)<<5) - (hash ^ *p);

return ((hash>>16) ^ hash) & 8191;
}

/*
Results for 'Sixtinsensitive':
Bytes per second performance: 19,808,709B/s
Words per second performance: 1,679,585W/s
Input File with a list of TEXTual Files: Leprechaun_vs_Wikipedia_LATIN-WORDS.lst
Size of all TEXTual Files: 415,982,896
Word count: 35,271,297 of them 22,202,980 distinct
Number Of Files: 8
Number Of Lines: 35271297
Allocated memory in MB: 1950
Number Of Trees(GREATER THE BETTER): 3531949
Forest population(Hash Function Quality regarding Collisions i.e. Hash Table Utilization): 53%
Number Of Hash Collisions(Distinct WORDs - Number Of Trees): 18671031
Maximum Attempts to Find/Put a WORD into a Binary-Search-Tree: '38'
Total Attempts to Find/Put WORDs into Binary-Search-Trees: 118,959,016
Total Number of LEAFs in Binary-Search-Trees(GREATER THE BETTER): 8,047,983
Perfectly-Balanced-Binary-Search-Tree for MaxNODEs = 98 must have PEAK = 7 = rounding down of integer (1+lb(98))
```



```
Binary-Search-Tree(1st out of 1) with MaxNODEs = 98 has PEAK = 36 and LEAFs = 30
Binary-Search-Tree(1st out of 1) with MaxPEAK = '38' has NODEs = 54 and LEAFs = 11
Binary-Search-Tree(1st out of 2) with MaxLEAFs = 30 has NODEs = 98 and PEAK = 36
*/
// Tuned for lowercase-and-uppercase letters i.e. 26 ASCII symbols 65-90 and 97-122 decimal.
UINT Sixtinsensitive(const char *str, unsigned int wrdlen)
{
UINT hash = 2166136261;
UINT hashBUFFER_EAX, hashBUFFER_BH, hashBUFFER_BL;
const char * p = str;

// 0x41 = 065 'A' 010 [0 0001]
// 0x5A = 090 'Z' 010 [1 1010]
// 0x61 = 097 'a' 011 [0 0001]
// 0x7A = 122 'z' 011 [1 1010]

// Reduce the number of multiplications by unrolling the loop
for(; wrdlen >= 6; wrdlen -= 6, p += 6) {
    //hashBUFFER_AX = (*(DWORD*)(p+0)&0xFFFF);
    hashBUFFER_EAX = (*(DWORD*)(p+0)&0x1F1F1F1F);
    hashBUFFER_BL = (*(p+4)&0x1F);
    hashBUFFER_BH = (*(p+5)&0x1F);
    //6bytes-in-4bytes or 48bits-to-30bits
    // Two times next:
    //3bytes-in-2bytes or 24bits-to-15bits
    //EAX                       BL          BH
    //[5bit][3bit][5bit][3bit][5bit][3bit][5bit][3bit]
    //       5th[0..15]  13th[0..15]
    //       BL lower 3  BL higher 2bits
    // OR or XOR no difference
    hashBUFFER_EAX = hashBUFFER_EAX ^ ((hashBUFFER_BL&0x07)<<5); // BL lower 3bits of 5bits
    hashBUFFER_EAX = hashBUFFER_EAX ^ ((hashBUFFER_BL&0x18)<<(2+8)); // BL higher 2bits of 5bits
    hashBUFFER_EAX = hashBUFFER_EAX ^ ((hashBUFFER_BH&0x07)<<(5+16)); // BH lower 3bits of 5bits
    hashBUFFER_EAX = hashBUFFER_EAX ^ ((hashBUFFER_BH&0x18)<<((2+8)+16)); // BH higher 2bits of 5bits
    //hash = (hash ^ hashBUFFER_EAX)*1607; //What a mess: <<7 becomes imul but <<5 not!?
    hash = ((hash ^ hashBUFFER_EAX)<<5) - (hash ^ hashBUFFER_EAX);
    //1607:[2118599]
    // 127:[2121081]
    //  31:[2139242]
    //  17:[2150803]
    //   7:[2166336]
    //   5:[2183044]
    //8191:[2200477]
    //   3:[2205095]
    // 257:[2206188]
}
// Post-Variant #1:
for(; wrdlen; wrdlen--, p++) {
    hash = ((hash ^ (*p&0x1F))<<5) - (hash ^ (*p&0x1F));
}
/*
// Post-Variant #2:
for(; wrdlen >= 2; wrdlen -= 2, p += 2) {
    hash = ((hash ^ (*(DWORD*)p&0xFFFF))<<5) - (hash ^ (*(DWORD*)p&0xFFFF));
}
if (wrdlen & 1)
    hash = ((hash ^ *p)<<5) - (hash ^ *p);
*/
/*
// Post-Variant #3:
for(; wrdlen >= 4; wrdlen -= 4, p += 4) {
    hash = ((hash ^ *(DWORD*)p)<<5) - (hash ^ *(DWORD*)p);
}
if (wrdlen & -2) {
    hash = ((hash ^ (*(DWORD*)p&0xFFFF))<<5) - (hash ^ (*(DWORD*)p&0xFFFF));
    p++;p++;
}
if (wrdlen & 1)
    hash = ((hash ^ *p)<<5) - (hash ^ *p);
*/
return ((hash>>16) ^ hash) & 8191;
}

/*
#define FNV1_32_INIT ((UINT)2166136261)
#define FNV1_32_PRIME ((UINT)1709)

#define FNV_32A_OP(hash, octet) \
    (((UINT)(hash) ^ (unsigned char)(octet)) * FNV1_32_PRIME)

#define FNV_32A_OP32(hash, octet) \
    (((UINT)(hash) ^ (UINT)(octet)) * FNV1_32_PRIME)

UINT FNV1A_Hash_WHIZ(const char *str, SIZE_T wrdlen)
{

UINT hash32;
const char *p;
```

```
2357
2358 hash32 = FNV1_32_INIT;
2359 p=str;
2360
2361 for(; wrdlen >= 4; wrdlen -= 4, p += 4) {
2362 hash32 = FNV_32A_OP32(hash32, (UINT)*(UINT *)p);
2363 }
2364 if (wrdlen & -2) {
2365     hash32 = FNV_32A_OP32(hash32, *(UINT*)p&0xFFFF);
2366     p++;p++;
2367 }
2368 if (wrdlen & 1)
2369     hash32 = FNV_32A_OP(hash32, *p);
2370
2371 return hash32 ^ (hash32 >> 16);
2372 }
2373 */
2374
2375 /*
2376 Results for 'FNV1A_Hash_Jester':
2377 Bytes per second performance: 19,808,709B/s
2378 Words per second performance: 1,679,585W/s
2379 Input File with a list of TEXTual Files: Leprechaun_vs_Wikipedia_LATIN-WORDS.lst
2380 Size of all TEXTual Files: 415,982,896
2381 Word count: 35,271,297 of them 22,202,980 distinct
2382 Number Of Files: 8
2383 Number Of Lines: 35271297
2384 Allocated memory in MB: 1950
2385 Number Of Trees(GREATER THE BETTER): 3537352
2386 Forest population(Hash Function Quality regarding Collisions i.e. Hash Table Utilization): 53%
2387 Number Of Hash Collisions(Distinct WORDs - Number Of Trees): 18665628
2388 Maximum Attempts to Find/Put a WORD into a Binary-Search-Tree: '37'
2389 Total Attempts to Find/Put WORDs into Binary-Search-Trees: 117,243,563
2390 Total Number of LEAFs in Binary-Search-Trees(GREATER THE BETTER): 8,063,361
2391 Perfectly-Balanced-Binary-Search-Tree for MaxNODEs = 87 must have PEAK = 7 = rounding down of integer (1+lb(87))
2392 Binary-Search-Tree(1st out of 2) with MaxNODEs = 87 has PEAK = 27 and LEAFs = 23
2393 Binary-Search-Tree(1st out of 1) with MaxPEAK = '37' has NODEs = 66 and LEAFs = 18
2394 Binary-Search-Tree(1st out of 3) with MaxLEAFs = 27 has NODEs = 84 and PEAK = 27
2395 */
2396 UINT FNV1A_Hash_Jester(const char *str, unsigned int wrdlen)
2397 {
2398 const UINT PRIME = 709607;
2399 UINT hash32 = 2166136261;
2400 const char *p = str;
2401
2402 // Idea comes from Igor Pavlov's 7zCRC, thanks.
2403 /*
2404 for(; wrdlen && ((unsigned)(ptrdiff_t)p&3); wrdlen -= 1, p++) {
2405     hash32 = (hash32 ^ *p) * PRIME;
2406 }
2407 */
2408 for(; wrdlen >= 2*sizeof(DWORD); wrdlen -= 2*sizeof(DWORD), p += 2*sizeof(DWORD)) {
2409     hash32 = (hash32 ^ *(DWORD *)p) * PRIME;
2410     hash32 = (hash32 ^ *(DWORD *)(p+4)) * PRIME;
2411 }
2412 // Cases: 0,1,2,3,4,5,6,7
2413 if (wrdlen & sizeof(DWORD)) {
2414     hash32 = (hash32 ^ *(DWORD*)p) * PRIME;
2415     p += sizeof(DWORD);
2416 }
2417 if (wrdlen & sizeof(WORD)) {
2418     hash32 = (hash32 ^ *(WORD*)p) * PRIME;
2419     p += sizeof(WORD);
2420 }
2421 if (wrdlen & 1)
2422     hash32 = (hash32 ^ *p) * PRIME;
2423
2424 return (hash32 ^ (hash32 >> 16)) & 8191;
2425 }
2426
2427 /*
2428 Results for 'FNV1A_Hash_Jesteress':
2429 Bytes per second performance: 19,808,709B/s
2430 Words per second performance: 1,679,585W/s
2431 Input File with a list of TEXTual Files: Leprechaun_vs_Wikipedia_LATIN-WORDS.lst
2432 Size of all TEXTual Files: 415,982,896
2433 Word count: 35,271,297 of them 22,202,980 distinct
2434 Number Of Files: 8
2435 Number Of Lines: 35271297
2436 Allocated memory in MB: 1950
2437 Number Of Trees(GREATER THE BETTER): 3537293
2438 Forest population(Hash Function Quality regarding Collisions i.e. Hash Table Utilization): 53%
2439 Number Of Hash Collisions(Distinct WORDs - Number Of Trees): 18665687
2440 Maximum Attempts to Find/Put a WORD into a Binary-Search-Tree: '40'
2441 Total Attempts to Find/Put WORDs into Binary-Search-Trees: 117,526,680
2442 Total Number of LEAFs in Binary-Search-Trees(GREATER THE BETTER): 8,051,512
2443 Perfectly-Balanced-Binary-Search-Tree for MaxNODEs = 89 must have PEAK = 7 = rounding down of integer (1+lb(89))
2444 Binary-Search-Tree(1st out of 1) with MaxNODEs = 89 has PEAK = 25 and LEAFs = 23
```



```
2445 Binary-Search-Tree(1st out of 1) with MaxPEAK = '40' has NODEs = 49 and LEAFs = 8
2446 Binary-Search-Tree(1st out of 1) with MaxLEAFs = 28 has NODEs = 72 and PEAK = 21
2447 */
2448 #define ROL(x, n) (((x) << (n)) | ((x) >> (32-(n))))
2449 UINT FNV1A_Hash_Jesteress(const char *str, unsigned int wrdlen)
2450 {
2451 const UINT PRIME = 709607;
2452 UINT hash32 = 2166136261;
2453 const char *p = str;
2454
2455 // Idea comes from Igor Pavlov's 7zCRC, thanks.
2456 /*
2457 for(; wrdlen && ((unsigned)(ptrdiff_t)p&3); wrdlen -= 1, p++) {
2458     hash32 = (hash32 ^ *p) * PRIME;
2459 }
2460 */
2461 for(; wrdlen >= 2*sizeof(DWORD); wrdlen -= 2*sizeof(DWORD), p += 2*sizeof(DWORD)) {
2462     hash32 = (hash32 ^ (ROL(*(DWORD *)p,5)^*(DWORD *)(p+4))) * PRIME;
2463 }
2464 // Cases: 0,1,2,3,4,5,6,7
2465 if (wrdlen & sizeof(DWORD)) {
2466     hash32 = (hash32 ^ *(DWORD*)p) * PRIME;
2467     p += sizeof(DWORD);
2468 }
2469 if (wrdlen & sizeof(WORD)) {
2470     hash32 = (hash32 ^ *(WORD*)p) * PRIME;
2471     p += sizeof(WORD);
2472 }
2473 if (wrdlen & 1)
2474     hash32 = (hash32 ^ *p) * PRIME;
2475
2476 return (hash32 ^ (hash32 >> 16)) & 8191;
2477 }
2478
2479 UINT FNV1A_Hash_Jesteress_27bit(const char *str, unsigned int wrdlen)
2480 {
2481 const UINT PRIME = 709607;
2482 UINT hash32 = 2166136261;
2483 const char *p = str;
2484
2485 // Idea comes from Igor Pavlov's 7zCRC, thanks.
2486 /*
2487 for(; wrdlen && ((unsigned)(ptrdiff_t)p&3); wrdlen -= 1, p++) {
2488     hash32 = (hash32 ^ *p) * PRIME;
2489 }
2490 */
2491 for(; wrdlen >= 2*sizeof(DWORD); wrdlen -= 2*sizeof(DWORD), p += 2*sizeof(DWORD)) {
2492     hash32 = (hash32 ^ (ROL(*(DWORD *)p,5)^*(DWORD *)(p+4))) * PRIME;
2493 }
2494 // Cases: 0,1,2,3,4,5,6,7
2495 if (wrdlen & sizeof(DWORD)) {
2496     hash32 = (hash32 ^ *(DWORD*)p) * PRIME;
2497     p += sizeof(DWORD);
2498 }
2499 if (wrdlen & sizeof(WORD)) {
2500     hash32 = (hash32 ^ *(WORD*)p) * PRIME;
2501     p += sizeof(WORD);
2502 }
2503 if (wrdlen & 1)
2504     hash32 = (hash32 ^ *p) * PRIME;
2505
2506 return (hash32 ^ (hash32 >> 16)) & ((1<<HashInBITS)-1);
2507 }
2508
2509 /*
2510 UINT NextPowerOfTwo(UINT x) {
2511 // Henry Warren, "Hacker's Delight", ch. 3.2
2512 x--;
2513 x |= (x >> 1);
2514 x |= (x >> 2);
2515 x |= (x >> 4);
2516 x |= (x >> 8);
2517 x |= (x >> 16);
2518 return x + 1;
2519 }
2520
2521 UINT NextLog2(UINT x) {
2522 // Henry Warren, "Hacker's Delight", ch. 5.3
2523 if(x <= 1) return x;
2524 x--;
2525 UINT n = 0;
2526 UINT y;
2527 y = x >>16; if(y) {n += 16; x = y;}
2528 y = x >> 8; if(y) {n +=  8; x = y;}
2529 y = x >> 4; if(y) {n +=  4; x = y;}
2530 y = x >> 2; if(y) {n +=  2; x = y;}
2531 y = x >> 1; if(y) return n + 2;
2532 return n + x;
```

```
}
*/

// The following example code in the C language computes the binary logarithm (rounding down) of an integer, rounded down. [2] The operator '>>' represents 'unsigned right shift'. The rounding down form of binary logarithm is identical to computing the position of the most significant 1 bit.
/**
 * Returns the floor form of binary logarithm for a 32 bit integer.
 * -1 is returned if n is 0.
 */
int floorLog2(unsigned int n) {
  int pos = 0;
  if (n >= 1<<16) { n >>= 16; pos += 16; }
  if (n >= 1<< 8) { n >>=  8; pos +=  8; }
  if (n >= 1<< 4) { n >>=  4; pos +=  4; }
  if (n >= 1<< 2) { n >>=  2; pos +=  2; }
  if (n >= 1<< 1) {           pos +=  1; }
  return ((n == 0) ? (-1) : pos);
}

// QuickSortExternal_4+GB.c [

int strcmpKAZE13 (
        const char * src,
        const char * dst
        )
{
        int ret = 0 ;

        while( ! (ret = *(unsigned char *)src - *(unsigned char *)dst) && (*dst!=13-13))
                ++src, ++dst;

        if ( ret < 0 )
                ret = -1 ;
        else if ( ret > 0 )
                ret = 1 ;

        return( ret );
}

//#define LongestLineInclusive 51 //31 former, CAUTION: for command line options 'x' and 'y' it cannot be other than 31 [YET]!

#ifdef singleton
#define LongestLineInclusive 31
#endif
#ifdef doubleton
#define LongestLineInclusive 41
#endif
#ifdef tripleton
#define LongestLineInclusive 41
#endif
#ifdef quadrupleton
#define LongestLineInclusive 51
#endif
#ifdef quintupleton
#define LongestLineInclusive 61
#endif
#ifdef sextupleton
#define LongestLineInclusive 71
#endif
#ifdef septupleton
#define LongestLineInclusive 81
#endif
#ifdef octupleton
#define LongestLineInclusive 91
#endif
#ifdef nonupleton
#define LongestLineInclusive 101
#endif
#ifdef decupleton
#define LongestLineInclusive 111
#endif

// _ngram_  1  1-31
// _ngram_  2  5-41
// _ngram_  3  9-41
// _ngram_  4 13-51
// _ngram_  5 17-61
// _ngram_  6 21-71
// _ngram_  7 25-81
// _ngram_  8 29-91
// _ngram_  9 33-101
// _ngram_ 10 37-111
// For Leaf of 256bytes LongestLineInclusive should be 256 = 8+8+8+2*(LongestLineInclusive+1+4) or LongestLineInclusive = (256 - (8+8+8) - 2*(1+4))/2 = 111

char FourGramL[LongestLineInclusive+1+4]; // 31bytes longest 4-gram + 1byte NULL + 4bytes COUNTER
char FourGramR[LongestLineInclusive+1+4]; // 31bytes longest 4-gram + 1byte NULL + 4bytes COUNTER
```



```
char LEAF[8+8+8+2*(LongestLineInclusive+1+4)]; // 136bytes = 3 pointers + 2 keys
char LEAFNEW[8+8+8+2*(LongestLineInclusive+1+4)]; // 136bytes = 3 pointers + 2 keys
FILE *fp_outRG; // Global - not to burden the extract/compare function with one more parameter
int CompareStringsEndingWith13_EXTERNAL(unsigned long long AtPosition64L, unsigned long long AtPosition64R) {

int i;
unsigned long long *AtPosition64Lpointer=&AtPosition64L;
unsigned long long *AtPosition64Rpointer=&AtPosition64R;

// Caramba: seek and tell report OK but in fact they lie, only setpos works?!?!?!

//#if defined(_WIN32_ENVIRONMENT_)
//_lseeki64( fileno(fp_outRG), AtPosition64L, 0 );
//#else
//fseeko( fp_outRG, AtPosition64L, SEEK_SET );
//#endif /* defined(_WIN32_ENVIRONMENT_)  */

//  _CRTIMP __int64 __cdecl _telli64(int);
//  off64_t ftello64 (FILE *stream)


fsetpos(fp_outRG, AtPosition64Lpointer);
for (i=0; i<(LongestLineInclusive+1+4); i++) {fread(&FourGramL[i], 1, 1, fp_outRG); if (FourGramL[i]==13-13) break;}
//Commented line below is slower than the one above: 778156 clocks vs 756297 clocks.
//fread(&FourGramL[0], 31+1, 1, fp_outRG);

fsetpos(fp_outRG, AtPosition64Rpointer);
for (i=0; i<(LongestLineInclusive+1+4); i++) {fread(&FourGramR[i], 1, 1, fp_outRG); if (FourGramR[i]==13-13) break;}
//Commented line below is slower than the one above: 778156 clocks vs 756297 clocks.
//fread(&FourGramR[0], 31+1, 1, fp_outRG);

return(strcmpKAZE13(FourGramL, FourGramR));
}

int CompareStringsEndingWith13_INTERNAL(unsigned long long AtPosition64L, unsigned long long AtPosition64R, char *POOLinternal) {

int i;
//char FourGramL[LongestLineInclusive+2]; // 31 longest 4-gram + CR + LF
//char FourGramR[LongestLineInclusive+2]; // 31 longest 4-gram + CR + LF

for (i=0; i<(LongestLineInclusive+1+4); i++) {
//fread(&FourGramL[i], 1, 1, fp_in);
FourGramL[i] = *(char *)(POOLinternal + AtPosition64L);
if (FourGramL[i]==13-13) break;
}

for (i=0; i<(LongestLineInclusive+1+4); i++) {
//fread(&FourGramR[i], 1, 1, fp_in);
FourGramR[i] = *(char *)(POOLinternal + AtPosition64R);
if (FourGramR[i]==13-13) break;
}

return(strcmpKAZE13(FourGramL, FourGramR));
}

// QuickSortExternal_4+GB.c ]


int main( argc, argv )
int argc; char *argv[];
{
   int nlines;
   string *backup = NULL;

      FILE *fp_in, *fp_out, *fp_outLOG, *fp_inLINE;
      int LetterOffset;
      unsigned long long FilesLEN;
      unsigned long long WORDcount;
      unsigned long long WORDcountBOTTOM;
      unsigned long long WORDcountAttemptsToPut;
      int Thunderwith;
        unsigned long NumberOfFiles, WORDcountDistinct, WORDcountDistinctTOTAL = 0, TotalMemoryNeededForOnePass = 0;
        unsigned long long NumberOfLines;  // rev. 12+
      unsigned long WHOLEletter_BufferSize;
      unsigned long long WHOLEletter_BufferSize_L14;
        unsigned long memory_size, LetterBuffer, j, k, LINE1Olen, wrdlen;
        unsigned long k_FIX;
        unsigned long long i;              // rev. 12+
        //unsigned long size_in, size_out, size_inLINE;
        unsigned long size_in;             // rev. 12+
#if defined(_WIN32_ENVIRONMENT_)
      unsigned long long size_inLINESIXFOUR;
#else
      size_t size_inLINESIXFOUR;
#endif /* defined(_WIN32_ENVIRONMENT_)  */

      //unsigned long t1, t2, t3;
      time_t t1, t2, t3, t4,tMainB,tMainE;
```

```
2706
2707       const int NumberOfSLOTs = 4096*2; // Since r.12+ in rev.12 it was 4096
2708   unsigned long StackPtr;
2709   //unsigned long BSTstack [65536*3]; // BST in worst case could become a LL.
2710   unsigned long long BSTstack [8192*3]; // BST in worst case could become a LL.
2711 unsigned long NumberOfTrees=0, NumberOfHashCollisions=0;
2712 unsigned long iBSTwithMAXpeak, jBSTwithMAXpeak;
2713   unsigned int PEAKibBST;
2714   unsigned long BSTsTotalLEAFs=0; // ?! MADHOUSE: if BSTcurrent is not zeroed here then INSANE values of BSTcurrent occur below where 'break'
     is ?!
2715   unsigned long BSTwithMAXnode=0, BSTcurrentNode=0; // ?! MADHOUSE: if BSTcurrent is not zeroed here then INSANE values of BSTcurrent occur
     below where 'break' is ?!
2716   unsigned long                   BSTcurrentNodeMAXqUANTITY=0; // ?! MADHOUSE: if BSTcurrent is not zeroed here then INSANE values of
     BSTcurrent occur below where 'break' is ?!
2717   unsigned long BSTwithMAXnodePEAK=1, BSTwithMAXnodeLEAF=1; // ?! MADHOUSE: if BSTcurrent is not zeroed here then INSANE values of BSTcurrent
     occur below where 'break' is ?!
2718   unsigned long BSTwithMAXpeak=0, BSTcurrentPeak=0; // ?! MADHOUSE: if BSTcurrent is not zeroed here then INSANE values of BSTcurrent occur
     below where 'break' is ?!
2719   unsigned long                   BSTcurrentPeakMAX=0; // ?! MADHOUSE: if BSTcurrent is not zeroed here then INSANE values of BSTcurrent occur
     below where 'break' is ?!
2720   unsigned long                   BSTcurrentPeakMAXqUANTITY=0; // ?! MADHOUSE: if BSTcurrent is not zeroed here then INSANE values of
     BSTcurrent occur below where 'break' is ?!
2721   unsigned long BSTwithMAXpeakNODE=1, BSTwithMAXpeakLEAF=1; // ?! MADHOUSE: if BSTcurrent is not zeroed here then INSANE values of BSTcurrent
     occur below where 'break' is ?!
2722   unsigned long BSTwithMAXleaf=0, BSTcurrentLeaf=0; // ?! MADHOUSE: if BSTcurrent is not zeroed here then INSANE values of BSTcurrent occur
     below where 'break' is ?!
2723   unsigned long                   BSTcurrentLeafMAXqUANTITY=0; // ?! MADHOUSE: if BSTcurrent is not zeroed here then INSANE values of
     BSTcurrent occur below where 'break' is ?!
2724   unsigned long BSTwithMAXleafNODE=1, BSTwithMAXleafPEAK=1; // ?! MADHOUSE: if BSTcurrent is not zeroed here then INSANE values of BSTcurrent
     occur below where 'break' is ?!
2725
2726       char *pointerflush, *pointerflushUNALIGN, *BufStart, *Flushing;
2727       char *pointerflush_64, *pointerflushUNALIGN_64; // r.14++
2728       unsigned long PseudoLinkedPointer, PseudoLinkedPointerNEW, PseudoLinkedPointerROOT, PseudoLinkedPointerNEWold;
2729       unsigned long PseudoLinkedPointerNEWleft, PseudoLinkedPointerNEWright;
2730       unsigned long PseudoLinkedPointerNEWmiddle;
2731       char *bufend[ 806 ]; // 'a'=0, ... 'z'=25 - 26 letters x 31 lengths
2732       long bufNumberOfWords[ 806 ]; // 'a'=0, ... 'z'=25 - 26 letters x 31 lengths
2733       // long bufNoWpS[ 806 ][ 8192 ]; // ?! crashes below when an attempt to use it occur
2734       char wrd[LongestLineInclusive+1+4];  // 0..30, 31 = 0
2735       char wrdUP[LongestLineInclusive+1+4];  // 0..30, 31 = 0
2736       char wrdUPold[LongestLineInclusive+1+4];  // 0..30, 31 = 0
2737       char LINE10[257]; // 000..255, 256 = 0
2738       char ZEROS[4]; // 0..3, 0 = 0, 1 = 0, 2 = 0, 3 = 0
2739       char CRdLFa[2];  // 0..1, 0 = 13, 1 = 10
2740       char workbyte;
2741       char workK[1024*128];
2742       long workKoffset = -1;
2743       int FoundInLinkedList, Slot;
2744       unsigned long OffsetsInBuffer[31]; // 00..30
2745       unsigned long MAXusedBuffer[32]; // 00 not used, only 01..31
2746       unsigned long GRMBLhill[32]; // 00..31
2747       unsigned long GRMBLFoolAgain[32]; // 00..31
2748       int Melnitchka;
2749       unsigned long MAXusedBufferABS = 0;
2750       unsigned long Utiliza1 = 0;
2751       unsigned long Utiliza2 = 0;
2752       unsigned long TotalWLchars = 0;
2753
2754 /* minimum signed 64 bit value */
2755 #define _I64_MIN    (-9223372036854775807i64 - 1)
2756 /* maximum signed 64 bit value */
2757 #define _I64_MAX      9223372036854775807i64
2758 /* maximum unsigned 64 bit value */
2759 #define _UI64_MAX     0xffffffffffffffffui64
2760
2761 /* minimum signed 128 bit value */
2762 #define _I128_MIN   (-170141183460469231731687303715884105727i128 - 1)
2763 /* maximum signed 128 bit value */
2764 #define _I128_MAX     170141183460469231731687303715884105727i128
2765 /* maximum unsigned 128 bit value */
2766 #define _UI128_MAX    0xffffffffffffffffffffffffffffffffui128
2767
2768       char llTOaDigits[27]; // 9,223,372,036,854,775,807: 1(sign or carry)+19(digits)+1('\0')+6(,)
2769       // below duplicates are needed because of one_line_invoking need different buffers.
2770       char llTOaDigits2[27]; // 9,223,372,036,854,775,807: 1(sign or carry)+19(digits)+1('\0')+6(,)
2771       char llTOaDigits3[27]; // 9,223,372,036,854,775,807: 1(sign or carry)+19(digits)+1('\0')+6(,)
2772       char llTOaDigits4[27]; // 9,223,372,036,854,775,807: 1(sign or carry)+19(digits)+1('\0')+6(,)
2773       unsigned long HEADOffetFromStartBUKVA = 0;
2774       unsigned long TAILOffetFromStartBUKVA = 0;
2775       int BSTorBtree = 0;
2776       int SplitOccured;
2777       int POffsetInLEAF;
2778 char *Auberge[4] = {"|\0","/\0","-\0","\\\0"};
2779 int hashAlfalfa, iAlfalfa;
2780 int PLE_words=0; // Quadruple!
2781 char wrd1st[LongestLineInclusive+1+4];  // 0..30, 31 = 0
2782 char wrd2nd[LongestLineInclusive+1+4];  // 0..30, 31 = 0
```



```
2783 char wrd3rd[LongestLineInclusive+1+4];  // 0..30, 31 = 0
2784 char wrd4th[LongestLineInclusive+1+4];  // 0..30, 31 = 0
2785 char wrd5th[LongestLineInclusive+1+4];  // 0..30, 31 = 0
2786 char wrd6th[LongestLineInclusive+1+4];  // 0..30, 31 = 0
2787 char wrd7th[LongestLineInclusive+1+4];  // 0..30, 31 = 0
2788 char wrd8th[LongestLineInclusive+1+4];  // 0..30, 31 = 0
2789 char wrd9th[LongestLineInclusive+1+4];  // 0..30, 31 = 0
2790 char wrd10th[LongestLineInclusive+1+4];  // 0..30, 31 = 0
2791 char *DelimiterUnderscore = "_\0";
2792 int PLE_words_INITflag = 0;
2793
2794 // QuickSortExternal_4+GB [
2795 unsigned long long ThunderwithL64_L14;
2796 unsigned long long Strnglen64_L14;
2797 unsigned long long size_in64_L14, size_in2_L14;
2798 unsigned long long Over4billionLines, j_Over4billion;
2799 char OneChar_ieByte = '\0';
2800 char CR_ieByte = '\r';
2801 char SomeByte;
2802 unsigned long long BufEnd_64;
2803 unsigned long long SeekPosition;
2804 unsigned long long *PointerToSeekPosition;
2805 char FourGram[LongestLineInclusive+2]; // 31 longest 4-gram + CR + LF
2806 char *PoolPhysical;
2807 unsigned long long fsetpos_ZERO=0;
2808 char OneCkusterZEROES[1024*4]; // Caution: must be ZEROed(NULLified)!
2809 char *FileSwapTag = "LEPRECHAUNISH";
2810 char EOFcode = 0x1A;
2811 unsigned long long PseudoLinkedPointer_64, PseudoLinkedPointerNEW_64, PseudoLinkedPointerROOT_64, PseudoLinkedPointerNEWold_64;
2812       unsigned long long PseudoLinkedPointerNEWleft_64, PseudoLinkedPointerNEWright_64;
2813       unsigned long long PseudoLinkedPointerNEWmiddle_64;
2814       unsigned long long NULLs_64 = 0;
2815 unsigned long long PseudoLinkedPointerAUX_64;
2816 unsigned long long PseudoLinkedPointerAUXdumbo_64;
2817       char wrdAUX[LongestLineInclusive+1+4];  // 0..30, 31 = 0
2818 // QuickSortExternal_4+GB ]
2819
2820 unsigned long CounterOccurrencies;
2821 unsigned long long NumberOfLEAFs=0;
2822 unsigned long LevelsInCorona_Not_Counting_ROOT=0;
2823 char *ngram[11] =
    {"NULLleton\0","singleton\0","doubleton\0","tripleton\0","quadrupleton\0","quintupleton\0","sextupleton\0","septupleton\0","octupleton\0","non
    upleton\0","decupleton\0"};
2824
2825 unsigned long RipPasses;
2826 unsigned long long NULLsForWRD=0;
2827
2828 //15+
2829 int DoNotInsertFlag = 0;
2830 int METACOMMANDflag;
2831
2832 //16
2833 int REUSE=0;
2834 int HSHexist;
2835 int SWPexist;
2836
2837 // INIT INIT INIT INIT INIT INIT INIT INIT INIT INIT INIT INIT INIT INIT INIT
2838 printf ("Leprechaun_%s (Fast-In-Future Greedy n-gram-Ripper), rev. 16FIX, written by Svalqyatchx.\n", ngram[_ngram_]);
2839 //puts( "Leprechaun: 'Oh, well, didn't you hear? Bigger is good, but jumbo is dear.'" );
2840
2841 #ifdef singleton
2842 printf ("Purpose: Rips all distinct %d-grams (%d-word phrases) with length 1..31 chars from incoming texts.\n", _ngram_, _ngram_);
2843 #endif
2844 #ifdef doubleton
2845 printf ("Purpose: Rips all distinct %d-grams (%d-word phrases) with length 5..41 chars from incoming texts.\n", _ngram_, _ngram_);
2846 #endif
2847 #ifdef tripleton
2848 printf ("Purpose: Rips all distinct %d-grams (%d-word phrases) with length 9..41 chars from incoming texts.\n", _ngram_, _ngram_);
2849 #endif
2850 #ifdef quadrupleton
2851 printf ("Purpose: Rips all distinct %d-grams (%d-word phrases) with length 13..51 chars from incoming texts.\n", _ngram_, _ngram_);
2852 #endif
2853 #ifdef quintupleton
2854 printf ("Purpose: Rips all distinct %d-grams (%d-word phrases) with length 17..61 chars from incoming texts.\n", _ngram_, _ngram_);
2855 #endif
2856 #ifdef sextupleton
2857 printf ("Purpose: Rips all distinct %d-grams (%d-word phrases) with length 21..71 chars from incoming texts.\n", _ngram_, _ngram_);
2858 #endif
2859 #ifdef septupleton
2860 printf ("Purpose: Rips all distinct %d-grams (%d-word phrases) with length 25..81 chars from incoming texts.\n", _ngram_, _ngram_);
2861 #endif
2862 #ifdef octupleton
2863 printf ("Purpose: Rips all distinct %d-grams (%d-word phrases) with length 29..91 chars from incoming texts.\n", _ngram_, _ngram_);
2864 #endif
2865 #ifdef nonupleton
2866 printf ("Purpose: Rips all distinct %d-grams (%d-word phrases) with length 33..101 chars from incoming texts.\n", _ngram_, _ngram_);
2867 #endif
2868 #ifdef decupleton
```

```
2869 printf ("Purpose: Rips all distinct %d-grams (%d-word phrases) with length 37..111 chars from incoming texts.\n", _ngram_, _ngram_);
2870 #endif
2871 puts( "Feature1: All words within x-lets/n-grams are in range 1..31 chars inclusive." );
2872 //puts( "Feature2: In this revision 128MB 1-way hash is used which results in 16,777,216 external B-Trees of order 3." );
2873
2874 if (HashInBITS+3<10)
2875 printf ("Feature2: In this revision %sbytes 1-way hash is used which results in %s external B-Trees of order 3.\n",
     _ui64toaKAZEcomma(((1<<HashInBITS)<<3), llTOaDigits, 10), _ui64toaKAZEcomma((1<<HashInBITS), llTOaDigits2, 10) );
2876 else if (HashInBITS+3>=10 && HashInBITS+3<20)
2877 printf ("Feature2: In this revision %sKB 1-way hash is used which results in %s external B-Trees of order 3.\n", _ui64toaKAZEcomma(
     (((1<<HashInBITS)<<3))>>10, llTOaDigits, 10), _ui64toaKAZEcomma((1<<HashInBITS), llTOaDigits2, 10) );
2878 else
2879 printf ("Feature2: In this revision %sMB 1-way hash is used which results in %s external B-Trees of order 3.\n", _ui64toaKAZEcomma(
     (((1<<HashInBITS)<<3))>>20, llTOaDigits, 10), _ui64toaKAZEcomma((1<<HashInBITS), llTOaDigits2, 10) );
2880 if (HashInBITS-HashChunkSizeInBITS==0)
2881 printf ("Feature3: In this revision %s pass is to be made.\n", _ui64toaKAZEcomma(1<<(HashInBITS-HashChunkSizeInBITS), llTOaDigits, 10));
2882 else
2883 printf ("Feature3: In this revision %s passes are to be made.\n", _ui64toaKAZEcomma(1<<(HashInBITS-HashChunkSizeInBITS), llTOaDigits, 10));
2884
2885 puts( "Feature4: If the external memory has latency 99+microseconds then !(look no further), IOPS(seek-time) rules." );
2886 // The phrase 'look no further' was used in amazon.com review meaning 'stop searching for better thing this is it'.
2887 //puts( "Kaze: Let's see what a 3-way hash + 6,602,752 Binary-Search-Trees can give us," );
2888 //puts( "      also the performance of a 3-way hash + 6,602,752 B-Trees of order 3," );
2889 //puts( "      also the performance of a 1-way hash + 134,217,728 external B-Trees of order 3." );
2890 //puts( "Note1: Compiled with Microsoft C v. 13.10.3077: 'cl /Ox /TcLeprechaun.c'." );
2891 //puts( "Note2: This WORDLISTER makes as output pseudo(unsorted)_wordlist_CRLF_file." );
2892 if( argc != 3 && argc != 4 && argc != 5 && argc != 6 ) // +1 for program name
2893 {
2894   puts( "" );
2895 puts( "'The Little Monster' short notes:" );
2896 puts( "Note1: I wish to thank to R.N. Horspool, Ranjan Sinha, Dmitry Shkarin," );
2897 puts( "       Michael Abrash, J. Bentley, R. Sedgewick, Igor Pavlov, Lasse Reinhold," );
2898 puts( "       Landon Noll, Peter Kankowski for sharing their knowledge to public." );
2899 puts( "Note2: Run it without parameters to get usage and short notes." );
2900 puts( "Note3: This simple amateurish(more over I am not versed well neither in C nor" );
2901 puts( "       in mathematics nor in English language, but I am persistent in INDEXING" );
2902 puts( "       GBs of english TEXTS) tool is written in ANSI C(at least its source is" );
2903 puts( "       compileable for CL(Windows) and GCC(Linux)), and its purpose is to" );
2904 puts( "       create a WordList for a group of files(given via filelist)." );
2905 puts( "       Its name comes(according to Heritage Dictionary) from 'low corpus' or" );
2906 puts( "       'little body', in fact from amazing movie saga 'Leprechaun 1-2-3-4-5-6'" );
2907 puts( "       starring by Warwick Davis." );
2908 puts( "Note4: Only words up to 31 chars are proceeded - the reason is 'DDT'(the" );
2909 puts( "       longest word in Heritage Dictionary 3rd edition) or" );
2910 puts( "       'dichlorodiphenyltrichloroethane'." );
2911 puts( "Note5: Cursor hiding in C - mission impossible for me." );
2912 puts( "Note6: By default(third parameter is 1023) allocated memory is 393MB." );
2913 puts( "       Due to 'malloc()' limitation under WINDOWS, maximum value of third" );
2914 puts( "       parameter is 5174 which is 1988MB allocated block." );
2915 puts( "Note7: File Leprechaun.LOG is a log, where new statistics are appended." );
2916 puts( "Note8: Revision 12+ can handle files larger than 4GB." );
2917 puts( "Note9: Revision 12++ has a buffered 'fread()' - therefore I/O READ-BURST SPEED" );
2918 puts( "       is the first(worst) bottleneck, as a result r.12++ is much-much faster;" );
2919 puts( "       the second(worse) bottleneck: the linked lists - the b-trees" );
2920 puts( "       might be the answer; the third(bad) bottleneck: the amateurish author." );
2921 puts( "NoteA: Revision 12+++ has an improved(2 bits were used doltishly) main hash" );
2922 puts( "       function - therefore less collisions, for example:" );
2923 puts( "       for file 'wikipedia-de-html.tar' 42,291,855,360 bytes with" );
2924 puts( "       5,750,179,678 words of them 7,375,373 distinct attempts to Find/Put" );
2925 puts( "       a WORD into a linked list are 6,117,675,470(r.12++) and 5,845,989,790" );
2926 puts( "       (r.12+++); also two 'if' sections were moved because they were executed" );
2927 puts( "       unnecessarily many times." );
2928 puts( "NoteB: Revision 13 uses BSTs instead of LLs, that is Linked-Lists were" );
2929 puts( "       replaced by Binary-Search-Trees, as a result for 22,202,980 distinct" );
2930 puts( "       words(out of 35,271,297) r.12+++ needs 225,548,268 total attempts to" );
2931 puts( "       Find/Put WORDs into linked lists where r.13 needs 121,674,042 total" );
2932 puts( "       attempts to Find/Put WORDs into Binary-Search-Trees. But this is a" );
2933 puts( "       significant boost in performance only for wordlists of million words." );
2934 puts( "NoteC: Revision 13+ gives only more statistics. Future revisions could lessen" );
2935 puts( "       number of attempts to Find/Put WORDs into Binary-Search-Trees" );
2936 puts( "       furthermore by making them at some point Perfectly-Balanced. But" );
2937 puts( "       for huge amount(multi-(m|b)illion) of distinct words the b-tree family" );
2938 puts( "       must come in, until then this is the leprechaunish niche." );
2939 puts( "NoteD: Revision 13++ has a little fix(2 unnecessary ZEROings, when a new word" );
2940 puts( "       is inserted, were deleted) and a fixed bug(13+ adds stupidly the" );
2941 puts( "       highest BST to the wordlist). Also B-Tree of order 3 is added as a" );
2942 puts( "       searching method. Main goal of B-Tree is to reduce number of" );
2943 puts( "       comparisons but at nasty cost: a precious time wasted to construct it" );
2944 puts( "       and twice more memory, i.e. one step forward two backward: this tree is" );
2945 puts( "       more effective than BST in cases of 2++ billion/million" );
2946 puts( "       different/distinct words." );
2947 puts( "       The improvement which comes from using B-Tree of order 3 is about 200%" );
2948 puts( "       much more pleasing than I expected, for wikipedia-en-html.tar.wrd with" );
2949 puts( "       12,561,874 distinct words Total Attempts to Find/Put WORDs into:" );
2950 puts( "       Binary-Search-Trees was 61,895,043 while for" );
2951 puts( "       B-trees order 3 was 19,295,791." );
2952 puts( "NoteE: Revision 13+++ has a faster(not heavily tested yet) and with" );
2953 puts( "       better(0.6% to 1.1%) dispersion Fowler/Noll/Vo hash," );
```



```
2954 puts( "       so called FNV1a hash. Revision 13+++++ boosting: Leprechaun_Intel.exe" );
2955 puts( "       gives 1,256,187W/s for wikipedia-en-html.tar.wrd with FNV1_32_PRIME:" );
2956 puts( "       107712257 with 3,551,736 dispersion for 'FNV1A_Hash_Granularity'." );
2957 puts( "NoteF: For old r.12+ a USB connected HDD crippled test:" );
2958 puts( "       for 'H:\\>Leprechaun.exe static.wikipedia.org_downloads_2008-06_en.lst" );
2959 puts( "       wikipedia-en-html.tar.wrd 5400'" );
2960 puts( "       where 223,674,511,360 wikipedia-en-html.tar" );
2961 puts( "       on laptop Toshiba Pentium T3400 2166 MHz with" );
2962 puts( "       Motherboard Name:                                Toshiba Satellite L305" );
2963 puts( "       CPU Type:          Mobile DualCore Intel Pentium, 2166 MHz (13 x 167)" );
2964 puts( "       CPU Alias:                                                 Merom-1M" );
2965 puts( "       L1 Code Cache:                                          32 KB per core" );
2966 puts( "       L1 Data Cache:                                          32 KB per core" );
2967 puts( "       L2 Cache:                        1 MB (On-Die, ECC, ASC, Full-Speed)" );
2968 puts( "       Bus Type:                                           Dual DDR2 SDRAM" );
2969 puts( "       Bus Width:                                                128-bit" );
2970 puts( "       Real Clock:                                        333 MHz (DDR)" );
2971 puts( "       Effective Clock:                                           666 MHz" );
2972 puts( "       EVEREST v5.00.1650 Memory Copy:       3725MB/s with timings 5-5-5-13" );
2973 puts( "       result is logged to 'Leprechaun.LOG':" );
2974 puts( "  Bytes per second performance: 20,658,955B/s" );
2975 puts( "  Words per second performance: 2,860,880W/s" );
2976 puts( "  Input File with a list of TEXTual Files:" );
2977 puts( "   static.wikipedia.org_downloads_2008-06_en.lst" );
2978 puts( "  Size of all TEXTual Files: 223,674,511,360" );
2979 puts( "  Word count: 30,974,750,142 of them 12,561,874 distinct" );
2980 puts( "  Number Of Files: 1" );
2981 puts( "  Number Of Lines: 2088618575" );
2982 puts( "  Allocated memory in MB: 1920" );
2983 puts( "  Words with length 01 occupy 0,033KB of 0,349KB given i.e. 09% utilization" );
2984 puts( "  Words with length 02 occupy 0,033KB of 0,349KB given i.e. 09% utilization" );
2985 puts( "  Words with length 03 occupy 0,037KB of 0,697KB given i.e. 05% utilization" );
2986 puts( "  Words with length 04 occupy 0,151KB of 0,871KB given i.e. 17% utilization" );
2987 puts( "  Words with length 05 occupy 0,744KB of 1,568KB given i.e. 47% utilization" );
2988 puts( "  Words with length 06 occupy 1,470KB of 3,136KB given i.e. 46% utilization" );
2989 puts( "  Words with length 07 occupy 2,605KB of 5,923KB given i.e. 43% utilization" );
2990 puts( "  Words with length 08 occupy 3,296KB of 6,968KB given i.e. 47% utilization" );
2991 puts( "  Words with length 09 occupy 3,714KB of 6,968KB given i.e. 53% utilization" );
2992 puts( "  Words with length 10 occupy 3,483KB of 6,968KB given i.e. 49% utilization" );
2993 puts( "  Words with length 11 occupy 3,235KB of 5,923KB given i.e. 54% utilization" );
2994 puts( "  Words with length 12 occupy 2,691KB of 4,181KB given i.e. 64% utilization" );
2995 puts( "  Words with length 13 occupy 2,230KB of 3,484KB given i.e. 64% utilization" );
2996 puts( "  Words with length 14 occupy 1,718KB of 3,484KB given i.e. 49% utilization" );
2997 puts( "  Words with length 15 occupy 1,357KB of 2,613KB given i.e. 51% utilization" );
2998 puts( "  Words with length 16 occupy 1,063KB of 2,613KB given i.e. 40% utilization" );
2999 puts( "  Words with length 17 occupy 0,814KB of 1,742KB given i.e. 46% utilization" );
3000 puts( "  Words with length 18 occupy 0,617KB of 1,742KB given i.e. 35% utilization" );
3001 puts( "  Words with length 19 occupy 0,485KB of 1,742KB given i.e. 27% utilization" );
3002 puts( "  Words with length 20 occupy 0,402KB of 1,742KB given i.e. 23% utilization" );
3003 puts( "  Words with length 21 occupy 0,327KB of 1,742KB given i.e. 18% utilization" );
3004 puts( "  Words with length 22 occupy 0,274KB of 1,742KB given i.e. 15% utilization" );
3005 puts( "  Words with length 23 occupy 0,224KB of 1,394KB given i.e. 16% utilization" );
3006 puts( "  Words with length 24 occupy 0,190KB of 1,394KB given i.e. 13% utilization" );
3007 puts( "  Words with length 25 occupy 0,162KB of 1,394KB given i.e. 11% utilization" );
3008 puts( "  Words with length 26 occupy 0,136KB of 1,220KB given i.e. 11% utilization" );
3009 puts( "  Words with length 27 occupy 0,119KB of 1,046KB given i.e. 11% utilization" );
3010 puts( "  Words with length 28 occupy 0,107KB of 0,871KB given i.e. 12% utilization" );
3011 puts( "  Words with length 29 occupy 0,091KB of 0,697KB given i.e. 13% utilization" );
3012 puts( "  Words with length 30 occupy 0,080KB of 0,523KB given i.e. 15% utilization" );
3013 puts( "  Words with length 31 occupy 0,076KB of 0,523KB given i.e. 14% utilization" );
3014 puts( "  Total pseudo(including hash table) memory utilization: 42%" );
3015 puts( "  Total real(wordlist's words VS allocated block) memory utilization: 60/1000" );
3016 puts( "  Used value for third parameter in KB: 5400" );
3017 puts( "  Use next time as third parameter: 3475-" );
3018 puts( "  Time for making unsorted wordlist: 10827 second(s)" );
3019 puts( "  Time for sorting unsorted wordlist: 10 second(s)" );
3020 puts( "NoteG: 2011-Mar-07: Fixed a small command line parsing bug." );
3021 puts( "NoteH: A heavy blow for my illusions(regarding speed performance of external b-trees)," );
3022 puts( "       desperate results for ripping on HDD 7200rpm:" );
3023 puts( "       20,000,000 distinct 4-grams per 5 hours." );
3024 puts( "       D:\\Leprechaun_quadrupleton_r14_minus>Leprechaun_quadrupleton GRAFFITH_2048.lst GRAFFITH_2048.wrd 48000000 z" );
3025 puts( "       Leprechaun(Fast Greedy Word-Ripper), rev. 14_minus_quadrupleton, written by Svalqyatchx." );
3026 puts( "       Leprechaun: 'Oh, well, didn't you hear? Bigger is good, but jumbo is dear.'" );
3027 puts( "       Kaze: Let's see what a 3-way hash + 6,602,752 Binary-Search-Trees can give us," );
3028 puts( "             also the performance of a 3-way hash + 6,602,752 B-Trees of order 3," );
3029 puts( "             also the performance of a 1-way hash + 134,217,728 external B-Trees of order 3." );
3030 puts( "       Size of input file with files for Leprechauning: 42140" );
3031 puts( "       Allocating HASH memory 1,073,741,889 bytes ... OK" );
3032 puts( "       Allocating/ZEROing 49,152,000,014 bytes swap file ... OK" );
3033 puts( "       Size of Input TEXTual file: 33,470,581" );
3034 puts( "       |; Word count: 3,045,077 of them 2,597,942 distinct; Done: 64/64" );
3035 puts( "       Size of Input TEXTual file: 17,229,900" );
3036 puts( "       -; Word count: 4,235,032 of them 3,588,757 distinct; Done: 64/64" );
3037 puts( "       Size of Input TEXTual file: 19,191,256" );
3038 puts( "       |; Word count: 5,803,400 of them 4,866,213 distinct; Done: 64/64" );
3039 puts( "       Size of Input TEXTual file: 34,651,077" );
3040 puts( "       \\; Word count: 8,714,961 of them 6,941,108 distinct; Done: 64/64" );
3041 puts( "       Size of Input TEXTual file: 26,875,458" );
```

```
3042 puts( "       /; Word count: 11,022,830 of them 8,579,931 distinct; Done: 64/64" );
3043 puts( "       Size of Input TEXTual file: 19,605,129" );
3044 puts( "       -; Word count: 12,924,821 of them 10,078,191 distinct; Done: 64/64" );
3045 puts( "       Size of Input TEXTual file: 17,053,521" );
3046 puts( "       /; Word count: 14,577,010 of them 11,455,983 distinct; Done: 64/64" );
3047 puts( "       Size of Input TEXTual file: 44,087,709" );
3048 puts( "       -; Word count: 18,953,280 of them 15,010,569 distinct; Done: 64/64" );
3049 puts( "       Size of Input TEXTual file: 32,796,705" );
3050 puts( "       |; Word count: 22,412,912 of them 17,621,649 distinct; Done: 64/64" );
3051 puts( "       Size of Input TEXTual file: 19,538,360" );
3052 puts( "       /; Word count: 24,381,005 of them 19,137,701 distinct; Done: 64/64" );
3053 puts( "       Size of Input TEXTual file: 29,565,366" );
3054 puts( "       \\; Word count: 26,214,400 of them 20,528,357 distinct; Done: 40/64" );
3055 puts( "       ..." );
3056 puts( "NoteI: In revision 14- the resultant wordlist is NOT sorted when 'Z' is used." );
3057 puts( "NoteJ: In revision 14 'x' and 'y' options are disabled, for 7++ million phrases their usefulness is no more," );
3058 puts( "       the real loads are of order 800+ million, too many limitations exist, they must be rewritten as 64bit." );
3059 puts( "NoteK: Ripping OSHO.TXT (10,165,640 4-grams) on HDD daunts because of 6+hours needed:" );
3060 puts( "       Number Of Trees(GREATER THE BETTER): 9,433,894" );
3061 puts( "       Used value for third parameter in KB: 3,145,728" );
3062 puts( "       Use next time as third parameter: 1,262,186" );
3063 puts( "       One leaf has size: 8+8+8+(51+1+4)+(51+1+4)=136bytes," );
3064 puts( "       or MAX (one 4-gram per leaf) 10,165,640*136=1,382,527,040bytes." );
3065 puts( "NoteL: Each phrase in extracted file is preceded by TAB ASCII code, this (TAB being a delimiter symbol) allows" );
3066 puts( "       the phrase-list to be ripped again i.e. to treat already ripped files as any other text." );
3067 puts( "NoteM: Too many 'fsetpos', 'fread', 'fwrite' invocations were put in the straight port (from 32bit internal memory to" );
3068 puts( "       64bit external memory), a optimization is needed, something like reading/writing a LEAF at once." );
3069 puts( "NoteN: Since revision 14+: Optimized(LEAFwise) search (fragment 1] and 2]), insert (fragment 3]) and dump." );
3070 puts( "NoteO: In next revisions a 2in1 is to be done i.e. one code fragment will deal with virtual and physical memory," );
3071 puts( "       thus establishing pure 64bit mode of operation, a single flag will decide whether 'memcpy' or" );
3072 puts( "       the slow I/O triad sub-fragments will be used. DONE." );
3073 puts( "NoteP: In next revisions a multi-pass (by chunking the hash table) mode is to be added in order to avoid" );
3074 puts( "       these sick-seeks. DONE." );
3075 puts( "NoteQ: Fixed occurrencies bug due to not NULLifying the field housing the occurrencies, a nasty thing: all" );
3076 puts( "       the revisions 14??? were buggy, how stupid from my side, grumble." );
3077 puts( "NoteR: In r.14++++FIXFIX were fixed STATS(Leprechaun.LOG) bugs (appearing only in multi-pass mode) due to not" );
3078 puts( "       NULLifying the variables housing the stats, they do not affect the results - they are for informative use." );
3079 puts( "NoteS: Fixed a division-by-zero bug, occurs when finishing-starting time is under 1 second." );
3080 puts( "       Fixed a nasty bug causing very restrictive way of forming x-grams." );
3081 puts( "NoteT: At last and finally the nasty bug causing very restrictive way of forming x-grams was REALLY fixed - lack of" );
3082 puts( "       calmness jammed (again) my actions - a lesson to be relearnt." );
3083
3084 puts( "NoteU: Since r.15FIXFIX+ the ability to command Leprechaun (from inside the list file with 2 metacommands) to enter/exit" );
3085 puts( "       INSERT mode was added. This allows to control whether new (to current hash-tree structure) x-grams are to be counted" );
3086 puts( "       [and] INSERTed. These two metacommands are:" );
3087 puts( "       Leprechaun says x-gram inserting disabled for next files: ON" );
3088 puts( "       Leprechaun says x-gram inserting disabled for next files: OFF" );
3089
3090 puts( "NoteV: When W/w option is used multiple-passes shouldn't be dumped - it is meaningless, dump when only one pass," );
3091 puts( "       that is, use W/w only in ONE-PASS mode otherwise it behaves as Z/z but DOES NOT dump to OutFile." );
3092 puts( "       It uses in READ mode the two HASH+TREES output files: 'Leprechaun_64bit.hsh' and 'Leprechaun_64bit.swp'." );
3093 puts( "       If during the start one of them is missing then Z/z behaviour is on, at end 'Leprechaun_64bit.hsh' is dumped." );
3094 puts( "       Also the OutFile has all incoming x-grams which are present in the corpus (i.e. HASH+TREES structure)." );
3095   puts( "" );
3096   puts( "Usage: Leprechaun InFile OutFile [BufferSize] [SortMethod] [TreeMethod]" );
3097   puts( "      <InFile>: Input file with files for Leprechauning, in WINDOWS console" );
3098   puts( "                you can create it by 'E:\\KAZEHOME>dir *.txt/s/b>Leprechaun.lst'" );
3099   puts( "      <OutFile>: Output WORDLIST(sorted since r.9, CRLF) file" );
3100   puts( "      <BufferSize>: Optional Dynamic RAM buffer in KB, default(and minimum" );
3101   puts( "                    in the same time) is 1023, i.e. omit or specify greater one" );
3102   puts( "      <SortMethod>: Optional Sort Method, default is 'D'," );
3103   puts( "                    A - InsertionSort" );
3104   puts( "                    B - InsertionX26Sort" );
3105   puts( "                    C - MultiKeyQuickSortSort by J. Bentley, R. Sedgewick" );
3106   puts( "                    D - MultiKeyQuickSortX26Sort' by J. Bentley, R. Sedgewick" );
3107   puts( "      <TreeMethod>: Optional Tree Method, default is 'X'," );
3108   puts( "                    X - Binary-Search-Trees" );
3109   puts( "                    y - B-Trees of order 3, INTERNAL/fast memory digitless i.e. no repetitions, 64bit addressing!" );
3110   puts( "                    Y - B-Trees of order 3, INTERNAL/fast memory, 64bit addressing!" );
3111   puts( "                    z - B-Trees of order 3, EXTERNAL/slow memory digitless i.e. no repetitions, 64bit addressing!" );
3112   puts( "                    Z - B-Trees of order 3, EXTERNAL/slow memory, 64bit addressing!" );
3113   puts( "                    w - B-Trees of order 3, EXTERNAL/slow memory digitless i.e. no repetitions, 64bit addressing! REUSE!" );
3114   puts( "                    W - B-Trees of order 3, EXTERNAL/slow memory, 64bit addressing! REUSE!" );
3115   puts( "" );
3116 puts( "Have a nice Leprechauning." );
3117 puts( "For contacts: sanmayce@sanmayce.com" );
3118 puts( "Sanmayce Svalgyatchx 'Kaze', 2005 Feb 07. Last revision: 2012 Dec 16." );
3119   return( 1 );
3120 }
3121
3122   GRMBLhill[0]=0;
3123   GRMBLhill[1]=1;
3124   GRMBLhill[2]=1;
3125   GRMBLhill[3]=1;
3126   GRMBLhill[4]=1;
3127   GRMBLhill[5]=1;
3128   GRMBLhill[6]=1;
3129   GRMBLhill[7]=1;
```



```
3130   GRMBLhill[8]=1;
3131   GRMBLhill[9]=1;
3132   GRMBLhill[10]=1;
3133   GRMBLhill[11]=1;
3134   GRMBLhill[12]=15;
3135   GRMBLhill[13]=15;
3136   GRMBLhill[14]=15;
3137   GRMBLhill[15]=20;
3138   GRMBLhill[16]=30;
3139   GRMBLhill[17]=40;
3140   GRMBLhill[18]=50;
3141   GRMBLhill[19]=50;
3142   GRMBLhill[20]=50;
3143   GRMBLhill[21]=40;
3144   GRMBLhill[22]=40;
3145   GRMBLhill[23]=40;
3146   GRMBLhill[24]=30;
3147   GRMBLhill[25]=20;
3148   GRMBLhill[26]=20;
3149   GRMBLhill[27]=20;
3150   GRMBLhill[28]=20;
3151   GRMBLhill[29]=20;
3152   GRMBLhill[30]=10;
3153   GRMBLhill[31]=10;
3154
3155 (void) time(&tMainB);
3156
3157 if( ( fp_out = fopen( argv[2], "wb+" ) ) == NULL )
3158 { printf( "Leprechaun: Can't create file %s \n", argv[2] ); return( 1 ); }
3159 fclose(fp_out); // The file must be with size 0 because it is opened for appending down below.
3160
3161 // 2^(HashInBITS-HashChunkSizeInBITS)=2^0=1 passe(s).
3162 // 14++++ [
3163 //for( RipPasses = 1-1; RipPasses <= (1<<(HashInBITS-HashChunkSizeInBITS))-1; RipPasses++ )
3164 //{
3165 // 14++++ ]
3166 RipPasses = 1-1;
3167 WhyTheHellForIsNotWorking:
3168 printf( "Pass #%lu of %lu:\n", RipPasses+1, (1<<(HashInBITS-HashChunkSizeInBITS)));
3169
3170 if( ( fp_in = fopen( argv[1], "rb" ) ) == NULL )
3171 { printf( "Leprechaun: Can't open file %s \n", argv[1] ); return( 1 ); }
3172
3173 fseek( fp_in, 0L, SEEK_END );
3174 size_in = ftell( fp_in );
3175 fseek( fp_in, 0L, SEEK_SET );
3176 printf( "Size of input file with files for Leprechauning: %lu\n", size_in );
3177
3178 if( ( fp_outLOG = fopen( "Leprechaun.LOG", "a+" ) ) == NULL )
3179 { printf( "Leprechaun: Can't open file Leprechaun.LOG.\n" ); return( 1 ); }
3180
3181 // argc is 4|5|6 due to eventual missing BufferSize
3182 if( argc == 4 ) // not 6 due to eventual missing BufferSize and SortMethod
3183     k_FIX = 3;
3184 if( argc == 5)  // not 6 due to eventual missing BufferSize or SortMethod
3185     k_FIX = 4;
3186 if( argc == 6 )
3187     k_FIX = 5;
3188 if (*argv[k_FIX] == 'Y' || *argv[k_FIX] == 'y') BSTorBtree = 1+2; // +2 since r.14++
3189 if (*argv[k_FIX] == 'Z' || *argv[k_FIX] == 'z') BSTorBtree = 2;
3190 if (*argv[k_FIX] == 'W' || *argv[k_FIX] == 'w') {BSTorBtree = 2; REUSE=1;}
3191
3192 if( argc == 4 || argc == 5 || argc == 6 ) Thunderwith = atoi( argv[3] );
3193 else Thunderwith = 527; // for r.12:   527=17*31 this is minimum because of 4096*1*4=16KB+ needed for each buffer!
3194                         // for r.12+: 1023=33*31 this is minimum because of 4096*2*4=32KB+ needed for each buffer!
3195 if (Thunderwith < 1023) {Thunderwith = 1023;}
3196
3197 //printf( "Use next time as third parameter: %s\n", _ui64toaKAZEcomma((25123456789>>10)+1, llTOaDigits, 10) );
3198 //printf( "Use next time as third parameter: %s\n", _ui64toaKAZEcomma((25123456789/1024)+1, llTOaDigits, 10) );
3199
3200 if (BSTorBtree < 2) { printf( "Leprechaun: In this particular revision 'x' option is disabled.\n" ); return( 1 ); }
3201
3202 if (BSTorBtree < 2) {
3203 LetterBuffer = Thunderwith * 1024;
3204   WHOLEletter_BufferSize = 0;
3205   for( i = 1; i <= 31; i++ )
3206   { OffsetsInBuffer[i-1] = 0;
3207     for( j = 1; j <= i; j++ )
3208     { OffsetsInBuffer[i-1] = OffsetsInBuffer[i-1] + (GRMBLhill[(int)(j-1)] * LetterBuffer)/31;
3209     }
3210     WHOLEletter_BufferSize = WHOLEletter_BufferSize + (GRMBLhill[(int)i] * LetterBuffer)/31;
3211     GRMBLFoolAgain[(int)i] = (GRMBLhill[(int)i] * LetterBuffer)/31;
3212   }
3213   memory_size = 26 * WHOLEletter_BufferSize + 1 + 64;
3214 printf( "Allocating memory %luMB ... ", (memory_size>>20)+1 );
3215 pointerflushUNALIGN = (char *)malloc( memory_size );
3216 if( pointerflushUNALIGN == NULL )
3217 { puts( "\nLeprechaun: Needed memory allocation denied!\n" ); return( 1 ); }
```

```
pointerflush = pointerflushUNALIGN + 64 - (((size_t)pointerflushUNALIGN) % 64); // 13_6+
//offset=64-int((long)data&63);

printf( "OK\n");
                    fprintf( fp_outLOG, "Leprechaun report:\n" );

// Check once for ever whether allocated memory is ZEROed!? Answer: YES
//for( i = 0; i < memory_size; i++ )
// if (*(char *)(pointerflush+i)!=0) printf("NON-ZERO encountered, so 'NO'.");

for( i = 0; i < 26; i++ )
{ for( k = 1; k <= 31; k++ )
  { bufend[i*31+k-1] = pointerflush + i * WHOLEletter_BufferSize + OffsetsInBuffer[k-1]; // i*31+k-1 must be 0..805
        if (i==25) { MAXusedBuffer[k] = (unsigned long)bufend[i*31+k-1]; }
    for( j = 0; j < (NumberOfSLOTs+1)*4; j++ ) // ? memset(bufend[i],0,(NumberOfSLOTs+1)*4);
    { *bufend[i*31+k-1]++ = 0;
      //++bufend[i*31+k-1];
    }
        if (i==25) { MAXusedBuffer[k] = (unsigned long)bufend[i*31+k-1]-MAXusedBuffer[k]; }
    bufNumberOfWords[i*31+k-1]=0;
//for( j = 0; j < NumberOfSLOTs; j++ )
//bufNowpS[i*31+k-1][j]=0;
  }
}

} else { //if (BSTorBtree != 2) {
// ` ASCII code 095
// ~ ASCII code 096  \
// a ASCII code 097  / In total 26+1+1 radix instead of 27 to avoid +1 for each '_', code 096 not used.
// z ASCII code 122
// The hash  for 'a_quadruplet_for_example' will be calculated for first 5 chars:
// = (byte1-'_')*28*28*28*28 + (byte2-'_')*28*28*28 + (byte3-'_')*28*28 + (byte4-'_')*28 + (byte5-'_')
// Hash slots are 28*28*28*28*28 = 17,210,368 each containing one 64bit pointer i.e. 8bytes in length.
// Hash size = 17,210,368*8 = 137,682,944 bytes
// When at end all these slots(17,210,368- Btrees) are traversed the outcome is a sorted wordlist - no need of sorting.
//unsigned long long SeekPosition;
//unsigned long long *PointerToSeekPosition;
// 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.
// For final stats all non-zero slots point to one btree.
//   printf( "Allocating HASH memory %s bytes ... ", _ui64toaKAZEcomma( (17210368*8) + 1 + 64 , llTOaDigits, 10) );
//   pointerflushUNALIGN = (char *)malloc( (17210368*8) + 1 + 64 );
// Hash slots are 27bit = 2^27 = 134,217,728 each containing one 64bit pointer i.e. 8bytes in length.
printf( "Allocating HASH memory %s bytes ... ", _ui64toaKAZEcomma( ((1<<HashInBITS)*8) + 1 + 64 , llTOaDigits, 10) );
pointerflushUNALIGN = (char *)malloc( (1<<HashInBITS)*8 + 1 + 64 );
if( pointerflushUNALIGN == NULL )
{ puts( "\nLeprechaun: Needed memory allocation denied!\n" ); return( 1 ); }
// r16
pointerflush = pointerflushUNALIGN;
//pointerflush = pointerflushUNALIGN + 64 - (((size_t)pointerflushUNALIGN) % 64); // 13_6+
//offset=64-int((long)data&63);
printf( "OK\n");
//  memset(pointerflush,0,17210368*8);
memset(pointerflush,0,(1<<HashInBITS)*8);
                        if (BSTorBtree == 2) {
                        
if( ( fp_outRG = fopen( "Leprechaun_64bit.hsh", "rb" ) ) == NULL )
{
    HSHexist=0;
    if ( REUSE && ((HashInBITS-HashChunkSizeInBITS)==0) ) // Multiple-passes shouldn't be uploaded - it is meaningless, dump when only one pass.
                printf( "Leprechaun: Can't find file 'Leprechaun_64bit.hsh'.\n" );
} else {
    HSHexist=1;
fclose(fp_outRG);
}

if( ( fp_outRG = fopen( "Leprechaun_64bit.swp", "rb" ) ) == NULL )
{
    SWPexist=0;
    if ( REUSE && ((HashInBITS-HashChunkSizeInBITS)==0) )
                printf( "Leprechaun: Can't find file 'Leprechaun_64bit.swp'.\n" );
} else {
    SWPexist=1;
fclose(fp_outRG);
}

if ( REUSE && ((HashInBITS-HashChunkSizeInBITS)==0) && (SWPexist+HSHexist == 2) ) {
    REUSE=2;
    if( ( fp_out = fopen( argv[2], "wb+" ) ) == NULL )
    { printf( "Leprechaun: Can't create file %s \n", argv[2] ); return( 1 ); }
}

if( ( fp_outRG = fopen( "Leprechaun_64bit.hsh", "rb" ) ) != NULL ) {
    if ( REUSE == 2 ) { //  REUSE [

#if defined(_WIN32_ENVIRONMENT_)
   // 64bit:
```



```
_lseeki64( fileno(fp_outRG), 0L, SEEK_END );
size_inLINESIXFOUR = _telli64( fileno(fp_outRG) );
_lseeki64( fileno(fp_outRG), 0L, SEEK_SET );
#else
   // 64bit:
fseeko( fp_outRG, 0L, SEEK_END );
size_inLINESIXFOUR = ftello( fp_outRG );
fseeko( fp_outRG, 0L, SEEK_SET );
#endif /* defined(_WIN32_ENVIRONMENT_)  */
printf( "Uploading-n-Reusing 'Leprechaun_64bit.hsh' file: %s bytes\n", _ui64toaKAZEcomma(size_inLINESIXFOUR, llTOaDigits, 10) );

              fread( pointerflushUNALIGN, 1, (1<<HashInBITS)*8 + 1 + 64, fp_outRG ); // Notice that the actual size of .HSH file is not calculated since it won't work if not the same as during the creation.
    }
fclose(fp_outRG);
}

// Tag for the swap file is: LEPRECHAUNISH{ASCIIcode26}
// or 14bytes, then when type of the swap is requested:
// D:\_KAZE_~1\LEPREC~1>type Leprechaun_64bit.swp
// LEPRECHAUNISH
// D:\_KAZE_~1\LEPREC~1>
size_in64_L14 = 1024 * (unsigned long long)Thunderwith + 14;
BufEnd_64 = 0+14;
// 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.

/*
The opentype argument is a string that controls how the file is opened and specifies attributes of the resulting stream. It must begin with one of the following sequences of characters:

'r'
    Open an existing file for reading only.
'w'
    Open the file for writing only. If the file already exists, it is truncated to zero length. Otherwise a new file is created.
'a'
    Open a file for append access; that is, writing at the end of file only. If the file already exists, its initial contents are unchanged and output to the stream is appended to the end of the file. Otherwise, a new, empty file is created.
'r+'
    Open an existing file for both reading and writing. The initial contents of the file are unchanged and the initial file position is at the beginning of the file.
'w+'
    Open a file for both reading and writing. If the file already exists, it is truncated to zero length. Otherwise, a new file is created.
'a+'
    Open or create file for both reading and appending. If the file exists, its initial contents are unchanged. Otherwise, a new file is created. The initial file position for reading is at the beginning of the file, but output is always appended to the end of the file.
*/

    // r16: Three+One conditions to reuse: Leprechaun_64bit.swp to exist, Not in multi-pass mode, W/w specified. The last one is the HASH upload to have been successful!
    if ( REUSE == 2 ) { //  REUSE [
if( ( fp_outRG = fopen( "Leprechaun_64bit.swp", "rb+" ) ) == NULL )
{ printf( "Leprechaun: Can't create file 'Leprechaun_64bit.swp'.\n" ); return( 1 ); }

#if defined(_WIN32_ENVIRONMENT_)
   // 64bit:
_lseeki64( fileno(fp_outRG), 0L, SEEK_END );
size_inLINESIXFOUR = _telli64( fileno(fp_outRG) );
_lseeki64( fileno(fp_outRG), 0L, SEEK_SET );
#else
   // 64bit:
fseeko( fp_outRG, 0L, SEEK_END );
size_inLINESIXFOUR = ftello( fp_outRG );
fseeko( fp_outRG, 0L, SEEK_SET );
#endif /* defined(_WIN32_ENVIRONMENT_)  */
printf( "Reusing 'Leprechaun_64bit.swp' file: %s bytes\n", _ui64toaKAZEcomma(size_inLINESIXFOUR, llTOaDigits, 10) );

fsetpos(fp_outRG, &BufEnd_64); // SOMETHING ROTTEN with lseeki64/fseeko and fsetpos ??!! So DO-IT-OVER.
    }
    else { // REUSE
if( ( fp_outRG = fopen( "Leprechaun_64bit.swp", "wb+" ) ) == NULL )
{ printf( "Leprechaun: Can't create file 'Leprechaun_64bit.swp'.\n" ); return( 1 ); }
printf( "Allocating/ZEROing %s bytes swap file ... ", _ui64toaKAZEcomma(size_in64_L14, llTOaDigits, 10) );
fsetpos(fp_outRG, &fsetpos_ZERO); // SOMETHING ROTTEN with lseeki64/fseeko and fsetpos ??!! So DO-IT-OVER.
memset(OneCkusterZEROES,0,1024*4);
for (ThunderwithL64_L14=0; ThunderwithL64_L14 < size_in64_L14/(1024*4); ThunderwithL64_L14++)
              fwrite(OneCkusterZEROES, 1024*4, 1, fp_outRG);
for (ThunderwithL64_L14=0; ThunderwithL64_L14 < size_in64_L14%(1024*4); ThunderwithL64_L14++)
              fwrite(&OneChar_ieByte, 1, 1, fp_outRG);
fsetpos(fp_outRG, &fsetpos_ZERO); // SOMETHING ROTTEN with lseeki64/fseeko and fsetpos ??!! So DO-IT-OVER.
              fwrite(FileSwapTag, 13, 1, fp_outRG);
              fwrite(&EOFcode, 1, 1, fp_outRG);
fsetpos(fp_outRG, &BufEnd_64); // SOMETHING ROTTEN with lseeki64/fseeko and fsetpos ??!! So DO-IT-OVER.
printf( "OK\n");
    } // REUSE ]
                        } else { // ########## 64bit memory manipulations [
size_in64_L14 = 1024 * (unsigned long long)Thunderwith + 14 + 1 + 64;
printf( "Allocating memory %luMB ... ", (size_in64_L14>>20)+1 );
```

```
3385 pointerflushUNALIGN_64 = (char *)malloc( size_in64_L14 );
3386 if( pointerflushUNALIGN_64 == NULL )
3387 { puts( "\nLeprechaun: Needed memory allocation denied!\n" ); return( 1 ); }
3388 pointerflush_64 = pointerflushUNALIGN_64 + 64 - (((size_t)pointerflushUNALIGN_64) % 64); // 13_6+
3389 //offset=64-int((long)data&63);
3390 //memset(pointerflush_64,0,1024 * (unsigned long long)Thunderwith + 14);
3391 BufEnd_64 = (unsigned long long)pointerflush_64;
3392 /*
3393 printf( "BufEnd_64: %s\n", _ui64toaKAZEcomma(BufEnd_64, llTOaDigits, 10) );
3394 printf( "pointerflush_64: %s\n", _ui64toaKAZEcomma(pointerflush_64, llTOaDigits, 10) );
3395 pointerflush_64 = (char *)BufEnd_64;
3396 printf( "pointerflush_64: %s\n", _ui64toaKAZEcomma(pointerflush_64, llTOaDigits, 10) );
3397 exit (1);
3398 //BufEnd_64: 541,261,888
3399 //pointerflush_64: 541,261,888
3400 //pointerflush_64: 541,261,888
3401 */
3402 printf( "OK\n");
3403                         } // ######### 64bit memory manipulations ]
3404 fprintf( fp_outLOG, "Leprechaun report:\n" );
3405 } //if (BSTorBtree != 2) {
3406
3407
3408 // PROGRAM PROGRAM PROGRAM PROGRAM PROGRAM PROGRAM PROGRAM PROGRAM PROGRAM
3409 (void) time(&t1);
3410         Melnitchka = 0;
3411         WORDcount = 0; // Total word count i.e. for all files!
3412         WORDcountDistinct = 0;
3413         NumberOfFiles = 0;
3414         NumberOfLines = 0;
3415         FilesLEN = 0;
3416         LINE10len = 0;
3417 // Added in r.14+++++FIXFIX [
3418 NumberOfTrees=0; NumberOfHashCollisions=0;
3419 NumberOfLEAFs=0;
3420 WORDcountAttemptsToPut=0;
3421 LevelsInCorona_Not_Counting_ROOT=0;
3422 // Added in r.14+++++FIXFIX ]
3423
3424         for( k = 0; k < size_in; k++ )
3425 {
3426                 fread( &workbyte, 1, 1, fp_in );
3427                 if( workbyte != 10 )
3428                 { if( workbyte != 13 ) // NON_UNIX
3429                   { if( LINE10len < 255 ) { LINE10[ LINE10len ] = workbyte; }
3430                     LINE10len++;
3431                   }
3432                   else
3433                   {
3434                   }
3435     }
3436     else
3437                 { if( 1 <= LINE10len && LINE10len <= 255 )
3438                   { LINE10[ LINE10len ] = 0;
3439 METACOMMANDFlag = 0;
3440 if ( strcmp(LINE10, "Leprechaun says x-gram inserting disabled for next files: ON\0") == 0 ) {DoNotInsertFlag = 1; METACOMMANDFlag = 1;}
3441 if ( strcmp(LINE10, "Leprechaun says x-gram inserting disabled for next files: OFF\0") == 0 ) {DoNotInsertFlag = 0; METACOMMANDFlag = 1;}
3442
3443 if( METACOMMANDFlag == 0 )
3444 { // ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ IT IS a FILENAME not a METACOMMAND [
3445 if( ( fp_inLINE = fopen( LINE10, "rb" ) ) == NULL )          // Since r15FIXFIX+ a command [METACOMMAND] inside the .LST file is allowed:
    'Leprechaun says x-gram inserting disabled for next files: ON'
3446                                                             // To allow again (which is default) use: 'Leprechaun says x-gram inserting disabled for
    next files: OFF'
3447 { printf( "Leprechaun: Can't open file %s \n", LINE10 ); return( 1 ); }
3448
3449 //fseek( fp_inLINE, 0L, SEEK_END );  //Rev. 12
3450 //size_inLINE = ftell( fp_inLINE );  //Rev. 12
3451 //fseek( fp_inLINE, 0L, SEEK_SET );  //Rev. 12
3452
3453 #if defined(_WIN32_ENVIRONMENT_)
3454     // 64bit:
3455 _lseeki64( fileno(fp_inLINE), 0L, SEEK_END );
3456 size_inLINESIXFOUR = _telli64( fileno(fp_inLINE) );
3457 _lseeki64( fileno(fp_inLINE), 0L, SEEK_SET );
3458 #else
3459     // 64bit:
3460 fseeko( fp_inLINE, 0L, SEEK_END );
3461 size_inLINESIXFOUR = ftello( fp_inLINE );
3462 fseeko( fp_inLINE, 0L, SEEK_SET );
3463 #endif /* defined(_WIN32_ENVIRONMENT_)  */
3464
3465 printf( "Size of Input TEXTual file: %s\n", _ui64toaKAZEcomma(size_inLINESIXFOUR, llTOaDigits, 10) );
3466 FilesLEN = FilesLEN + size_inLINESIXFOUR;
3467 NumberOfFiles++;
3468
3469         //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
3470 wrdlen = 0;
```



```
3471         for( i = 0; i < size_inLINESIXFOUR; i++ )
3472 {
3473
3474                 // ~~~~~~~~~~~~ Buffering fread [
3475                 if (workKoffset == -1) {
3476                         if (i + 1024*128 < size_inLINESIXFOUR) {
3477                                 fread( &workK[0], 1, 1024*128, fp_inLINE );
3478                                 workKoffset = 0;
3479                                 workbyte = workK[workKoffset];
3480                         } else
3481                         fread( &workbyte, 1, 1, fp_inLINE );
3482                 } else {
3483                         workKoffset++;
3484                         workbyte = workK[workKoffset];
3485                         if (workKoffset == 1024*128 - 1) workKoffset = -1;
3486                 }
3487                 // ~~~~~~~~~~~~ Buffering fread ]
3488
3489 //                if( isalpha( workbyte ) )
3490 //                {
3491 //                        if( wrdlen < 31 )
3492 //                        { wrd[ wrdlen ] = tolower( workbyte ); }
3493 //                        wrdlen++;
3494 //                }
3495
3496                 if ( workbyte < 'A' ) // Most characters are under alphabet - only one if
3497                 {
3498 ElStupido:
3499                         // This fragment is MIRRORed: #1 copy [
3500                         if (workbyte == 10) {NumberOfLines++;}
3501
3502 // Quadruple! [
3503 // Sliding window for 'wrd': The incoming string 'a lot of things must' becomes 'a_lot_of_things' and 'lot_of_things_must':
3504
3505 // ain_t_that_a
3506 // didn_t_feel_a
3507 // i_didn_t_feel
3508 // t_feel_a_thing
3509 // t_that_a_cake
3510
3511 // 316
3512 // 00:17:55,859 --> 00:17:58,447
3513 // Ain't that a cake ? I didn't feel a thing !
3514
3515                           if ( PLE_words_INITflag == 0 && ( (PLE_words != 0) || (PLE_words == 0 && wrdlen != 0) ) )
3516                           if ( workbyte == '.' || workbyte == '!' || workbyte == '?' || workbyte == ':' || workbyte == ';' || workbyte == ','
    || workbyte == '\t' ) {
3517                              PLE_words_INITflag = 1;
3518                           }
3519 // Quadruple! ]
3520
3521 //r.15fixfix [
3522                         if( wrdlen > 31 ) PLE_words_INITflag = 1;
3523 //r.15fixfix ]
3524
3525                         //if ( 1 <= wrdlen && wrdlen <= LongestLineInclusive ) // Enforce no word with length greater than 31 with below line
    to enter x-lets.
3526                         if ( 1 <= wrdlen && wrdlen <= 31 )
3527                 {
3528                                 wrd[ wrdlen ] = 0;
3529                                 // OTKACHAM: 1<<17-1 gives 65536 i.e. '-' have had high priority than '<<'
3530                                 //Next line gives error due to mix of '&' and 'double'
3531                                 if ((WORDcount & ((1<<18)-1)) == 0)
3532                                 { //_ui64toaKAZEzerocomma(WORDcount, llTOaDigits, 10);
3533                                   //printf( "Word count: %s(%lu/128 done)\r", llTOaDigits, ((long long)i*100) / size_inLINESIXFOUR );
3534 //++Melnitchka;
3535 //Melnitchka = Melnitchka % 4;
3536 //if (Melnitchka == 0){ printf( "|; Word count: %s of them %s distinct; Done: %lu/64\r", _ui64toaKAZEcomma(WORDcount, llTOaDigits, 10),
    _ui64toaKAZEcomma((unsigned long long)WORDcountDistinct, llTOaDigits2, 10), ((long long)i<<6) / size_inLINESIXFOUR ); }
3537 //if (Melnitchka == 1){ printf( "/; Word count: %s of them %s distinct; Done: %lu/64\r", _ui64toaKAZEcomma(WORDcount, llTOaDigits, 10),
    _ui64toaKAZEcomma((unsigned long long)WORDcountDistinct, llTOaDigits2, 10), ((long long)i<<6) / size_inLINESIXFOUR ); }
3538 //if (Melnitchka == 2){ printf( "-; Word count: %s of them %s distinct; Done: %lu/64\r", _ui64toaKAZEcomma(WORDcount, llTOaDigits, 10),
    _ui64toaKAZEcomma((unsigned long long)WORDcountDistinct, llTOaDigits2, 10), ((long long)i<<6) / size_inLINESIXFOUR ); }
3539 //if (Melnitchka == 3){ printf( "\\; Word count: %s of them %s distinct; Done: %lu/64\r", _ui64toaKAZEcomma(WORDcount, llTOaDigits, 10),
    _ui64toaKAZEcomma((unsigned long long)WORDcountDistinct, llTOaDigits2, 10), ((long long)i<<6) / size_inLINESIXFOUR ); }
3540 Melnitchka = Melnitchka & 3; // 0 1 2 3: 00 01 10 11
3541                         (void) time(&t4);
3542                         if (t4 <= t1) {t4 = t1; t4++;}
3543 printf( "%s; %sP/s; Phrase count: %s of them %s distinct; Done: %lu/64\r", Auberge[Melnitchka++], _ui64toaKAZEzerocomma(WORDcount/((int) t4-
    t1), llTOaDigits3, 10)+(26-10), _ui64toaKAZEcomma(WORDcount, llTOaDigits, 10), _ui64toaKAZEcomma((unsigned long long)WORDcountDistinct,
    llTOaDigits2, 10), ((long long)i<<6) / size_inLINESIXFOUR );
3544                                 }
3545
3546 //14+++ [
3547 PLE_words++;
3548
3549 #ifdef singleton
3550                                 PLE_words_INITflag = 1;
```

```
#endif
#ifdef doubleton
if (PLE_words == 1)
        strcpy( wrd1st, wrd );
else if (PLE_words == 2) {
        strcpy( wrd2nd, wrd );
        wrdlen = strlen(wrd1st)+strlen(wrd2nd)+1; // '_'
        //wrdlen = strlen(wrd);
        //if ( wrdlen <= 31 ) {
        if ( wrdlen <= LongestLineInclusive ) {
  strcpy(wrd, wrd1st);
  strcat(wrd, DelimiterUnderscore);
  strcat(wrd, wrd2nd);
        }
}
else {
        PLE_words = 2;
        strcpy( wrd1st, wrd2nd );
        strcpy( wrd2nd, wrd );
        wrdlen = strlen(wrd1st)+strlen(wrd2nd)+1; // '_'
        //wrdlen = strlen(wrd);
        //if ( wrdlen <= 31 ) {
        if ( wrdlen <= LongestLineInclusive ) {
  strcpy(wrd, wrd1st);
  strcat(wrd, DelimiterUnderscore);
  strcat(wrd, wrd2nd);
        }
}
#endif
#ifdef tripleton
if (PLE_words == 1)
        strcpy( wrd1st, wrd );
else if (PLE_words == 2)
        strcpy( wrd2nd, wrd );
else if (PLE_words == 3) {
        strcpy( wrd3rd, wrd );
        wrdlen = strlen(wrd1st)+strlen(wrd2nd)+strlen(wrd3rd)+1+1; // '_''_'
        //wrdlen = strlen(wrd);
        //if ( wrdlen <= 31 ) {
        if ( wrdlen <= LongestLineInclusive ) {
  strcpy(wrd, wrd1st);
  strcat(wrd, DelimiterUnderscore);
  strcat(wrd, wrd2nd);
  strcat(wrd, DelimiterUnderscore);
  strcat(wrd, wrd3rd);
        }
}
else {
        PLE_words = 3;
        strcpy( wrd1st, wrd2nd );
        strcpy( wrd2nd, wrd3rd );
        strcpy( wrd3rd, wrd );
        wrdlen = strlen(wrd1st)+strlen(wrd2nd)+strlen(wrd3rd)+1+1; // '_''_'
        //wrdlen = strlen(wrd);
        //if ( wrdlen <= 31 ) {
        if ( wrdlen <= LongestLineInclusive ) {
  strcpy(wrd, wrd1st);
  strcat(wrd, DelimiterUnderscore);
  strcat(wrd, wrd2nd);
  strcat(wrd, DelimiterUnderscore);
  strcat(wrd, wrd3rd);
        }
}
#endif
#ifdef quadrupleton
// Quadruple! [
if (PLE_words == 1)
        strcpy( wrd1st, wrd );
else if (PLE_words == 2)
        strcpy( wrd2nd, wrd );
else if (PLE_words == 3)
        strcpy( wrd3rd, wrd );
else if (PLE_words == 4) {
        strcpy( wrd4th, wrd );
        wrdlen = strlen(wrd1st)+strlen(wrd2nd)+strlen(wrd3rd)+strlen(wrd4th)+1+1+1; // '_''_''_'
        //wrdlen = strlen(wrd);
        //if ( wrdlen <= 31 ) {
        if ( wrdlen <= LongestLineInclusive ) {
  strcpy(wrd, wrd1st);
  strcat(wrd, DelimiterUnderscore);
  strcat(wrd, wrd2nd);
  strcat(wrd, DelimiterUnderscore);
  strcat(wrd, wrd3rd);
  strcat(wrd, DelimiterUnderscore);
  strcat(wrd, wrd4th);
        }
}
else {
```



```
        PLE_words = 4;
        strcpy( wrd1st, wrd2nd );
        strcpy( wrd2nd, wrd3rd );
        strcpy( wrd3rd, wrd4th );
        strcpy( wrd4th, wrd );
        wrdlen = strlen(wrd1st)+strlen(wrd2nd)+strlen(wrd3rd)+strlen(wrd4th)+1+1+1; // '_''_''_'
        //wrdlen = strlen(wrd);
        //if ( wrdlen <= 31 ) {
        if ( wrdlen <= LongestLineInclusive ) {
  strcpy(wrd, wrd1st);
  strcat(wrd, DelimiterUnderscore);
  strcat(wrd, wrd2nd);
  strcat(wrd, DelimiterUnderscore);
  strcat(wrd, wrd3rd);
  strcat(wrd, DelimiterUnderscore);
  strcat(wrd, wrd4th);
        }
}
// Quadruple! ]
#endif
#ifdef quintupleton
if (PLE_words == 1)
        strcpy( wrd1st, wrd );
else if (PLE_words == 2)
        strcpy( wrd2nd, wrd );
else if (PLE_words == 3)
        strcpy( wrd3rd, wrd );
else if (PLE_words == 4)
        strcpy( wrd4th, wrd );
else if (PLE_words == 5) {
        strcpy( wrd5th, wrd );
        wrdlen = strlen(wrd1st)+strlen(wrd2nd)+strlen(wrd3rd)+strlen(wrd4th)+strlen(wrd5th)+1+1+1+1; // '_''_''_''_'
        //wrdlen = strlen(wrd);
        //if ( wrdlen <= 31 ) {
        if ( wrdlen <= LongestLineInclusive ) {
  strcpy(wrd, wrd1st);
  strcat(wrd, DelimiterUnderscore);
  strcat(wrd, wrd2nd);
  strcat(wrd, DelimiterUnderscore);
  strcat(wrd, wrd3rd);
  strcat(wrd, DelimiterUnderscore);
  strcat(wrd, wrd4th);
  strcat(wrd, DelimiterUnderscore);
  strcat(wrd, wrd5th);
        }
}
else {
        PLE_words = 5;
        strcpy( wrd1st, wrd2nd );
        strcpy( wrd2nd, wrd3rd );
        strcpy( wrd3rd, wrd4th );
        strcpy( wrd4th, wrd5th );
        strcpy( wrd5th, wrd );
        wrdlen = strlen(wrd1st)+strlen(wrd2nd)+strlen(wrd3rd)+strlen(wrd4th)+strlen(wrd5th)+1+1+1+1; // '_''_''_''_'
        //wrdlen = strlen(wrd);
        //if ( wrdlen <= 31 ) {
        if ( wrdlen <= LongestLineInclusive ) {
  strcpy(wrd, wrd1st);
  strcat(wrd, DelimiterUnderscore);
  strcat(wrd, wrd2nd);
  strcat(wrd, DelimiterUnderscore);
  strcat(wrd, wrd3rd);
  strcat(wrd, DelimiterUnderscore);
  strcat(wrd, wrd4th);
  strcat(wrd, DelimiterUnderscore);
  strcat(wrd, wrd5th);
        }
}
#endif
#ifdef sextupleton
if (PLE_words == 1)
        strcpy( wrd1st, wrd );
else if (PLE_words == 2)
        strcpy( wrd2nd, wrd );
else if (PLE_words == 3)
        strcpy( wrd3rd, wrd );
else if (PLE_words == 4)
        strcpy( wrd4th, wrd );
else if (PLE_words == 5)
        strcpy( wrd5th, wrd );
else if (PLE_words == 6) {
        strcpy( wrd6th, wrd );
        wrdlen = strlen(wrd1st)+strlen(wrd2nd)+strlen(wrd3rd)+strlen(wrd4th)+strlen(wrd5th)+strlen(wrd6th)+1+1+1+1+1; // '_''_''_''_''_'
        //wrdlen = strlen(wrd);
        //if ( wrdlen <= 31 ) {
        if ( wrdlen <= LongestLineInclusive ) {
  strcpy(wrd, wrd1st);
  strcat(wrd, DelimiterUnderscore);
```

```
   strcat(wrd, wrd2nd);
   strcat(wrd, DelimiterUnderscore);
   strcat(wrd, wrd3rd);
   strcat(wrd, DelimiterUnderscore);
   strcat(wrd, wrd4th);
   strcat(wrd, DelimiterUnderscore);
   strcat(wrd, wrd5th);
   strcat(wrd, DelimiterUnderscore);
   strcat(wrd, wrd6th);
        }
}
else {
        PLE_words = 6;
        strcpy( wrd1st, wrd2nd );
        strcpy( wrd2nd, wrd3rd );
        strcpy( wrd3rd, wrd4th );
        strcpy( wrd4th, wrd5th );
        strcpy( wrd5th, wrd6th );
        strcpy( wrd6th, wrd );
        wrdlen = strlen(wrd1st)+strlen(wrd2nd)+strlen(wrd3rd)+strlen(wrd4th)+strlen(wrd5th)+strlen(wrd6th)+1+1+1+1+1+1; // '_''_''_''_''_'
        //wrdlen = strlen(wrd);
        //if ( wrdlen <= 31 ) {
        if ( wrdlen <= LongestLineInclusive ) {
   strcpy(wrd, wrd1st);
   strcat(wrd, DelimiterUnderscore);
   strcat(wrd, wrd2nd);
   strcat(wrd, DelimiterUnderscore);
   strcat(wrd, wrd3rd);
   strcat(wrd, DelimiterUnderscore);
   strcat(wrd, wrd4th);
   strcat(wrd, DelimiterUnderscore);
   strcat(wrd, wrd5th);
   strcat(wrd, DelimiterUnderscore);
   strcat(wrd, wrd6th);
        }
}
#endif
#ifdef septupleton
if (PLE_words == 1)
        strcpy( wrd1st, wrd );
else if (PLE_words == 2)
        strcpy( wrd2nd, wrd );
else if (PLE_words == 3)
        strcpy( wrd3rd, wrd );
else if (PLE_words == 4)
        strcpy( wrd4th, wrd );
else if (PLE_words == 5)
        strcpy( wrd5th, wrd );
else if (PLE_words == 6)
        strcpy( wrd6th, wrd );
else if (PLE_words == 7) {
        strcpy( wrd7th, wrd );
        wrdlen = strlen(wrd1st)+strlen(wrd2nd)+strlen(wrd3rd)+strlen(wrd4th)+strlen(wrd5th)+strlen(wrd6th)+strlen(wrd7th)+1+1+1+1+1+1; //
    '_''_''_''_''_''_'
        //wrdlen = strlen(wrd);
        //if ( wrdlen <= 31 ) {
        if ( wrdlen <= LongestLineInclusive ) {
   strcpy(wrd, wrd1st);
   strcat(wrd, DelimiterUnderscore);
   strcat(wrd, wrd2nd);
   strcat(wrd, DelimiterUnderscore);
   strcat(wrd, wrd3rd);
   strcat(wrd, DelimiterUnderscore);
   strcat(wrd, wrd4th);
   strcat(wrd, DelimiterUnderscore);
   strcat(wrd, wrd5th);
   strcat(wrd, DelimiterUnderscore);
   strcat(wrd, wrd6th);
   strcat(wrd, DelimiterUnderscore);
   strcat(wrd, wrd7th);
        }
}
else {
        PLE_words = 7;
        strcpy( wrd1st, wrd2nd );
        strcpy( wrd2nd, wrd3rd );
        strcpy( wrd3rd, wrd4th );
        strcpy( wrd4th, wrd5th );
        strcpy( wrd5th, wrd6th );
        strcpy( wrd6th, wrd7th );
        strcpy( wrd7th, wrd );
        wrdlen = strlen(wrd1st)+strlen(wrd2nd)+strlen(wrd3rd)+strlen(wrd4th)+strlen(wrd5th)+strlen(wrd6th)+strlen(wrd7th)+1+1+1+1+1+1; //
    '_''_''_''_''_''_'
        //wrdlen = strlen(wrd);
        //if ( wrdlen <= 31 ) {
        if ( wrdlen <= LongestLineInclusive ) {
   strcpy(wrd, wrd1st);
   strcat(wrd, DelimiterUnderscore);
```



```
   strcat(wrd, wrd2nd);
   strcat(wrd, DelimiterUnderscore);
   strcat(wrd, wrd3rd);
   strcat(wrd, DelimiterUnderscore);
   strcat(wrd, wrd4th);
   strcat(wrd, DelimiterUnderscore);
   strcat(wrd, wrd5th);
   strcat(wrd, DelimiterUnderscore);
   strcat(wrd, wrd6th);
   strcat(wrd, DelimiterUnderscore);
   strcat(wrd, wrd7th);
        }
}
#endif
#ifdef octupleton
if (PLE_words == 1)
        strcpy( wrd1st, wrd );
else if (PLE_words == 2)
        strcpy( wrd2nd, wrd );
else if (PLE_words == 3)
        strcpy( wrd3rd, wrd );
else if (PLE_words == 4)
        strcpy( wrd4th, wrd );
else if (PLE_words == 5)
        strcpy( wrd5th, wrd );
else if (PLE_words == 6)
        strcpy( wrd6th, wrd );
else if (PLE_words == 7)
        strcpy( wrd7th, wrd );
else if (PLE_words == 8) {
        strcpy( wrd8th, wrd );
        wrdlen =
    strlen(wrd1st)+strlen(wrd2nd)+strlen(wrd3rd)+strlen(wrd4th)+strlen(wrd5th)+strlen(wrd6th)+strlen(wrd7th)+strlen(wrd8th)+1+1+1+1+1+1+1; //
    '_''_''_''_''_''_''_'
        //wrdlen = strlen(wrd);
        //if ( wrdlen <= 31 ) {
        if ( wrdlen <= LongestLineInclusive ) {
   strcpy(wrd, wrd1st);
   strcat(wrd, DelimiterUnderscore);
   strcat(wrd, wrd2nd);
   strcat(wrd, DelimiterUnderscore);
   strcat(wrd, wrd3rd);
   strcat(wrd, DelimiterUnderscore);
   strcat(wrd, wrd4th);
   strcat(wrd, DelimiterUnderscore);
   strcat(wrd, wrd5th);
   strcat(wrd, DelimiterUnderscore);
   strcat(wrd, wrd6th);
   strcat(wrd, DelimiterUnderscore);
   strcat(wrd, wrd7th);
   strcat(wrd, DelimiterUnderscore);
   strcat(wrd, wrd8th);
        }
}
else {
        PLE_words = 8;
        strcpy( wrd1st, wrd2nd );
        strcpy( wrd2nd, wrd3rd );
        strcpy( wrd3rd, wrd4th );
        strcpy( wrd4th, wrd5th );
        strcpy( wrd5th, wrd6th );
        strcpy( wrd6th, wrd7th );
        strcpy( wrd7th, wrd8th );
        strcpy( wrd8th, wrd );
        wrdlen =
    strlen(wrd1st)+strlen(wrd2nd)+strlen(wrd3rd)+strlen(wrd4th)+strlen(wrd5th)+strlen(wrd6th)+strlen(wrd7th)+strlen(wrd8th)+1+1+1+1+1+1+1; //
    '_''_''_''_''_''_''_'
        //wrdlen = strlen(wrd);
        //if ( wrdlen <= 31 ) {
        if ( wrdlen <= LongestLineInclusive ) {
   strcpy(wrd, wrd1st);
   strcat(wrd, DelimiterUnderscore);
   strcat(wrd, wrd2nd);
   strcat(wrd, DelimiterUnderscore);
   strcat(wrd, wrd3rd);
   strcat(wrd, DelimiterUnderscore);
   strcat(wrd, wrd4th);
   strcat(wrd, DelimiterUnderscore);
   strcat(wrd, wrd5th);
   strcat(wrd, DelimiterUnderscore);
   strcat(wrd, wrd6th);
   strcat(wrd, DelimiterUnderscore);
   strcat(wrd, wrd7th);
   strcat(wrd, DelimiterUnderscore);
   strcat(wrd, wrd8th);
        }
}
#endif
```

```
#ifdef nonupleton
if (PLE_words == 1)
        strcpy( wrd1st, wrd );
else if (PLE_words == 2)
        strcpy( wrd2nd, wrd );
else if (PLE_words == 3)
        strcpy( wrd3rd, wrd );
else if (PLE_words == 4)
        strcpy( wrd4th, wrd );
else if (PLE_words == 5)
        strcpy( wrd5th, wrd );
else if (PLE_words == 6)
        strcpy( wrd6th, wrd );
else if (PLE_words == 7)
        strcpy( wrd7th, wrd );
else if (PLE_words == 8)
        strcpy( wrd8th, wrd );
else if (PLE_words == 9) {
        strcpy( wrd9th, wrd );
        wrdlen =
    strlen(wrd1st)+strlen(wrd2nd)+strlen(wrd3rd)+strlen(wrd4th)+strlen(wrd5th)+strlen(wrd6th)+strlen(wrd7th)+strlen(wrd8th)+strlen(wrd9th)+1+1+1+1
    +1+1+1+1; // '_''_''_''_''_''_''_''_'
        //wrdlen = strlen(wrd);
        //if ( wrdlen <= 31 ) {
        if ( wrdlen <= LongestLineInclusive ) {
  strcpy(wrd, wrd1st);
  strcat(wrd, DelimiterUnderscore);
  strcat(wrd, wrd2nd);
  strcat(wrd, DelimiterUnderscore);
  strcat(wrd, wrd3rd);
  strcat(wrd, DelimiterUnderscore);
  strcat(wrd, wrd4th);
  strcat(wrd, DelimiterUnderscore);
  strcat(wrd, wrd5th);
  strcat(wrd, DelimiterUnderscore);
  strcat(wrd, wrd6th);
  strcat(wrd, DelimiterUnderscore);
  strcat(wrd, wrd7th);
  strcat(wrd, DelimiterUnderscore);
  strcat(wrd, wrd8th);
  strcat(wrd, DelimiterUnderscore);
  strcat(wrd, wrd9th);
        }
}
else {
        PLE_words = 9;
        strcpy( wrd1st, wrd2nd );
        strcpy( wrd2nd, wrd3rd );
        strcpy( wrd3rd, wrd4th );
        strcpy( wrd4th, wrd5th );
        strcpy( wrd5th, wrd6th );
        strcpy( wrd6th, wrd7th );
        strcpy( wrd7th, wrd8th );
        strcpy( wrd8th, wrd9th );
        strcpy( wrd9th, wrd );
        wrdlen =
    strlen(wrd1st)+strlen(wrd2nd)+strlen(wrd3rd)+strlen(wrd4th)+strlen(wrd5th)+strlen(wrd6th)+strlen(wrd7th)+strlen(wrd8th)+strlen(wrd9th)+1+1+1+1
    +1+1+1+1; // '_''_''_''_''_''_''_''_'
        //wrdlen = strlen(wrd);
        //if ( wrdlen <= 31 ) {
        if ( wrdlen <= LongestLineInclusive ) {
  strcpy(wrd, wrd1st);
  strcat(wrd, DelimiterUnderscore);
  strcat(wrd, wrd2nd);
  strcat(wrd, DelimiterUnderscore);
  strcat(wrd, wrd3rd);
  strcat(wrd, DelimiterUnderscore);
  strcat(wrd, wrd4th);
  strcat(wrd, DelimiterUnderscore);
  strcat(wrd, wrd5th);
  strcat(wrd, DelimiterUnderscore);
  strcat(wrd, wrd6th);
  strcat(wrd, DelimiterUnderscore);
  strcat(wrd, wrd7th);
  strcat(wrd, DelimiterUnderscore);
  strcat(wrd, wrd8th);
  strcat(wrd, DelimiterUnderscore);
  strcat(wrd, wrd9th);
        }
}
#endif
#ifdef decupleton
if (PLE_words == 1)
        strcpy( wrd1st, wrd );
else if (PLE_words == 2)
        strcpy( wrd2nd, wrd );
else if (PLE_words == 3)
        strcpy( wrd3rd, wrd );
```



```
else if (PLE_words == 4)
        strcpy( wrd4th, wrd );
else if (PLE_words == 5)
        strcpy( wrd5th, wrd );
else if (PLE_words == 6)
        strcpy( wrd6th, wrd );
else if (PLE_words == 7)
        strcpy( wrd7th, wrd );
else if (PLE_words == 8)
        strcpy( wrd8th, wrd );
else if (PLE_words == 9)
        strcpy( wrd9th, wrd );
else if (PLE_words == 10) {
        strcpy( wrd10th, wrd );
        wrdlen =
    strlen(wrd1st)+strlen(wrd2nd)+strlen(wrd3rd)+strlen(wrd4th)+strlen(wrd5th)+strlen(wrd6th)+strlen(wrd7th)+strlen(wrd8th)+strlen(wrd9th)+strlen(
    wrd10th)+1+1+1+1+1+1+1+1+1+1; // '_''_''_''_''_''_''_''_''_''_'
        //wrdlen = strlen(wrd);
        //if ( wrdlen <= 31 ) {
        if ( wrdlen <= LongestLineInclusive ) {
  strcpy(wrd, wrd1st);
  strcat(wrd, DelimiterUnderscore);
  strcat(wrd, wrd2nd);
  strcat(wrd, DelimiterUnderscore);
  strcat(wrd, wrd3rd);
  strcat(wrd, DelimiterUnderscore);
  strcat(wrd, wrd4th);
  strcat(wrd, DelimiterUnderscore);
  strcat(wrd, wrd5th);
  strcat(wrd, DelimiterUnderscore);
  strcat(wrd, wrd6th);
  strcat(wrd, DelimiterUnderscore);
  strcat(wrd, wrd7th);
  strcat(wrd, DelimiterUnderscore);
  strcat(wrd, wrd8th);
  strcat(wrd, DelimiterUnderscore);
  strcat(wrd, wrd9th);
  strcat(wrd, DelimiterUnderscore);
  strcat(wrd, wrd10th);
        }
}
else {
        PLE_words = 10;
        strcpy( wrd1st, wrd2nd );
        strcpy( wrd2nd, wrd3rd );
        strcpy( wrd3rd, wrd4th );
        strcpy( wrd4th, wrd5th );
        strcpy( wrd5th, wrd6th );
        strcpy( wrd6th, wrd7th );
        strcpy( wrd7th, wrd8th );
        strcpy( wrd8th, wrd9th );
        strcpy( wrd9th, wrd10th );
        strcpy( wrd10th, wrd );
        wrdlen =
    strlen(wrd1st)+strlen(wrd2nd)+strlen(wrd3rd)+strlen(wrd4th)+strlen(wrd5th)+strlen(wrd6th)+strlen(wrd7th)+strlen(wrd8th)+strlen(wrd9th)+strlen(
    wrd10th)+1+1+1+1+1+1+1+1+1+1; // '_''_''_''_''_''_''_''_''_''_'
        //wrdlen = strlen(wrd);
        //if ( wrdlen <= 31 ) {
        if ( wrdlen <= LongestLineInclusive ) {
  strcpy(wrd, wrd1st);
  strcat(wrd, DelimiterUnderscore);
  strcat(wrd, wrd2nd);
  strcat(wrd, DelimiterUnderscore);
  strcat(wrd, wrd3rd);
  strcat(wrd, DelimiterUnderscore);
  strcat(wrd, wrd4th);
  strcat(wrd, DelimiterUnderscore);
  strcat(wrd, wrd5th);
  strcat(wrd, DelimiterUnderscore);
  strcat(wrd, wrd6th);
  strcat(wrd, DelimiterUnderscore);
  strcat(wrd, wrd7th);
  strcat(wrd, DelimiterUnderscore);
  strcat(wrd, wrd8th);
  strcat(wrd, DelimiterUnderscore);
  strcat(wrd, wrd9th);
  strcat(wrd, DelimiterUnderscore);
  strcat(wrd, wrd10th);
        }
}
#endif
//14+++ ]

//14+++ [
#ifdef singleton
                        if ( ( PLE_words == 1 ) && ( 1 <= wrdlen ) && ( wrdlen <= 31 ) ) {
#endif
#ifdef doubleton
```

```
                        if ( ( PLE_words == 2 ) && ( 5 <= wrdlen ) && ( wrdlen <= 41 ) ) {
#endif
#ifdef tripleton
                        if ( ( PLE_words == 3 ) && ( 9 <= wrdlen ) && ( wrdlen <= 41 ) ) {
#endif
#ifdef quadrupleton
                        if ( ( PLE_words == 4 ) && ( 13 <= wrdlen ) && ( wrdlen <= 51 ) ) {
#endif
#ifdef quintupleton
                        if ( ( PLE_words == 5 ) && ( 17 <= wrdlen ) && ( wrdlen <= 61 ) ) {
#endif
#ifdef sextupleton
                        if ( ( PLE_words == 6 ) && ( 21 <= wrdlen ) && ( wrdlen <= 71 ) ) {
#endif
#ifdef septupleton
                        if ( ( PLE_words == 7 ) && ( 25 <= wrdlen ) && ( wrdlen <= 81 ) ) {
#endif
#ifdef octupleton
                        if ( ( PLE_words == 8 ) && ( 29 <= wrdlen ) && ( wrdlen <= 91 ) ) {
#endif
#ifdef nonupleton
                        if ( ( PLE_words == 9 ) && ( 33 <= wrdlen ) && ( wrdlen <= 101 ) ) {
#endif
#ifdef decupleton
                        if ( ( PLE_words == 10 ) && ( 37 <= wrdlen ) && ( wrdlen <= 111 ) ) {
#endif
//14+++ ]
WORDcount++;
if (BSTorBtree < 2) {

  LetterOffset = (int)( wrd[0] - 'a' ) * 31 + (wrdlen-1); // 0..805
  //BufStart = pointerflush + LetterOffset * LetterBuffer; // OLD

  BufStart = pointerflush + (int)( wrd[0] - 'a' ) * WHOLEletter_BufferSize + OffsetsInBuffer[wrdlen-1];
  // Above line and Below line are equal
  //BufStart = pointerflush + (LetterOffset / 31) * WHOLEletter_BufferSize + OffsetsInBuffer[LetterOffset % 31];

              //Slot = KuxHash3plus(wrd)<<2; //13++
              //Slot = FNV1A_Hash_SHIFTless_XORless(wrd)<<2; //13+++
              //Slot = FNV1A_Hash_4_OCTETS(wrd, wrdlen>>2)<<2; //13++++
              //Slot = FNV1A_Hash_4_OCTETS_31(wrd, wrdlen>>2)<<2; //13+++++
/*
if (wrdlen<=19) // 4x4+3=19
              Slot = FNV1A_Hash_4_OCTETS(wrd, wrdlen>>2)<<2; //13++++
else            // 2x8+4=20 i.e. first contains 5 clashes
              Slot = FNV1A_Hash_8_OCTETS(wrd, wrdlen>>3)<<2; //13+++++
*/
//if (wrdlen<=19) // 4x4+3=19 i.e. last contains 7 clashes
//              Slot = FNV1A_Hash_Granularity(wrd, wrdlen>>2, 2)<<2; //13+++++
//else            // 2x8+4=20 i.e. first contains 6 clashes
//              Slot = FNV1A_Hash_Granularity(wrd, wrdlen>>3, 3)<<2; //13+++++

              //Slot = FNV1A_Hash_8_OCTETS(wrd, wrdlen>>3)<<2; //13_7p
              //Slot = HashFNV1A_unrolled_Final(wrd, wrdlen)<<2; //13_7p
       //Slot = HashAlfalfa_HALF(wrd, wrdlen)<<2; //13_7p
       //Slot = Hash_Alfalfa(wrd, wrdlen)<<2; //13_7p
//        Slot = Sixtinsensitive(wrd, wrdlen)<<2; //13_7p
       Slot = FNV1A_Hash_Jesteress(wrd, wrdlen)<<2; //13_7p
//        Slot = FNV1A_Hash_Jester(wrd, wrdlen)<<2; //13_7p

/*
hashAlfalfa = 7;
for(iAlfalfa = 0; iAlfalfa < (wrdlen & -2); iAlfalfa += 2) {
    hashAlfalfa = (17+9) * ((17+9) * hashAlfalfa + (wrd[iAlfalfa])) + (wrd[iAlfalfa+1]);
}
if(wrdlen & 1)
    hashAlfalfa = (17+9) * hashAlfalfa + (wrd[wrdlen-1]);

              Slot = (( hashAlfalfa ^ (hashAlfalfa >> 16) ) & 8191)<<2; //13_7p
*/

                    memcpy( &PseudoLinkedPointer, BufStart+Slot, 4 );
//; Line 917
//  mov     edx, DWORD PTR [eax+ebp]
//  add     esp, 4
                    // ?! DANGEROUS: above and below lines are(must:long must be 4bytes) identical
                    //PseudoLinkedPointer = (unsigned long)*(long *)(BufStart+Slot);
//; Line 919
//  mov     edx, DWORD PTR [eax+ebp]
//  add     esp, 4
                    //        while (count--) {
                    //        *(char *)dst = *(char *)src;
                    //        dst = (char *)dst + 1;
                    //        src = (char *)src + 1;
                    //        }

if (BSTorBtree == 0)
{
```



```
// @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ BST fragment [
                  if (PseudoLinkedPointer == 0) // means EMPTY-SLOT
                  {
          //if( (unsigned long)(bufend[LetterOffset] - BufStart) + wrdlen + 4 < (GRMBLhill[(int)wrdlen] * LetterBuffer)/31 ) // OLD slower
          if( (unsigned long)(bufend[LetterOffset] - BufStart) + wrdlen + 4 + 4 < GRMBLFoolAgain[(int)wrdlen] ) // +4 more for BST instead of LL
                    {
                    memcpy( BufStart+Slot, &bufend[LetterOffset], 4 );
//; Line 932
//  mov     DWORD PTR [eax+ebp], esi
                    // ?! DANGEROUS: above and below lines are(must:long must be 4bytes) identical
                    //*(long *)(BufStart+Slot) = *(long *)&bufend[LetterOffset];
//; Line 936
//  mov     DWORD PTR [eax+ebp], esi

                    // Below 3 lines are commented due to experiment below malloc which shows that allocated memory is ZEROed.
                    //memcpy( bufend[LetterOffset], &BufStart[NumberOfSLOTs*4], 4 ); // means next exists not: Means PseudoLinkedPointerL = 0
//; Line 940
//  mov     ecx, DWORD PTR [ebp+32768]
                    // ?! DANGEROUS: above and below lines are(must:long must be 4bytes) identical
                    //*(long *)bufend[LetterOffset] = *(long *)&BufStart[NumberOfSLOTs*4];
//; Line 944
//  mov     ecx, DWORD PTR [ebp+32768]
                    //bufend[LetterOffset] = bufend[LetterOffset] + 4;
                    //memcpy( bufend[LetterOffset], &BufStart[NumberOfSLOTs*4], 4 ); // means next exists not: Means PseudoLinkedPointerR = 0
                    bufend[LetterOffset] = bufend[LetterOffset] + 4 + 4; // + 4 due to above commenting
                    memcpy( bufend[LetterOffset], wrd, wrdlen ); WORDcountDistinct++; bufNumberOfWords[LetterOffset]++;
                              //bufNoWpS[LetterOffset][Slot]++; // ?! crashes
                    bufend[LetterOffset] = bufend[LetterOffset] + wrdlen;
                    if (MAXusedBuffer[wrdlen] < (unsigned long)(bufend[LetterOffset] - BufStart)) {MAXusedBuffer[wrdlen] = (unsigned long)(bufend[LetterOffset] - BufStart);}
                    }
          else
          { printf( "\nLeprechaun: Failure! Increment 'Memory for each Letter' parameter(third one)!\n" );
fprintf( fp_outLOG, "Input File with a list of TEXTual Files: %s\n", argv[1] );
fprintf( fp_outLOG, "Size of all TEXTual Files: %s\n", _ui64toaKAZEcomma(FilesLEN, llTOaDigits, 10) );
fprintf( fp_outLOG, "Word count: %s of them %s distinct\n", _ui64toaKAZEcomma(WORDcount, llTOaDigits, 10), _ui64toaKAZEcomma((unsigned long long)WORDcountDistinct, llTOaDigits2, 10) );
fprintf( fp_outLOG, "Number Of Files: %lu\n", NumberOfFiles );
fprintf( fp_outLOG, "Number Of Lines: %lu\n", NumberOfLines );
fprintf( fp_outLOG, "Allocated memory in MB: %lu\n", (unsigned long)(memory_size>>20)+1 );
fprintf( fp_outLOG, "Total Attempts to Find/Put WORDs into Binary-Search-Trees: %s\n", _ui64toaKAZEcomma(WORDcountAttemptsToPut, llTOaDigits, 10) );
for( k = 1; k < 32; k++ )
{ 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
}
fprintf( fp_outLOG, "Used value for third parameter in KB: %lu\n", (unsigned long)Thunderwith );
             fprintf( fp_outLOG, "Leprechaun: Failure! Increment 'Memory for each Letter' parameter(third one)!\n\n");
             return( 1 );
          }
                  }
                  else // means USED-SLOT
                  { FoundInLinkedList = 0;
                    while (PseudoLinkedPointer != 0 && FoundInLinkedList == 0)
                    {
                      if (memcmp(PseudoLinkedPointer+4+4,wrd,wrdlen) == 0)
                  //        while ( --count && *(char *)buf1 == *(char *)buf2 ) {
                  //        buf1 = (char *)buf1 + 1;
                  //        buf2 = (char *)buf2 + 1;
                  //        }
                  //        return( *((unsigned char *)buf1) - *((unsigned char *)buf2) );
                      { FoundInLinkedList = 1;
                      }
                      else // i.e < or >
                      {
                        if (memcmp(PseudoLinkedPointer+4+4,wrd,wrdlen) > 0)
                        memcpy( &PseudoLinkedPointerNEW, PseudoLinkedPointer, 4 );
                        else
                        {
                        PseudoLinkedPointer = PseudoLinkedPointer + 4;
                        memcpy( &PseudoLinkedPointerNEW, PseudoLinkedPointer, 4 );
                        }

                        if (PseudoLinkedPointerNEW == 0)
                  {
          //if( (unsigned long)(bufend[LetterOffset] - BufStart) + wrdlen + 4 < (GRMBLhill[(int)wrdlen] * LetterBuffer)/31 ) // OLD slower
          if( (unsigned long)(bufend[LetterOffset] - BufStart) + wrdlen + 4 + 4 < GRMBLFoolAgain[(int)wrdlen] )
                    { memcpy( PseudoLinkedPointer, &bufend[LetterOffset], 4 );
                      // Below 3 lines are commented due to experiment below malloc which shows that allocated memory is ZEROed.
                      //memcpy( bufend[LetterOffset], &BufStart[NumberOfSLOTs*4], 4 ); // means next exists not
                      //bufend[LetterOffset] = bufend[LetterOffset] + 4;
                      //memcpy( bufend[LetterOffset], &BufStart[NumberOfSLOTs*4], 4 ); // means next exists not
                      bufend[LetterOffset] = bufend[LetterOffset] + 4 + 4; // + 4 due to above commenting
```

```
                              memcpy( bufend[LetterOffset], wrd, wrdlen ); WORDcountDistinct++; bufNumberOfWords[LetterOffset]++;
                                 //bufNoWpS[LetterOffset][Slot]++; // ?! crashes
                              bufend[LetterOffset] = bufend[LetterOffset] + wrdlen;
                              if (MAXusedBuffer[wrdlen] < (unsigned long)(bufend[LetterOffset] - BufStart)) {MAXusedBuffer[wrdlen] = (unsigned long)(bufend[LetterOffset] - BufStart);}
                           }
            else
            { printf( "\nLeprechaun: Failure! Increment 'Memory for each Letter' parameter(third one)!\n" );
fprintf( fp_outLOG, "Input File with a list of TEXTual Files: %s\n", argv[1] );
fprintf( fp_outLOG, "Size of all TEXTual Files: %s\n", _ui64toaKAZEcomma(FilesLEN, llTOaDigits, 10) );
fprintf( fp_outLOG, "Word count: %s of them %s distinct\n", _ui64toaKAZEcomma(WORDcount, llTOaDigits, 10), _ui64toaKAZEcomma((unsigned long long)WORDcountDistinct, llTOaDigits2, 10) );
fprintf( fp_outLOG, "Number Of Files: %lu\n", NumberOfFiles );
fprintf( fp_outLOG, "Number Of Lines: %lu\n", NumberOfLines );
fprintf( fp_outLOG, "Allocated memory in MB: %lu\n", (unsigned long)(memory_size>>20)+1 );
fprintf( fp_outLOG, "Total Attempts to Find/Put WORDs into Binary-Search-Trees: %s\n", _ui64toaKAZEcomma(WORDcountAttemptsToPut, llTOaDigits, 10) );
for( k = 1; k < 32; k++ )
{ 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
}
fprintf( fp_outLOG, "Used value for third parameter in KB: %lu\n", (unsigned long)Thunderwith );
              fprintf( fp_outLOG, "Leprechaun: Failure! Increment 'Memory for each Letter' parameter(third one)!\n\n");
              return( 1 );
            }
                    }
                       PseudoLinkedPointer = PseudoLinkedPointerNEW;
                     }
                     WORDcountAttemptsToPut++;
                   } // while
                  }
// @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ BST fragment ]
} else
{
// ######################################## B-tree order 3 fragment [
//
// LEAF structure: [LeftPointer][MiddlePointer][RightPointer][LeftWord][RightWord]
//                 4bytes       4bytes         4bytes        wrdlen     wrdlen
//                                                           *          *          <- if *(char *)==0 means the word cell is empty
// ALL B-tree order 3 fragment consists of 3 sub-fragments:
// 1] Search 2] if Search failed Trasirascht(pushing in stack PseudoLinkedPointer(visited LEAFs)) Search 3] Insert Iterative

// 1] Search [ ________1407 line in C - see below: whole Search in assembler________
                  if (PseudoLinkedPointer == 0) // means EMPTY-SLOT
                  {
            if( (unsigned long)(bufend[LetterOffset] - BufStart) + 2*wrdlen + 4 + 4 + 4 + 4 < GRMBLFoolAgain[(int)wrdlen] ) // +4 more for BST instead of LL; + more(see LEAF)
                    {
                    memcpy( BufStart+Slot, &bufend[LetterOffset], 4 );
                    bufend[LetterOffset] = bufend[LetterOffset] + 4 + 4 + 4; // + 4 due to above commenting
                    memcpy( bufend[LetterOffset], wrd, wrdlen ); WORDcountDistinct++; bufNumberOfWords[LetterOffset]++;
                    bufend[LetterOffset] = bufend[LetterOffset] + 2*wrdlen;
                    if (MAXusedBuffer[wrdlen] < (unsigned long)(bufend[LetterOffset] - BufStart)) {MAXusedBuffer[wrdlen] = (unsigned long)(bufend[LetterOffset] - BufStart);}
                    }
            else
            { printf( "\nLeprechaun: Failure! Increment 'Memory for each Letter' parameter(third one)!\n" );
fprintf( fp_outLOG, "Input File with a list of TEXTual Files: %s\n", argv[1] );
fprintf( fp_outLOG, "Size of all TEXTual Files: %s\n", _ui64toaKAZEcomma(FilesLEN, llTOaDigits, 10) );
fprintf( fp_outLOG, "Word count: %s of them %s distinct\n", _ui64toaKAZEcomma(WORDcount, llTOaDigits, 10), _ui64toaKAZEcomma((unsigned long long)WORDcountDistinct, llTOaDigits2, 10) );
fprintf( fp_outLOG, "Number Of Files: %lu\n", NumberOfFiles );
fprintf( fp_outLOG, "Number Of Lines: %lu\n", NumberOfLines );
fprintf( fp_outLOG, "Allocated memory in MB: %lu\n", (unsigned long)(memory_size>>20)+1 );
fprintf( fp_outLOG, "Total Attempts to Find/Put WORDs into B-trees order 3: %s\n", _ui64toaKAZEcomma(WORDcountAttemptsToPut, llTOaDigits, 10) );
for( k = 1; k < 32; k++ )
{ 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
}
fprintf( fp_outLOG, "Used value for third parameter in KB: %lu\n", (unsigned long)Thunderwith );
              fprintf( fp_outLOG, "Leprechaun: Failure! Increment 'Memory for each Letter' parameter(third one)!\n\n");
              return( 1 );
            }
                    FoundInLinkedList = 1;
                  }
                  else // means USED-SLOT
                  { FoundInLinkedList = 0;
                    while (PseudoLinkedPointer != 0 && FoundInLinkedList == 0)
                    {
// ***** 'P W P' section [
// LW: existence check if ( *(char *)(PseudoLinkedPointer+4+4+4) != 0 )
// RW: existence check if ( *(char *)(PseudoLinkedPointer+4+4+4+wrdlen) != 0 )
                       // here ALWAYS LW exists: no need for existence check - line below
```



```
                       // if ( *(char *)(PseudoLinkedPointer+4+4+4) != 0 )
                       if (memcmp(PseudoLinkedPointer+4+4+4,wrd,wrdlen) > 0) // go LP
                          { memcpy( &PseudoLinkedPointerNEW, PseudoLinkedPointer + 0, 4 ); //LP
                            PseudoLinkedPointer = PseudoLinkedPointerNEW;
                          }
                       else if (memcmp(PseudoLinkedPointer+4+4+4,wrd,wrdlen) < 0) // go RP or MP
                          { // RW existence check - line below:
                            if ( *(char *)(PseudoLinkedPointer+4+4+4+wrdlen) != 0 ) // RW exists
                               { // Here all 'P W P' section is repeated; the way of handling case when dynamic number of words in leaf
// ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
// ***** 'P W P' section 2 [
// LW: existence check if ( *(char *)(PseudoLinkedPointer+4+4+4) != 0 )
// RW: existence check if ( *(char *)(PseudoLinkedPointer+4+4+4+wrdlen) != 0 )
                       // here ALWAYS RW exists: no need for existence check - line below
                       // if ( *(char *)(PseudoLinkedPointer+4+4+4+wrdlen) != 0 )
                       if (memcmp(PseudoLinkedPointer+4+4+4+wrdlen,wrd,wrdlen) > 0) // go MP
                          { memcpy( &PseudoLinkedPointerNEW, PseudoLinkedPointer + 4, 4 ); //MP
                            PseudoLinkedPointer = PseudoLinkedPointerNEW;
                          }
                       else if (memcmp(PseudoLinkedPointer+4+4+4+wrdlen,wrd,wrdlen) < 0) // go RP
                          { // No ?W after RW - go RP
                            memcpy( &PseudoLinkedPointerNEW, PseudoLinkedPointer + 4 + 4, 4 ); //RP
                            PseudoLinkedPointer = PseudoLinkedPointerNEW;
                          }
                       else FoundInLinkedList = 1; // wrd is RW
                       WORDcountAttemptsToPut++;
// ***** 'P W P' section 2 ]
// ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                               }
                            else // RW empty - go MP
                               { memcpy( &PseudoLinkedPointerNEW, PseudoLinkedPointer + 4, 4 ); //MP
                                 PseudoLinkedPointer = PseudoLinkedPointerNEW;
                               }
                          }
                       else FoundInLinkedList = 1; // wrd is LW
                       WORDcountAttemptsToPut++;
// ***** 'P W P' section ]
                     } // while
                       WORDcountAttemptsToPut--; // - 1 due to BST way of counting i.e. direct hash hit is not counted only successors
                  }
// 1] Search ] ________1484 line in C - see below: whole Search in assembler________

/*
; Line 1397
jmp $L2139
$L2042:
; Line 1408
test edx, edx
jne SHORT $L2110
; Line 1410
mov ecx, DWORD PTR _bufend$[esp+esi*4+892340]
mov edi, DWORD PTR _GRMBLFoolAgain$[esp+ebx*4+892340]
lea edx, DWORD PTR _bufend$[esp+esi*4+892340]
lea esi, DWORD PTR [ebx+ebx+12]
sub esi, ebp
add esi, ecx
cmp esi, edi
mov DWORD PTR tv4122[esp+892340], edx
jae $L2113
; Line 1412
mov DWORD PTR [eax+ebp], ecx
; Line 1413
lea eax, DWORD PTR [ecx+12]
; Line 1436
jmp $L2749
$L2110:
; Line 1437
mov DWORD PTR _FoundInLinkedList$[esp+892340], 0
npad 11
$L2141:
; Line 1438
mov eax, DWORD PTR _FoundInLinkedList$[esp+892340]
test eax, eax
jne $L2142
; Line 1445
lea ebp, DWORD PTR [edx+12]
mov ecx, ebx
lea edi, DWORD PTR _wrd$[esp+892340]
mov esi, ebp
xor eax, eax
repe cmpsb
je  SHORT $L2682
sbb eax, eax
sbb eax, -1
$L2682:
test eax, eax
jle SHORT $L2143
; Line 1447
```

```
4395 mov edx, DWORD PTR [edx]
4396 ; Line 1449
4397 jmp $L2153
4398 $L2143:
4399 mov ecx, ebx
4400 lea edi, DWORD PTR _wrd$[esp+892340]
4401 mov esi, ebp
4402 xor eax, eax
4403 repe cmpsb
4404 je  SHORT $L2640
4405 sbb eax, eax
4406 sbb eax, -1
4407 $L2640:
4408 test eax, eax
4409 jge SHORT $L2145
4410 ; Line 1451
4411 mov cl, BYTE PTR [edx+ebx+12]
4412 test cl, cl
4413 lea eax, DWORD PTR [edx+ebx+12]
4414 je  SHORT $L2147
4415 ; Line 1459
4416 mov ecx, ebx
4417 lea edi, DWORD PTR _wrd$[esp+892340]
4418 mov esi, eax
4419 xor ebp, ebp
4420 repe cmpsb
4421 je  SHORT $L2695
4422 sbb ebp, ebp
4423 sbb ebp, -1
4424 $L2695:
4425 test ebp, ebp
4426 jle SHORT $L2148
4427 ; Line 1461
4428 mov edx, DWORD PTR [edx+4]
4429 ; Line 1463
4430 jmp SHORT $L2151
4431 $L2148:
4432 mov esi, eax
4433 mov ecx, ebx
4434 lea edi, DWORD PTR _wrd$[esp+892340]
4435 xor eax, eax
4436 repe cmpsb
4437 je  SHORT $L2642
4438 sbb eax, eax
4439 sbb eax, -1
4440 $L2642:
4441 test eax, eax
4442 jge SHORT $L2150
4443 ; Line 1466
4444 mov edx, DWORD PTR [edx+8]
4445 ; Line 1468
4446 jmp SHORT $L2151
4447 $L2150:
4448 mov DWORD PTR _FoundInLinkedList$[esp+892340], 1
4449 $L2151:
4450 ; Line 1469
4451 mov ecx, DWORD PTR _WORDcountAttemptsToPut$[esp+892340]
4452 mov eax, DWORD PTR _WORDcountAttemptsToPut$[esp+892344]
4453 add ecx, 1
4454 adc eax, 0
4455 mov DWORD PTR _WORDcountAttemptsToPut$[esp+892340], ecx
4456 mov DWORD PTR _WORDcountAttemptsToPut$[esp+892344], eax
4457 ; Line 1473
4458 jmp SHORT $L2153
4459 $L2147:
4460 ; Line 1475
4461 mov edx, DWORD PTR [edx+4]
4462 ; Line 1478
4463 jmp SHORT $L2153
4464 $L2145:
4465 mov DWORD PTR _FoundInLinkedList$[esp+892340], 1
4466 $L2153:
4467 ; Line 1479
4468 mov esi, DWORD PTR _WORDcountAttemptsToPut$[esp+892340]
4469 mov ecx, DWORD PTR _WORDcountAttemptsToPut$[esp+892344]
4470 add esi, 1
4471 adc ecx, 0
4472 test edx, edx
4473 mov DWORD PTR _WORDcountAttemptsToPut$[esp+892340], esi
4474 mov DWORD PTR _WORDcountAttemptsToPut$[esp+892344], ecx
4475 jne $L2141
4476 $L2142:
4477 ; Line 1482
4478 mov edx, DWORD PTR _WORDcountAttemptsToPut$[esp+892340]
4479 mov ecx, DWORD PTR _WORDcountAttemptsToPut$[esp+892344]
4480 or  eax, -1
4481 add edx, eax
4482 adc ecx, eax
```



```
4483 mov DWORD PTR _WORDcountAttemptsToPut$[esp+892344], ecx
4484 mov DWORD PTR _WORDcountAttemptsToPut$[esp+892340], edx
4485 $L2139:
4486 */
4487
4488 if (FoundInLinkedList == 0)
4489 {
4490 // 2] if Search failed Trasirascht(pushing in stack PseudoLinkedPointer(visited LEAFs)) Search [
4491 //    'TracingSearch' is the same as 'Search' except that adds the trail in my simulated stack,
4492 //    the goal is not to waste time in 'Search' by dealing with no needed trail in case of not 'Insert'.
4493 //    Simulated stack contains pairs of 'Address of ParentLEAF' + 'Offset of ParentPointer in ParentLEAF i.e. 0 for LP, 4 for MP, 8 for RP'.
4494 //    'Offset ...' saves unnecessary comparisons of NEWword which after splitting goes up.
4495                   memcpy( &PseudoLinkedPointer, BufStart+Slot, 4 );
4496                StackPtr = 0;
4497                      while (PseudoLinkedPointer != 0)
4498                         {
4499 // ***** 'P W P' section [
4500 // LW: existence check if ( *(char *)(PseudoLinkedPointer+4+4+4) != 0 )
4501 // RW: existence check if ( *(char *)(PseudoLinkedPointer+4+4+4+wrdlen) != 0 )
4502                         // here ALWAYS LW exists: no need for existence check - line below
4503                         // if ( *(char *)(PseudoLinkedPointer+4+4+4) != 0 )
4504                         if (memcmp(PseudoLinkedPointer+4+4+4,wrd,wrdlen) > 0) // go LP
4505                            { memcpy( &PseudoLinkedPointerNEW, PseudoLinkedPointer + 0, 4 ); //LP
4506                if (StackPtr > 8192*3-1-1) { printf( "\nLeprechaun: Failure! 'B-tree order 3' simulated stack overflow!\n" ); return( 13 );}
4507                BSTstack[StackPtr] = PseudoLinkedPointer; ++StackPtr; //pt to visited leaf
4508                BSTstack[StackPtr] = 0; ++StackPtr; //LPoffset=0;MPoffset=4;RPoffset=8;
4509                               PseudoLinkedPointer = PseudoLinkedPointerNEW;
4510                            }
4511                         else if (memcmp(PseudoLinkedPointer+4+4+4,wrd,wrdlen) < 0) // go RP or MP
4512                            { // RW existence check - line below:
4513                              if ( *(char *)(PseudoLinkedPointer+4+4+4+wrdlen) != 0 ) // RW exists
4514                                 { // Here all 'P W P' section is repeated; the way of handling case when dynamic number of words in leaf
4515 // ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
4516 // ***** 'P W P' section 2 [
4517 // LW: existence check if ( *(char *)(PseudoLinkedPointer+4+4+4) != 0 )
4518 // RW: existence check if ( *(char *)(PseudoLinkedPointer+4+4+4+wrdlen) != 0 )
4519                         // here ALWAYS RW exists: no need for existence check - line below
4520                         // if ( *(char *)(PseudoLinkedPointer+4+4+4+wrdlen) != 0 )
4521                         if (memcmp(PseudoLinkedPointer+4+4+4+wrdlen,wrd,wrdlen) > 0) // go MP
4522                            { memcpy( &PseudoLinkedPointerNEW, PseudoLinkedPointer + 4, 4 ); //MP
4523                if (StackPtr > 8192*3-1-1) { printf( "\nLeprechaun: Failure! 'B-tree order 3' simulated stack overflow!\n" ); return( 13 );}
4524                BSTstack[StackPtr] = PseudoLinkedPointer; ++StackPtr; //pt to visited leaf
4525                BSTstack[StackPtr] = 4; ++StackPtr; //LPoffset=0;MPoffset=4;RPoffset=8;
4526                               PseudoLinkedPointer = PseudoLinkedPointerNEW;
4527                            }
4528                         else if (memcmp(PseudoLinkedPointer+4+4+4+wrdlen,wrd,wrdlen) < 0) // go RP
4529                            { // No ?W after RW - go RP
4530                              memcpy( &PseudoLinkedPointerNEW, PseudoLinkedPointer + 4 + 4, 4 ); //RP
4531                if (StackPtr > 8192*3-1-1) { printf( "\nLeprechaun: Failure! 'B-tree order 3' simulated stack overflow!\n" ); return( 13 );}
4532                BSTstack[StackPtr] = PseudoLinkedPointer; ++StackPtr; //pt to visited leaf
4533                BSTstack[StackPtr] = 8; ++StackPtr; //LPoffset=0;MPoffset=4;RPoffset=8;
4534                               PseudoLinkedPointer = PseudoLinkedPointerNEW;
4535                            }
4536                         else FoundInLinkedList = 1; // wrd is RW
4537 // ***** 'P W P' section 2 ]
4538 // ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
4539                                 }
4540                            else // RW empty - go MP
4541                               { memcpy( &PseudoLinkedPointerNEW, PseudoLinkedPointer + 4, 4 ); //MP
4542                if (StackPtr > 8192*3-1-1) { printf( "\nLeprechaun: Failure! 'B-tree order 3' simulated stack overflow!\n" ); return( 13 );}
4543                BSTstack[StackPtr] = PseudoLinkedPointer; ++StackPtr; //pt to visited leaf
4544                BSTstack[StackPtr] = 4; ++StackPtr; //LPoffset=0;MPoffset=4;RPoffset=8;
4545                                  PseudoLinkedPointer = PseudoLinkedPointerNEW;
4546                               }
4547                         }
4548                      else FoundInLinkedList = 1; // wrd is LW
4549 // ***** 'P W P' section ]
4550                      } // while
4551 // 2] if Search failed Trasirascht(pushing in stack PseudoLinkedPointer(visited LEAFs)) Search ]
4552
4553 // 3] Insert Iterative [
4554 //    There are total 4 situations:
4555 //    Case #1: Outer NODE(including ROOT) [  ][  ][  ][LW][  ]
4556 //    Case #2: Outer NODE(including ROOT) [  ][  ][  ][LW][RW]  Split Occurs   -----
4557 //    Case #3: ROOT                       [LP][MP][  ][LW][  ]                  |  'wrdUP' (wrdlen bytes)
4558 //    Case #4: Inner NODE(including ROOT) [LP][MP][RP][LW][RW]  Split Occurs   --- |  &
4559 //                                                                               | |  'PseudoLinkedPointerNEW' (ptr to NEW LEAF)
4560 //    There are total 2 situations for PARENT LEAF: <------------------------------  ARE GOING UP
4561 //    Case #3: [LP][MP][  ][LW][  ]
4562 //    Case #4: [LP][MP][RP][LW][RW]  Split Occurs
4563
4564 // ~  First deal alonely with the OUTER NODE(LEAF) where Search stopped i.e Case #1 & Case #2:
4565       POffsetInLEAF = BSTstack[--StackPtr];
4566       PseudoLinkedPointer = BSTstack[--StackPtr];
4567 // NOTE: ONE LEAF IS FULL ONLY WHEN LAST CELL FOR KEY(here RW) EXISTS!
4568 // RW: existence check if ( *(char *)(PseudoLinkedPointer+4+4+4+wrdlen) != 0 )
4569 if ( *(char *)(PseudoLinkedPointer+4+4+4+wrdlen) != 0 ) // If LEAF is full: Case #2
4570    { SplitOccured = 1; WORDcountDistinct++; bufNumberOfWords[LetterOffset]++;
```

```
4571            // ALlocate NEW LEAF:
4572            if( (unsigned long)(bufend[LetterOffset] - BufStart) + 2*wrdlen + 4 + 4 + 4 < GRMBLFoolAgain[(int)wrdlen] ) // +4 more for BST
    instead of LL; + more(see LEAF)
4573                    {
4574                    memcpy( &PseudoLinkedPointerNEW, &bufend[LetterOffset], 4 );
4575                    bufend[LetterOffset] = bufend[LetterOffset] + 4 + 4 + 4; // + 4 due to above commenting
4576                    bufend[LetterOffset] = bufend[LetterOffset] + 2*wrdlen;
4577                    if (MAXusedBuffer[wrdlen] < (unsigned long)(bufend[LetterOffset] - BufStart)) {MAXusedBuffer[wrdlen] = (unsigned
    long)(bufend[LetterOffset] - BufStart);}
4578                    }
4579            else
4580            { printf( "\nLeprechaun: Failure! Increment 'Memory for each Letter' parameter(third one)!\n" );
4581 fprintf( fp_outLOG, "Input File with a list of TEXTual Files: %s\n", argv[1] );
4582 fprintf( fp_outLOG, "Size of all TEXTual Files: %s\n", _ui64toaKAZEcomma(FilesLEN, llTOaDigits, 10) );
4583 fprintf( fp_outLOG, "Word count: %s of them %s distinct\n", _ui64toaKAZEcomma(WORDcount, llTOaDigits, 10), _ui64toaKAZEcomma((unsigned long
    long)WORDcountDistinct, llTOaDigits2, 10) );
4584 fprintf( fp_outLOG, "Number Of Files: %lu\n", NumberOfFiles );
4585 fprintf( fp_outLOG, "Number Of Lines: %lu\n", NumberOfLines );
4586 fprintf( fp_outLOG, "Allocated memory in MB: %lu\n", (unsigned long)(memory_size>>20)+1 );
4587 fprintf( fp_outLOG, "Total Attempts to Find/Put WORDs into B-trees order 3: %s\n", _ui64toaKAZEcomma(WORDcountAttemptsToPut, llTOaDigits, 10)
    );
4588 for( k = 1; k < 32; k++ )
4589 { 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
4590 }
4591 fprintf( fp_outLOG, "Used value for third parameter in KB: %lu\n", (unsigned long)Thunderwith );
4592              fprintf( fp_outLOG, "Leprechaun: Failure! Increment 'Memory for each Letter' parameter(third one)!\n\n");
4593              return( 1 );
4594            }
4595      if (POffsetInLEAF == 0) // wrd < LW
4596         {
4597                    memcpy( wrdUP, PseudoLinkedPointer+4+4+4, wrdlen ); // LW up
4598                    memcpy( PseudoLinkedPointer+4+4+4, wrd, wrdlen );   // wrd go to OLD LEAF
4599                    memcpy( PseudoLinkedPointerNEW+4+4+4, PseudoLinkedPointer+4+4+4+wrdlen, wrdlen ); // RW go to NEW LEAF
4600                    *(char *)(PseudoLinkedPointer+4+4+4+wrdlen) = 0;    // RW mark unused in OLD LEAF
4601         }
4602      if (POffsetInLEAF == 4) // LW < wrd < RW
4603         {
4604                    memcpy( wrdUP, wrd, wrdlen );                       // wrd up
4605                    memcpy( PseudoLinkedPointerNEW+4+4+4, PseudoLinkedPointer+4+4+4+wrdlen, wrdlen ); // RW go to NEW LEAF
4606                    *(char *)(PseudoLinkedPointer+4+4+4+wrdlen) = 0;    // RW mark unused in OLD LEAF
4607         }
4608      if (POffsetInLEAF == 8) // wrd > RW
4609         {
4610                    memcpy( wrdUP, PseudoLinkedPointer+4+4+4+wrdlen, wrdlen ); // RW up
4611                    *(char *)(PseudoLinkedPointer+4+4+4+wrdlen) = 0;    // RW mark unused in OLD LEAF
4612                    memcpy( PseudoLinkedPointerNEW+4+4+4, wrd, wrdlen );   // wrd go to NEW LEAF
4613         }
4614    }
4615 else // If LEAF is not full: Case #1
4616    { SplitOccured = 0; WORDcountDistinct++; bufNumberOfWords[LetterOffset]++;
4617      if (POffsetInLEAF == 0) // wrd < [LW][] so [LW][] -> [][LW] -> [wrd][LW]
4618         {
4619                    memcpy( PseudoLinkedPointer+4+4+4+wrdlen, PseudoLinkedPointer+4+4+4, wrdlen );
4620                    memcpy( PseudoLinkedPointer+4+4+4, wrd, wrdlen );
4621         }
4622      if (POffsetInLEAF == 4) // wrd > [LW][] so [LW][] -> [LW][wrd]
4623         {
4624                    memcpy( PseudoLinkedPointer+4+4+4+wrdlen, wrd, wrdlen );
4625         }
4626    }
4627
4628 if (SplitOccured != 0)
4629 {
4630 // ~  Second deal with the INNER NODE(S) i.e Case #3 & Case #4:
4631        while (StackPtr != 0 || SplitOccured != 0)
4632               {
4633        // 'PseudoLinkedPointerNEW' is new LEAF to be inserted
4634        // 'wrdUP' is NEW word to be inserted
4635        if (StackPtr != 0)
4636        {
4637           POffsetInLEAF = BSTstack[--StackPtr];
4638           PseudoLinkedPointer = BSTstack[--StackPtr];
4639 if ( *(char *)(PseudoLinkedPointer+4+4+4+wrdlen) != 0 ) // If LEAF is full: Case #4
4640    { SplitOccured = 1;
4641                    memcpy( wrdUPold, wrdUP, wrdlen ); // LW up
4642                    PseudoLinkedPointerNEWold = PseudoLinkedPointerNEW;
4643            // ALlocate NEW LEAF:
4644            if( (unsigned long)(bufend[LetterOffset] - BufStart) + 2*wrdlen + 4 + 4 + 4 < GRMBLFoolAgain[(int)wrdlen] ) // +4 more for BST
    instead of LL; + more(see LEAF)
4645                    {
4646                    memcpy( &PseudoLinkedPointerNEW, &bufend[LetterOffset], 4 );
4647                    bufend[LetterOffset] = bufend[LetterOffset] + 4 + 4 + 4; // + 4 due to above commenting
4648                    bufend[LetterOffset] = bufend[LetterOffset] + 2*wrdlen;
4649                    if (MAXusedBuffer[wrdlen] < (unsigned long)(bufend[LetterOffset] - BufStart)) {MAXusedBuffer[wrdlen] = (unsigned
    long)(bufend[LetterOffset] - BufStart);}
```



```
4650                    }
4651            else
4652            { printf( "\nLeprechaun: Failure! Increment 'Memory for each Letter' parameter(third one)!\n" );
4653 fprintf( fp_outLOG, "Input File with a list of TEXTual Files: %s\n", argv[1] );
4654 fprintf( fp_outLOG, "Size of all TEXTual Files: %s\n", _ui64toaKAZEcomma(FilesLEN, llTOaDigits, 10) );
4655 fprintf( fp_outLOG, "Word count: %s of them %s distinct\n", _ui64toaKAZEcomma(WORDcount, llTOaDigits, 10), _ui64toaKAZEcomma((unsigned long
    long)WORDcountDistinct, llTOaDigits2, 10) );
4656 fprintf( fp_outLOG, "Number Of Files: %lu\n", NumberOfFiles );
4657 fprintf( fp_outLOG, "Number Of Lines: %lu\n", NumberOfLines );
4658 fprintf( fp_outLOG, "Allocated memory in MB: %lu\n", (unsigned long)(memory_size>>20)+1 );
4659 fprintf( fp_outLOG, "Total Attempts to Find/Put WORDs into B-trees order 3: %s\n", _ui64toaKAZEcomma(WORDcountAttemptsToPut, llTOaDigits, 10)
    );
4660 for( k = 1; k < 32; k++ )
4661 { 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
4662 }
4663 fprintf( fp_outLOG, "Used value for third parameter in KB: %lu\n", (unsigned long)Thunderwith );
4664              fprintf( fp_outLOG, "Leprechaun: Failure! Increment 'Memory for each Letter' parameter(third one)!\n\n");
4665              return( 1 );
4666            }
4667      if (POffsetInLEAF == 0) // wrdUPold < LW
4668         {
4669                    memcpy( wrdUP, PseudoLinkedPointer+4+4+4, wrdlen ); // LW up
4670                    memcpy( PseudoLinkedPointer+4+4+4, wrdUPold, wrdlen );   // wrdUPold go to OLD LEAF
4671                    memcpy( PseudoLinkedPointerNEW+4+4+4, PseudoLinkedPointer+4+4+4+wrdlen, wrdlen ); // RW go to NEW LEAF
4672                    *(char *)(PseudoLinkedPointer+4+4+4+wrdlen) = 0;    // RW mark unused in OLD LEAF
4673                    // [LP](PseudoLinkedPointerNEWold)[MP][RP](wrdUPold)[LW][RW]       ------
4674                    //                 pair [LW] PseudoLinkedPointerNEW goes up              |
4675                    // PseudoLinkedPointer:                          PseudoLinkedPointerNEW: |
4676                    // [LP](PseudoLinkedPointerNEWold)[](wrdUPold)  [MP][RP][][RW]      <----
4677                    // no need to put zero in RP because logic is based on words existence:
4678                    memcpy( PseudoLinkedPointerNEW+0, PseudoLinkedPointer+4, 4 );
4679                    memcpy( PseudoLinkedPointerNEW+4, PseudoLinkedPointer+8, 4 );
4680                    memcpy( PseudoLinkedPointer+4, &PseudoLinkedPointerNEWold, 4 );
4681         }
4682      if (POffsetInLEAF == 4) // LW < wrdUPold < RW
4683         {
4684                    memcpy( wrdUP, wrdUPold, wrdlen );                        // wrdUPold up
4685                    memcpy( PseudoLinkedPointerNEW+4+4+4, PseudoLinkedPointer+4+4+4+wrdlen, wrdlen ); // RW go to NEW LEAF
4686                    *(char *)(PseudoLinkedPointer+4+4+4+wrdlen) = 0;    // RW mark unused in OLD LEAF
4687                    // [LP][MP](PseudoLinkedPointerNEWold)[RP][LW](wrdUPold)[RW]       ------
4688                    //                 pair [wrdUPold] PseudoLinkedPointerNEW goes up          |
4689                    // PseudoLinkedPointer:  PseudoLinkedPointerNEW:                           |
4690                    // [LP][MP][][LW]        (PseudoLinkedPointerNEWold)[RP][][RW]      <----
4691                    // no need to put zero in RP because logic is based on words existence:
4692                    memcpy( PseudoLinkedPointerNEW+0, &PseudoLinkedPointerNEWold, 4 );
4693                    memcpy( PseudoLinkedPointerNEW+4, PseudoLinkedPointer+8, 4 );
4694         }
4695      if (POffsetInLEAF == 8) // wrdUPold > RW
4696         {
4697                    memcpy( wrdUP, PseudoLinkedPointer+4+4+4+wrdlen, wrdlen ); // RW up
4698                    *(char *)(PseudoLinkedPointer+4+4+4+wrdlen) = 0;    // RW mark unused in OLD LEAF
4699                    memcpy( PseudoLinkedPointerNEW+4+4+4, wrdUPold, wrdlen );   // wrdUPold go to NEW LEAF
4700                    // [LP][MP][RP](PseudoLinkedPointerNEWold)[LW][RW](wrdUPold)       ------
4701                    //                 pair [RW] PseudoLinkedPointerNEW goes up              |
4702                    // PseudoLinkedPointer:  PseudoLinkedPointerNEW:                         |
4703                    // [LP][MP][][LW]        [RP](PseudoLinkedPointerNEWold)[](wrdUPold) <---
4704                    // no need to put zero in RP because logic is based on words existence:
4705                    memcpy( PseudoLinkedPointerNEW+0, PseudoLinkedPointer+8, 4 );
4706                    memcpy( PseudoLinkedPointerNEW+4, &PseudoLinkedPointerNEWold, 4 );
4707         }
4708    }
4709 else // If LEAF is not full: Case #3
4710    { SplitOccured = 0;
4711      if (POffsetInLEAF == 0) // wrdUP < [LW][] so [LW][] -> [][LW] -> [wrdUP][LW]
4712         {
4713                    memcpy( PseudoLinkedPointer+4+4+4+wrdlen, PseudoLinkedPointer+4+4+4, wrdlen );
4714                    memcpy( PseudoLinkedPointer+4+4+4, wrdUP, wrdlen );
4715                           // [LP][MP][] -> [LP][][MP] -> [LP][np][MP]
4716                    memcpy( PseudoLinkedPointer+8, PseudoLinkedPointer+4, 4 );
4717                    memcpy( PseudoLinkedPointer+4, &PseudoLinkedPointerNEW, 4 );
4718         }
4719      if (POffsetInLEAF == 4) // wrdUP > [LW][] so [LW][] -> [LW][wrdUP]
4720         {
4721                    memcpy( PseudoLinkedPointer+4+4+4+wrdlen, wrdUP, wrdlen );
4722                           // [LP][MP][] -> [LP][MP][np]
4723                    memcpy( PseudoLinkedPointer+8, &PseudoLinkedPointerNEW, 4 );
4724         }
4725           break;
4726    }
4727       }
4728       else // Empty stack means ROOT and more over ROOT is already splitted(Case #4 is off)
4729       {
4730     // If LEAF is not full: Case #3
4731     // THIS IS WHERE A NEW(SECOND) LEAF 'PseudoLinkedPointerROOT' must be allocated:
4732           if( (unsigned long)(bufend[LetterOffset] - BufStart) + 2*wrdlen + 4 + 4 + 4 < GRMBLFoolAgain[(int)wrdlen] ) // +4 more for BST
```

```
    instead of LL; + more(see LEAF)
                            {
                            memcpy( &PseudoLinkedPointerROOT, &bufend[LetterOffset], 4 );
                            bufend[LetterOffset] = bufend[LetterOffset] + 4 + 4 + 4; // + 4 due to above commenting
                            memcpy( bufend[LetterOffset], wrdUP, wrdlen );
                            bufend[LetterOffset] = bufend[LetterOffset] + 2*wrdlen;
                            if (MAXusedBuffer[wrdlen] < (unsigned long)(bufend[LetterOffset] - BufStart)) {MAXusedBuffer[wrdlen] = (unsigned
    long)(bufend[LetterOffset] - BufStart);}
                            }
                else
                { printf( "\nLeprechaun: Failure! Increment 'Memory for each Letter' parameter(third one)!\n" );
fprintf( fp_outLOG, "Input File with a list of TEXTual Files: %s\n", argv[1] );
fprintf( fp_outLOG, "Size of all TEXTual Files: %s\n", _ui64toaKAZEcomma(FilesLEN, llTOaDigits, 10) );
fprintf( fp_outLOG, "Word count: %s of them %s distinct\n", _ui64toaKAZEcomma(WORDcount, llTOaDigits, 10), _ui64toaKAZEcomma((unsigned long
    long)WORDcountDistinct, llTOaDigits2, 10) );
fprintf( fp_outLOG, "Number Of Files: %lu\n", NumberOfFiles );
fprintf( fp_outLOG, "Number Of Lines: %lu\n", NumberOfLines );
fprintf( fp_outLOG, "Allocated memory in MB: %lu\n", (unsigned long)(memory_size>>20)+1 );
fprintf( fp_outLOG, "Total Attempts to Find/Put WORDs into B-trees order 3: %s\n", _ui64toaKAZEcomma(WORDcountAttemptsToPut, llTOaDigits, 10)
    );
for( k = 1; k < 32; k++ )
{ 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
}
fprintf( fp_outLOG, "Used value for third parameter in KB: %lu\n", (unsigned long)Thunderwith );
              fprintf( fp_outLOG, "Leprechaun: Failure! Increment 'Memory for each Letter' parameter(third one)!\n\n");
              return( 1 );
            }
            // Here                    --  'PseudoLinkedPointerROOT' --
            //                         |           (wrdUP)           |
            // 'PseudoLinkedPointer' <--                              --> 'PseudoLinkedPointerNEW'
            //         (LW)                                                          (RW)
            memcpy( PseudoLinkedPointerROOT, &PseudoLinkedPointer, 4 );      // LP
            memcpy( PseudoLinkedPointerROOT+4, &PseudoLinkedPointerNEW, 4 ); // MP
            // Here must NEW ROOT be updated i.e. HASH table(SLOT) must point it:
            memcpy( BufStart+Slot, &PseudoLinkedPointerROOT, 4 );
           break; //because it is ROOT without split
        }
              } // while
} //if (SplitOccured != 0)
// 3] Insert Iterative ]
} //if (FoundInLinkedList == 0)
// ######################################## B-tree order 3 ]
} //if (BSTorBtree == 0)
} else { //if (BSTorBtree != 2) {
// External Btrees [

// _ ASCII code 095
// ` ASCII code 096  \
// a ASCII code 097  / In total 26+1+1 radix instead of 27 to avoid +1 for each '_', code 096 not used.
// z ASCII code 122
// The hash  for 'a_quadruplet_for_example' will be calculated for first 5 chars:
// = (byte1-'_')*28*28*28*28 + (byte2-'_')*28*28*28 + (byte3-'_')*28*28 + (byte4-'_')*28 + (byte5-'_')
// Hash slots are 28*28*28*28*28 = 17,210,368 each containing one 64bit pointer i.e. 8bytes in length.
// Hash size = 17,210,368*8 = 137,682,944 bytes
// When at end all these slots(17,210,368- Btrees) are traversed the outcome is a sorted wordlist - no need of sorting.

// D:\_KAZE_new-stuff\Leprechaun_quadrupleton_r14_minus>Leprechaun_quadrupleton.exe GRAFFITH_2048.lst GRAFFITH_2048.wrd z
// Leprechaun(Fast Greedy Word-Ripper), rev. 14_minus_quadrupleton, written by Svalqyatchx.
// Leprechaun: 'Oh, well, didn't you hear? Bigger is good, but jumbo is dear.'
// Kaze: Let's see what a 3-way hash + 6,602,752 Binary-Search-Trees can give us,
//       also the performance of a 3-way hash + 6,602,752 B-Trees of order 3,
//       also the performance of a 1-way hash + 17,210,368 external B-Trees of order 3.
// Size of input file with files for Leprechauning: 42140
// Allocating HASH memory 137,683,009 bytes ... OK
// Allocating/ZEROing 1,047,566 bytes swap file ... OK
// Size of Input TEXTual file: 33,470,581
// |; Word count: 3,045,077 of them 0 distinct; Done: 64/64
// ...
// Size of Input TEXTual file: 17,403,406
// /; Word count: 2,710,601,882 of them 0 distinct; Done: 64/64
// Bytes per second performance: 17,694,246B/s
// Words per second performance: 1,730,907W/s
// Leprechaun: Done.
//
// Leprechaun report:
// Number Of Trees(GREATER THE BETTER): 1,646,004

// TO DO - it is long overdue: at last make sort stage at end unnecessary - only traversing-and-dumping!

BufStart = pointerflush;
//   Slot = ((wrd[0]-'_')*28*28*28*28 + (wrd[1]-'_')*28*28*28 + (wrd[2]-'_')*28*28 + (wrd[3]-'_')*28 + (wrd[4]-'_'))<<3;
//   Slot = FNV1A_Hash_Jesteress_27bit(wrd, wrdlen)<<3; // Commented since r.14++++ because of passes.
Slot = FNV1A_Hash_Jesteress_27bit(wrd, wrdlen);

// Bug fix for all r.14+++ and below! [
```



```
memcpy( &wrd[(LongestLineInclusive+1+4)-4], &NULLsForWRD, 4 );
// Bug fix for all r.14+++ and below! ]

// Example: HashInBITS-HashChunkSizeInBITS=2
//          HashInBITS = 5
//          HashChunkSizeInBITS = 3
//          RipPasses = 1<<(HashInBITS-HashChunkSizeInBITS) i.e. 1<<2 which is 4 i.e. 32 slots with 4 passes 8 slots each.
//          00??? 5bits 0-7
//          01??? 5bits 8-15
//          10??? 5bits 16-23
//          11??? 5bits 24-31
if ( (Slot>>HashChunkSizeInBITS) == RipPasses ) {
Slot = Slot<<3;

//Slot = 0; // One Tree only!
memcpy( &PseudoLinkedPointer_64, BufStart+Slot, 8 );

// ######################################## B-tree order 3 fragment 64bit [
//
// LEAF structure: [LeftPointer][MiddlePointer][RightPointer][LeftWord][RightWord]
//                  4bytes       4bytes         4bytes        wrdlen    wrdlen
//                                                            *         *          <- if *(char *)==0 means the word cell is empty
// ALL B-tree order 3 fragment consists of 3 sub-fragments:
// 1] Search 2] if Search failed Trasirascht(pushing in stack PseudoLinkedPointer(visited LEAFs)) Search 3] Insert Iterative

// LEAF_64 structure: [LeftPointer][MiddlePointer][RightPointer][LeftWord]              [RightWord]
//                     8bytes       8bytes         8bytes        LongestLineInclusive+1+4 LongestLineInclusive+1+4
//                                                               *         *          <- if *(char *)==0 means the word cell is empty
// 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
    choice ...
// Note: BufEnd_64 in fact is the first free position after the BUFFER END!

// 1] Search [
                    if (PseudoLinkedPointer_64 == 0) // means EMPTY-SLOT
                    {
if ( REUSE != 2 ) { //  REUSE [ // This line comes since r16
if (DoNotInsertFlag == 0)
{ // This line comes since r15FIXFIX+ [[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[
            //if( (unsigned long)(bufend[LetterOffset] - BufStart) + 2*wrdlen + 4 + 4 + 4 < GRMBLFoolAgain[(int)wrdlen] ) // +4 more for BST
    instead of LL; + more(see LEAF)
            //          {
            //          memcpy( BufStart+Slot, &bufend[LetterOffset], 4 );
            //          bufend[LetterOffset] = bufend[LetterOffset] + 4 + 4 + 4; // + 4 due to above commenting
            //          memcpy( bufend[LetterOffset], wrd, wrdlen ); WORDcountDistinct++; bufNumberOfWords[LetterOffset]++;
            //          bufend[LetterOffset] = bufend[LetterOffset] + 2*wrdlen;
            //          if (MAXusedBuffer[wrdlen] < (unsigned long)(bufend[LetterOffset] - BufStart)) {MAXusedBuffer[wrdlen] = (unsigned
    long)(bufend[LetterOffset] - BufStart);}
            //          }
            if( 8 + 8 + 8 + 2*(LongestLineInclusive+1+4) < size_in64_L14 - (BufEnd_64-(unsigned long long)pointerflush_64) ) // the longest
    wrdlen is LongestLineInclusive but actual is LongestLineInclusive(+ CR char)
                        {
                          memcpy( BufStart+Slot, &BufEnd_64, 8 );
              BufEnd_64 = BufEnd_64 + 8 + 8 + 8;
                        if (BSTorBtree == 2) {
              fsetpos(fp_outRG, &BufEnd_64);
              fwrite(wrd, wrdlen, 1, fp_outRG); WORDcountDistinct++;
              //fwrite(&OneChar_ieByte, 1, 1, fp_outRG); // Write ZERO ASCII code
              // r.14++ The above line was commented because the pool is already ZEROed.
                        } else { // ########## 64bit memory manipulations [
              memcpy( (char *)BufEnd_64, wrd, wrdlen ); WORDcountDistinct++;
                        } // ########## 64bit memory manipulations ]
              BufEnd_64 = BufEnd_64 + 2*(LongestLineInclusive+1+4);
              //fsetpos(fp_outRG, &BufEnd_64);
                        }
            else
            { printf( "\nLeprechaun: Failure! Increment 'Memory for each Letter' parameter(third one)!\n" );
fprintf( fp_outLOG, "Word count: %s of them %s distinct\n", _ui64toaKAZEcomma(WORDcount, llTOaDigits, 10), _ui64toaKAZEcomma((unsigned long
    long)WORDcountDistinct, llTOaDigits2, 10) );
fprintf( fp_outLOG, "Allocated memory: %s bytes\n", _ui64toaKAZEcomma(size_in64_L14, llTOaDigits, 10) );
fprintf( fp_outLOG, "Total Attempts to Find/Put WORDs into B-trees order 3: %s\n", _ui64toaKAZEcomma(WORDcountAttemptsToPut, llTOaDigits, 10)
    );
fprintf( fp_outLOG, "Used value for third parameter in KB: %lu\n", (unsigned long)Thunderwith );
              fprintf( fp_outLOG, "Leprechaun: Failure! Increment 'Memory for each Letter' parameter(third one)!\n\n");
              return( 1 );
            }
                        FoundInLinkedList = 1;
} // This line comes since r15FIXFIX+ ]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]
}
                    }
                    else // means USED-SLOT
                    { FoundInLinkedList = 0;
                  StackPtr = 0;
//                      while (PseudoLinkedPointer != 0 && FoundInLinkedList == 0)
                      while (PseudoLinkedPointer_64 != 0 && FoundInLinkedList == 0)
                      {
// ***** 'P W P' section [
// LW: existence check if ( *(char *)(PseudoLinkedPointer+4+4+4) != 0 )
// RW: existence check if ( *(char *)(PseudoLinkedPointer+4+4+4+wrdlen) != 0 )
```

```
                        // here ALWAYS LW exists: no need for existence check - line below
                        // if ( *(char *)(PseudoLinkedPointer+4+4+4) != 0 )
//                        if (memcmp(PseudoLinkedPointer+4+4+4,wrd,wrdlen) > 0) // go LP
              // [  //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
              //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8;
              //fsetpos(fp_outRG, &PseudoLinkedPointerAUX_64);
              //fread(&FourGramL[0], (LongestLineInclusive+1+4), 1, fp_outRG);
              // ]  //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
              // [  //r.14+
              PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64;
                        if (BSTorBtree == 2) {
              fsetpos(fp_outRG, &PseudoLinkedPointerAUX_64);
              fread(&LEAF[0], 8+8+8+2*(LongestLineInclusive+1+4), 1, fp_outRG);
                        } else { // ########## 64bit memory manipulations [
              memcpy( &LEAF[0], (char *)PseudoLinkedPointerAUX_64, 8+8+8+2*(LongestLineInclusive+1+4) );
                        } // ########## 64bit memory manipulations ]
              memcpy( &FourGramL[0], &LEAF[8 + 8 + 8], (LongestLineInclusive+1+4) );
              // ]  //r.14+
                        if (strcmpKAZE13(FourGramL, wrd) > 0) // go LP
//                          { memcpy( &PseudoLinkedPointerNEW, PseudoLinkedPointer + 0, 4 ); //LP
//                            PseudoLinkedPointer = PseudoLinkedPointerNEW;
//                          }
                        {
              // [  //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
              //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 0; //LP
              // ]  //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
                if (StackPtr > 8192*3-1-1) { printf( "\nLeprechaun: Failure! 'B-tree order 3' simulated stack overflow!\n" ); return( 13 );}
                BSTstack[StackPtr] = PseudoLinkedPointer_64; ++StackPtr; //pt to visited leaf
                BSTstack[StackPtr] = 0; ++StackPtr; //LPoffset=0;MPoffset=8;RPoffset=16;
              // [  //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
              //fsetpos(fp_outRG, &PseudoLinkedPointerAUX_64);
              //fread(&PseudoLinkedPointer_64, 8, 1, fp_outRG);
              // ]  //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
              // [  //r.14+
              memcpy( &PseudoLinkedPointer_64, &LEAF[0], 8 );
              // ]  //r.14+
                        }
//                        else if (memcmp(PseudoLinkedPointer+4+4+4,wrd,wrdlen) < 0) // go RP or MP
                      else if (strcmpKAZE13(FourGramL, wrd) < 0) // go RP or MP
                        { // RW existence check - line below:
//                          if ( *(char *)(PseudoLinkedPointer+4+4+4+wrdlen) != 0 ) // RW exists
              // [  //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
              //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + 8 + (LongestLineInclusive+1+4); //RW
              //fsetpos(fp_outRG, &PseudoLinkedPointerAUX_64);
              //fread(&SomeByte, 1, 1, fp_outRG);
              // ]  //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
              // [  //r.14+
              memcpy( &SomeByte, &LEAF[8 + 8 + 8 + (LongestLineInclusive+1+4)], 1 );
              // ]  //r.14+
              if (SomeByte != 0 ) // RW exists
                          { // Here all 'P W P' section is repeated; the way of handling case when dynamic number of words in leaf
// ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
// ***** 'P W P' section 2 [
// LW: existence check if ( *(char *)(PseudoLinkedPointer+4+4+4) != 0 )
// RW: existence check if ( *(char *)(PseudoLinkedPointer+4+4+4+wrdlen) != 0 )
                        // here ALWAYS RW exists: no need for existence check - line below
                        // if ( *(char *)(PseudoLinkedPointer+4+4+4+wrdlen) != 0 )
//                        if (memcmp(PseudoLinkedPointer+4+4+4+wrdlen,wrd,wrdlen) > 0) // go MP
              // [  //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
              //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1+4);
              //fsetpos(fp_outRG, &PseudoLinkedPointerAUX_64);
              //fread(&FourGramL[0], (LongestLineInclusive+1+4), 1, fp_outRG);
              // ]  //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
              // [  //r.14+
              memcpy( &FourGramL[0], &LEAF[8 + 8 + 8 + (LongestLineInclusive+1+4)], (LongestLineInclusive+1+4) );
              // ]  //r.14+
                        if (strcmpKAZE13(FourGramL, wrd) > 0) // go MP
//                          { memcpy( &PseudoLinkedPointerNEW, PseudoLinkedPointer + 4, 4 ); //MP
//                            PseudoLinkedPointer = PseudoLinkedPointerNEW;
//                          }
                        {
              // [  //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
              //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8; //MP
              // ]  //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
                if (StackPtr > 8192*3-1-1) { printf( "\nLeprechaun: Failure! 'B-tree order 3' simulated stack overflow!\n" ); return( 13 );}
                BSTstack[StackPtr] = PseudoLinkedPointer_64; ++StackPtr; //pt to visited leaf
                BSTstack[StackPtr] = 8; ++StackPtr; //LPoffset=0;MPoffset=8;RPoffset=16;
              // [  //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
              //fsetpos(fp_outRG, &PseudoLinkedPointerAUX_64);
              //fread(&PseudoLinkedPointer_64, 8, 1, fp_outRG);
              // ]  //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
              // [  //r.14+
              memcpy( &PseudoLinkedPointer_64, &LEAF[8], 8 );
              // ]  //r.14+
                        }
//                        else if (memcmp(PseudoLinkedPointer+4+4+4+wrdlen,wrd,wrdlen) < 0) // go RP
                      else if (strcmpKAZE13(FourGramL, wrd) < 0) // go RP
                        { // No ?W after RW - go RP
```



```
//                          memcpy( &PseudoLinkedPointerNEW, PseudoLinkedPointer + 4 + 4, 4 ); //RP
//                          PseudoLinkedPointer = PseudoLinkedPointerNEW;
//                        }
                        {
              // [  //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
              //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8; //RP
              // ]  //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
                if (StackPtr > 8192*3-1-1) { printf( "\nLeprechaun: Failure! 'B-tree order 3' simulated stack overflow!\n" ); return( 13 );}
                BSTstack[StackPtr] = PseudoLinkedPointer_64; ++StackPtr; //pt to visited leaf
                BSTstack[StackPtr] = 16; ++StackPtr; //LPoffset=0;MPoffset=8;RPoffset=16;
              // [  //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
              //fsetpos(fp_outRG, &PseudoLinkedPointerAUX_64);
              //fread(&PseudoLinkedPointer_64, 8, 1, fp_outRG);
              // ]  //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
              // [  //r.14+
              memcpy( &PseudoLinkedPointer_64, &LEAF[8 + 8], 8 );
              // ]  //r.14+
                        }
                      else { FoundInLinkedList = 1; // wrd is RW
              // Counter [
                        if (BSTorBtree == 2) {
              fsetpos(fp_outRG, &PseudoLinkedPointerAUX_64);
                        }
              memcpy( &CounterOccurrencies, &FourGramL[(LongestLineInclusive+1+4)-4], 4 );

                        // r16 [
                        if ( REUSE == 2 ) {
                        if (*argv[k_FIX] == 'W')
                                fprintf(fp_out, "%s\t%s\r\n", _ui64toaKAZEzerocomma(CounterOccurrencies+1, llTOaDigits2, 10)+(26-9), wrd);
//WORDcountBOTTOM++;
                        if (*argv[k_FIX] == 'w')
                                fprintf(fp_out, "%s\r\n", wrd); //WORDcountBOTTOM++;
                        }
                        // r16 ]

                        if ( REUSE != 2 ) {  // r16
              if (CounterOccurrencies<9999999) CounterOccurrencies++;
              memcpy( &FourGramL[(LongestLineInclusive+1+4)-4], &CounterOccurrencies, 4 );
              // [  //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
              //fwrite(&FourGramL[0], (LongestLineInclusive+1+4), 1, fp_outRG);
              // ]  //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
              // [  //r.14+
              memcpy( &LEAF[8 + 8 + 8 + (LongestLineInclusive+1+4)], &FourGramL[0], (LongestLineInclusive+1+4) );
                        if (BSTorBtree == 2) {
              fwrite(&LEAF[0], 8+8+8+2*(LongestLineInclusive+1+4), 1, fp_outRG);
                        } else { // ########## 64bit memory manipulations [
              memcpy( (char *)PseudoLinkedPointerAUX_64, &LEAF[0], 8+8+8+2*(LongestLineInclusive+1+4) );
                        } // ########## 64bit memory manipulations ]
              // ]  //r.14+
                        } // r16
              // Counter ]
                      }
                        WORDcountAttemptsToPut++;
// ***** 'P W P' section 2 ]
// ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                        }
                      else // RW empty - go MP
//                        { memcpy( &PseudoLinkedPointerNEW, PseudoLinkedPointer + 4, 4 ); //MP
//                          PseudoLinkedPointer = PseudoLinkedPointerNEW;
//                        }
                        {
              // [  //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
              //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8; //MP
              // ]  //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
                if (StackPtr > 8192*3-1-1) { printf( "\nLeprechaun: Failure! 'B-tree order 3' simulated stack overflow!\n" ); return( 13 );}
                BSTstack[StackPtr] = PseudoLinkedPointer_64; ++StackPtr; //pt to visited leaf
                BSTstack[StackPtr] = 8; ++StackPtr; //LPoffset=0;MPoffset=8;RPoffset=16;
              // [  //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
              //fsetpos(fp_outRG, &PseudoLinkedPointerAUX_64);
              //fread(&PseudoLinkedPointer_64, 8, 1, fp_outRG);
              // ]  //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
              // [  //r.14+
              memcpy( &PseudoLinkedPointer_64, &LEAF[8], 8 );
              // ]  //r.14+
                        }
                      else { FoundInLinkedList = 1; // wrd is LW
              // Counter [
                        if (BSTorBtree == 2) {
              fsetpos(fp_outRG, &PseudoLinkedPointerAUX_64);
                        }
              memcpy( &CounterOccurrencies, &FourGramL[(LongestLineInclusive+1+4)-4], 4 );

                        // r16 [
                        if ( REUSE == 2 ) {
                        if (*argv[k_FIX] == 'W')
                                fprintf(fp_out, "%s\t%s\r\n", _ui64toaKAZEzerocomma(CounterOccurrencies+1, llTOaDigits2, 10)+(26-9), wrd);
//WORDcountBOTTOM++;
```

```
                        if (*argv[k_FIX] == 'w')
                                fprintf(fp_out, "%s\r\n", wrd); //WORDcountBOTTOM++;
                        }
                        // r16 ]

                        if ( REUSE != 2 ) {  // r16
                if (CounterOccurrencies<9999999) CounterOccurrencies++;
                memcpy( &FourGramL[(LongestLineInclusive+1+4)-4], &CounterOccurrencies, 4 );
                // [  //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
                //fwrite(&FourGramL[0], (LongestLineInclusive+1+4), 1, fp_outRG);
                // ]  //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
                // [  //r.14+
                memcpy( &LEAF[8 + 8 + 8], &FourGramL[0], (LongestLineInclusive+1+4) );
                        if (BSTorBtree == 2) {
                fwrite(&LEAF[0], 8+8+8+2*(LongestLineInclusive+1+4), 1, fp_outRG);
                        } else { // ########## 64bit memory manipulations [
                memcpy( (char *)PseudoLinkedPointerAUX_64, &LEAF[0], 8+8+8+2*(LongestLineInclusive+1+4) );
                        } // ########## 64bit memory manipulations ]
                // ]  //r.14+
                        } // r16
                // Counter ]
                    }
                        WORDcountAttemptsToPut++;
// ***** 'P W P' section ]
                    } // while
                        WORDcountAttemptsToPut--; // - 1 due to BST way of counting i.e. direct hash hit is not counted only successors
                  }
// 1] Search ]

if ( REUSE != 2 ) { //  REUSE [ // This line comes since r16
if (DoNotInsertFlag == 0) { // This line comes since r15FIXFIX+
if (FoundInLinkedList == 0)
{
/*
// ==================== [ The whole section/sub-fragment 2 is commented due to great time differences for Internal_vs_External memory accesses - it is far more cheap to have the STACK overhead (moved to sub-fragment 1) ] ==================== [
// 2] if Search failed Trasirascht(pushing in stack PseudoLinkedPointer(visited LEAFs)) Search [
//    'TracingSearch' is the same as 'Search' except that adds the trail in my simulated stack,
//    the goal is not to waste time in 'Search' by dealing with no needed trail in case of not 'Insert'.
//    Simulated stack contains pairs of 'Address of ParentLEAF' + 'Offset of ParentPointer in ParentLEAF i.e. 0 for LP, 4 for MP, 8 for RP'.
//    'Offset ...' saves unnecessary comparisons of NEWword which after splitting goes up.
                memcpy( &PseudoLinkedPointer, BufStart+Slot, 4 );
              StackPtr = 0;
                    while (PseudoLinkedPointer != 0)
                    {
// ***** 'P W P' section [
// LW: existence check if ( *(char *)(PseudoLinkedPointer+4+4+4) != 0 )
// RW: existence check if ( *(char *)(PseudoLinkedPointer+4+4+4+wrdlen) != 0 )
                        // here ALWAYS LW exists: no need for existence check - line below
                        // if ( *(char *)(PseudoLinkedPointer+4+4+4) != 0 )
                        if (memcmp(PseudoLinkedPointer+4+4+4,wrd,wrdlen) > 0) // go LP
                           { memcpy( &PseudoLinkedPointerNEW, PseudoLinkedPointer + 0, 4 ); //LP
              if (StackPtr > 8192*3-1-1) { printf( "\nLeprechaun: Failure! 'B-tree order 3' simulated stack overflow!\n" ); return( 13 );}
              BSTstack[StackPtr] = PseudoLinkedPointer; ++StackPtr; //pt to visited leaf
              BSTstack[StackPtr] = 0; ++StackPtr; //LPoffset=0;MPoffset=4;RPoffset=8;
                              PseudoLinkedPointer = PseudoLinkedPointerNEW;
                           }
                        else if (memcmp(PseudoLinkedPointer+4+4+4,wrd,wrdlen) < 0) // go RP or MP
                           { // RW existence check - line below:
                             if ( *(char *)(PseudoLinkedPointer+4+4+4+wrdlen) != 0 ) // RW exists
                                { // Here all 'P W P' section is repeated; the way of handling case when dynamic number of words in leaf
// ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
// ***** 'P W P' section 2 [
// LW: existence check if ( *(char *)(PseudoLinkedPointer+4+4+4) != 0 )
// RW: existence check if ( *(char *)(PseudoLinkedPointer+4+4+4+wrdlen) != 0 )
                        // here ALWAYS RW exists: no need for existence check - line below
                        // if ( *(char *)(PseudoLinkedPointer+4+4+4+wrdlen) != 0 )
                        if (memcmp(PseudoLinkedPointer+4+4+4+wrdlen,wrd,wrdlen) > 0) // go MP
                           { memcpy( &PseudoLinkedPointerNEW, PseudoLinkedPointer + 4, 4 ); //MP
              if (StackPtr > 8192*3-1-1) { printf( "\nLeprechaun: Failure! 'B-tree order 3' simulated stack overflow!\n" ); return( 13 );}
              BSTstack[StackPtr] = PseudoLinkedPointer; ++StackPtr; //pt to visited leaf
              BSTstack[StackPtr] = 4; ++StackPtr; //LPoffset=0;MPoffset=4;RPoffset=8;
                              PseudoLinkedPointer = PseudoLinkedPointerNEW;
                           }
                        else if (memcmp(PseudoLinkedPointer+4+4+4+wrdlen,wrd,wrdlen) < 0) // go RP
                           { // No ?W after RW - go RP
                             memcpy( &PseudoLinkedPointerNEW, PseudoLinkedPointer + 4 + 4, 4 ); //RP
              if (StackPtr > 8192*3-1-1) { printf( "\nLeprechaun: Failure! 'B-tree order 3' simulated stack overflow!\n" ); return( 13 );}
              BSTstack[StackPtr] = PseudoLinkedPointer; ++StackPtr; //pt to visited leaf
              BSTstack[StackPtr] = 8; ++StackPtr; //LPoffset=0;MPoffset=4;RPoffset=8;
                              PseudoLinkedPointer = PseudoLinkedPointerNEW;
                           }
                        else FoundInLinkedList = 1; // wrd is RW
// ***** 'P W P' section 2 ]
// ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                                }
                             else // RW empty - go MP
                                { memcpy( &PseudoLinkedPointerNEW, PseudoLinkedPointer + 4, 4 ); //MP
```



```
              if (StackPtr > 8192*3-1-1) { printf( "\nLeprechaun: Failure! 'B-tree order 3' simulated stack overflow!\n" ); return( 13 );}
              BSTstack[StackPtr] = PseudoLinkedPointer; ++StackPtr; //pt to visited leaf
              BSTstack[StackPtr] = 4; ++StackPtr; //LPoffset=0;MPoffset=4;RPoffset=8;
                              PseudoLinkedPointer = PseudoLinkedPointerNEW;
                           }
                        }
                     else FoundInLinkedList = 1; // wrd is LW
// ***** 'P W P' section ]
                    } // while
// 2] if Search failed Trasirascht(pushing in stack PseudoLinkedPointer(visited LEAFs)) Search ]
// ==================== [ The whole section/sub-fragment 2 is commented due to great time differences for Internal_vs_External memory accesses - it is far more cheap to have the STACK overhead (moved to sub-fragment 1) ] ==================== ]
*/

// 3] Insert Iterative [
//    There are total 4 situations:
//    Case #1: Outer NODE(including ROOT) [  ][  ][  ][LW][  ]
//    Case #2: Outer NODE(including ROOT) [  ][  ][  ][LW][RW]  Split Occurs   -----
//    Case #3: ROOT                       [LP][MP][  ][LW][  ]                  |  'wrdUP' (wrdlen bytes)
//    Case #4: Inner NODE(including ROOT) [LP][MP][RP][LW][RW]  Split Occurs   --- |  &
//                                                                               | |  'PseudoLinkedPointerNEW' (ptr to NEW LEAF)
//    There are total 2 situations for PARENT LEAF: <------------------------------  ARE GOING UP
//    Case #3: [LP][MP][  ][LW][  ]
//    Case #4: [LP][MP][RP][LW][RW]  Split Occurs

// ~  First deal alonely with the OUTER NODE(LEAF) where Search stopped i.e Case #1 & Case #2:
      POffsetInLEAF = BSTstack[--StackPtr];
      PseudoLinkedPointer_64 = BSTstack[--StackPtr];
// NOTE: ONE LEAF IS FULL ONLY WHEN LAST CELL FOR KEY(here RW) EXISTS!
// RW: existence check if ( *(char *)(PseudoLinkedPointer+4+4+4+wrdlen) != 0 )
//if ( *(char *)(PseudoLinkedPointer+4+4+4+wrdlen) != 0 ) // If LEAF is full: Case #2
              // [  //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
              //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1+4); //RW
              //fsetpos(fp_outRG, &PseudoLinkedPointerAUX_64);
              //fread(&SomeByte, 1, 1, fp_outRG);
              // ]  //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
              // [  //r.14+
              PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64;
                      if (BSTorBtree == 2) {
              fsetpos(fp_outRG, &PseudoLinkedPointerAUX_64);
              fread(&LEAF[0], 8+8+8+2*(LongestLineInclusive+1+4), 1, fp_outRG);
                      } else { // ########## 64bit memory manipulations [
              memcpy( &LEAF[0], (char *)PseudoLinkedPointerAUX_64, 8+8+8+2*(LongestLineInclusive+1+4) );
                      } // ########## 64bit memory manipulations ]
              memcpy( &SomeByte, &LEAF[8 + 8 + 8 + (LongestLineInclusive+1+4)], 1 );
              // ]  //r.14+
              if (SomeByte != 0 ) // RW exists
    { SplitOccured = 1; WORDcountDistinct++;
            // ALlocate NEW LEAF:
//            if( (unsigned long)(bufend[LetterOffset] - BufStart) + 2*wrdlen + 4 + 4 + 4 < GRMBLFoolAgain[(int)wrdlen] ) // +4 more for BST instead of LL; + more(see LEAF)
//                        {
//                        memcpy( &PseudoLinkedPointerNEW, &bufend[LetterOffset], 4 );
//                        bufend[LetterOffset] = bufend[LetterOffset] + 4 + 4 + 4; // + 4 due to above commenting
//                        bufend[LetterOffset] = bufend[LetterOffset] + 2*wrdlen;
//                        if (MAXusedBuffer[wrdlen] < (unsigned long)(bufend[LetterOffset] - BufStart)) {MAXusedBuffer[wrdlen] = (unsigned long)(bufend[LetterOffset] - BufStart);}
//                        }
            if( 8 + 8 + 8 + 2*(LongestLineInclusive+1+4) < size_in64_L14 - (BufEnd_64-(unsigned long long)pointerflush_64) ) // the longest wrdlen is LongestLineInclusive but actual is LongestLineInclusive(+ CR char)
                        {
              PseudoLinkedPointerNEW_64 = BufEnd_64;
              BufEnd_64 = BufEnd_64 + 8 + 8 + 8;
              BufEnd_64 = BufEnd_64 + 2*(LongestLineInclusive+1+4);
                        }
              else
              { printf( "\nLeprechaun: Failure! Increment 'Memory for each Letter' parameter(third one)!\n" );
      fprintf( fp_outLOG, "Word count: %s of them %s distinct\n", _ui64toaKAZEcomma(WORDcount, llTOaDigits, 10), _ui64toaKAZEcomma((unsigned long long)WORDcountDistinct, llTOaDigits2, 10) );
      fprintf( fp_outLOG, "Allocated memory: %s bytes\n", _ui64toaKAZEcomma(size_in64_L14, llTOaDigits, 10) );
      fprintf( fp_outLOG, "Total Attempts to Find/Put WORDs into B-trees order 3: %s\n", _ui64toaKAZEcomma(WORDcountAttemptsToPut, llTOaDigits, 10) );
      fprintf( fp_outLOG, "Used value for third parameter in KB: %lu\n", (unsigned long)Thunderwith );
      fprintf( fp_outLOG, "Leprechaun: Failure! Increment 'Memory for each Letter' parameter(third one)!\n\n");
                return( 1 );
              }
      if (POffsetInLEAF == 0) // wrd < LW
         {
//                        memcpy( wrdUP, PseudoLinkedPointer+4+4+4, wrdlen ); // LW up
//                        memcpy( PseudoLinkedPointer+4+4+4, wrd, wrdlen );   // wrd go to OLD LEAF
//                        memcpy( PseudoLinkedPointerNEW+4+4+4, PseudoLinkedPointer+4+4+4+wrdlen, wrdlen ); // RW go to NEW LEAF
//                        *(char *)(PseudoLinkedPointer+4+4+4+wrdlen) = 0;    // RW mark unused in OLD LEAF
              // [  //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
              //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8;
              //fsetpos(fp_outRG, &PseudoLinkedPointerAUX_64);
              //fread(&wrdUP[0], (LongestLineInclusive+1+4), 1, fp_outRG);
               //fsetpos(fp_outRG, &PseudoLinkedPointerAUX_64);
               //fwrite(&wrd[0], (LongestLineInclusive+1+4), 1, fp_outRG);
```

```
5239                //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1+4);
5240                //fsetpos(fp_outRG, &PseudoLinkedPointerAUX_64);
5241                //fread(&wrdAUX[0], (LongestLineInclusive+1+4), 1, fp_outRG);
5242                //PseudoLinkedPointerAUX_64 = PseudoLinkedPointerNEW_64 + 8 + 8 + 8;
5243                //fsetpos(fp_outRG, &PseudoLinkedPointerAUX_64);
5244                //fwrite(&wrdAUX[0], (LongestLineInclusive+1+4), 1, fp_outRG);
5245                 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1+4);
5246                 //fsetpos(fp_outRG, &PseudoLinkedPointerAUX_64);
5247                 //fwrite(&OneChar_ieByte, 1, 1, fp_outRG); // Write ZERO ASCII code
5248              // ]  //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
5249              // [  //r.14+
5250              memcpy( &wrdUP[0], &LEAF[8 + 8 + 8], (LongestLineInclusive+1+4) );
5251              memcpy( &LEAF[8 + 8 + 8], &wrd[0], (LongestLineInclusive+1+4) );
5252              memcpy( &wrdAUX[0], &LEAF[8 + 8 + 8 + (LongestLineInclusive+1+4)], (LongestLineInclusive+1+4) );
5253              // Here reordering (of writing wrdAUX) is needed to avoid seek the position NEW and stupidly to seek again OLD/current position!
5254              memcpy( &LEAF[8 + 8 + 8 + (LongestLineInclusive+1+4)], &OneChar_ieByte, 1 );
5255                      if (BSTorBtree == 2) {
5256              fsetpos(fp_outRG, &PseudoLinkedPointerAUX_64);
5257              fwrite(&LEAF[0], 8+8+8+2*(LongestLineInclusive+1+4), 1, fp_outRG);
5258                      } else { // ########## 64bit memory manipulations [
5259              memcpy( (char *)PseudoLinkedPointerAUX_64, &LEAF[0], 8+8+8+2*(LongestLineInclusive+1+4) );
5260                      } // ########## 64bit memory manipulations ]
5261                PseudoLinkedPointerAUX_64 = PseudoLinkedPointerNEW_64 + 8 + 8 + 8;
5262                      if (BSTorBtree == 2) {
5263                fsetpos(fp_outRG, &PseudoLinkedPointerAUX_64);
5264                fwrite(&wrdAUX[0], (LongestLineInclusive+1+4), 1, fp_outRG);
5265                      } else { // ########## 64bit memory manipulations [
5266              memcpy( (char *)PseudoLinkedPointerAUX_64, &wrdAUX[0], (LongestLineInclusive+1+4) );
5267                      } // ########## 64bit memory manipulations ]
5268              // ]  //r.14+
5269            }
5270          if (POffsetInLEAF == 8) // LW < wrd < RW
5271            {
5272 //                     memcpy( wrdUP, wrd, wrdlen );                       // wrd up
5273 //                     memcpy( PseudoLinkedPointerNEW+4+4+4, PseudoLinkedPointer+4+4+4+wrdlen, wrdlen ); // RW go to NEW LEAF
5274 //                     *(char *)(PseudoLinkedPointer+4+4+4+wrdlen) = 0;    // RW mark unused in OLD LEAF
5275                       memcpy( wrdUP, wrd, (LongestLineInclusive+1+4) );     // wrd up
5276              // [  //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
5277                //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1+4);
5278                //fsetpos(fp_outRG, &PseudoLinkedPointerAUX_64);
5279                //fread(&wrdAUX[0], (LongestLineInclusive+1+4), 1, fp_outRG);
5280                //PseudoLinkedPointerAUX_64 = PseudoLinkedPointerNEW_64 + 8 + 8 + 8;
5281                //fsetpos(fp_outRG, &PseudoLinkedPointerAUX_64);
5282                //fwrite(&wrdAUX[0], (LongestLineInclusive+1+4), 1, fp_outRG);
5283                 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1+4);
5284                 //fsetpos(fp_outRG, &PseudoLinkedPointerAUX_64);
5285                 //fwrite(&OneChar_ieByte, 1, 1, fp_outRG); // Write ZERO ASCII code
5286              // ]  //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
5287              // [  //r.14+
5288              memcpy( &wrdAUX[0], &LEAF[8 + 8 + 8 + (LongestLineInclusive+1+4)], (LongestLineInclusive+1+4) );
5289              // Here reordering (of writing wrdAUX) is needed to avoid seek the position NEW and stupidly to seek again OLD/current position!
5290              memcpy( &LEAF[8 + 8 + 8 + (LongestLineInclusive+1+4)], &OneChar_ieByte, 1 );
5291                      if (BSTorBtree == 2) {
5292              fsetpos(fp_outRG, &PseudoLinkedPointerAUX_64);
5293              fwrite(&LEAF[0], 8+8+8+2*(LongestLineInclusive+1+4), 1, fp_outRG);
5294                      } else { // ########## 64bit memory manipulations [
5295              memcpy( (char *)PseudoLinkedPointerAUX_64, &LEAF[0], 8+8+8+2*(LongestLineInclusive+1+4) );
5296                      } // ########## 64bit memory manipulations ]
5297                PseudoLinkedPointerAUX_64 = PseudoLinkedPointerNEW_64 + 8 + 8 + 8;
5298                      if (BSTorBtree == 2) {
5299                fsetpos(fp_outRG, &PseudoLinkedPointerAUX_64);
5300                fwrite(&wrdAUX[0], (LongestLineInclusive+1+4), 1, fp_outRG);
5301                      } else { // ########## 64bit memory manipulations [
5302              memcpy( (char *)PseudoLinkedPointerAUX_64, &wrdAUX[0], (LongestLineInclusive+1+4) );
5303                      } // ########## 64bit memory manipulations ]
5304              // ]  //r.14+
5305            }
5306          if (POffsetInLEAF == 16) // wrd > RW
5307            {
5308 //                     memcpy( wrdUP, PseudoLinkedPointer+4+4+4+wrdlen, wrdlen ); // RW up
5309 //                     *(char *)(PseudoLinkedPointer+4+4+4+wrdlen) = 0;    // RW mark unused in OLD LEAF
5310 //                     memcpy( PseudoLinkedPointerNEW+4+4+4, wrd, wrdlen );   // wrd go to NEW LEAF
5311              // [  //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
5312              //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1+4);
5313              //fsetpos(fp_outRG, &PseudoLinkedPointerAUX_64);
5314              //fread(&wrdUP[0], (LongestLineInclusive+1+4), 1, fp_outRG);
5315               //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1+4);
5316               //fsetpos(fp_outRG, &PseudoLinkedPointerAUX_64);
5317               //fwrite(&OneChar_ieByte, 1, 1, fp_outRG); // Write ZERO ASCII code
5318                //PseudoLinkedPointerAUX_64 = PseudoLinkedPointerNEW_64 + 8 + 8 + 8;
5319                //fsetpos(fp_outRG, &PseudoLinkedPointerAUX_64);
5320                //fwrite(&wrd[0], (LongestLineInclusive+1+4), 1, fp_outRG);
5321              // ]  //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
5322              // [  //r.14+
5323              memcpy( &wrdUP[0], &LEAF[8 + 8 + 8 + (LongestLineInclusive+1+4)], (LongestLineInclusive+1+4) );
5324              memcpy( &LEAF[8 + 8 + 8 + (LongestLineInclusive+1+4)], &OneChar_ieByte, 1 );
5325                      if (BSTorBtree == 2) {
5326              fsetpos(fp_outRG, &PseudoLinkedPointerAUX_64);
```



```
5327              fwrite(&LEAF[0], 8+8+8+2*(LongestLineInclusive+1+4), 1, fp_outRG);
5328                      } else { // ########## 64bit memory manipulations [
5329              memcpy( (char *)PseudoLinkedPointerAUX_64, &LEAF[0], 8+8+8+2*(LongestLineInclusive+1+4) );
5330                      } // ########## 64bit memory manipulations ]
5331              // ]  //r.14+
5332              // [  //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one! Here NO need!
5333                PseudoLinkedPointerAUX_64 = PseudoLinkedPointerNEW_64 + 8 + 8 + 8;
5334                      if (BSTorBtree == 2) {
5335                fsetpos(fp_outRG, &PseudoLinkedPointerAUX_64);
5336                fwrite(&wrd[0], (LongestLineInclusive+1+4), 1, fp_outRG);
5337                      } else { // ########## 64bit memory manipulations [
5338              memcpy( (char *)PseudoLinkedPointerAUX_64, &wrd[0], (LongestLineInclusive+1+4) );
5339                      } // ########## 64bit memory manipulations ]
5340              // ]  //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one! Here NO need!
5341          }
5342      }
5343 else // If LEAF is not full: Case #1
5344      { SplitOccured = 0; WORDcountDistinct++;
5345        if (POffsetInLEAF == 0) // wrd < [LW][] so [LW][] -> [][LW] -> [wrd][LW]
5346          {
5347 //                     memcpy( PseudoLinkedPointer+4+4+4+wrdlen, PseudoLinkedPointer+4+4+4, wrdlen );
5348 //                     memcpy( PseudoLinkedPointer+4+4+4, wrd, wrdlen );
5349              // [  //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
5350               //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8;
5351               //fsetpos(fp_outRG, &PseudoLinkedPointerAUX_64);
5352               //fread(&wrdAUX[0], (LongestLineInclusive+1+4), 1, fp_outRG);
5353               //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1+4);
5354               //fsetpos(fp_outRG, &PseudoLinkedPointerAUX_64);
5355               //fwrite(&wrdAUX[0], (LongestLineInclusive+1+4), 1, fp_outRG);
5356                //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8;
5357                //fsetpos(fp_outRG, &PseudoLinkedPointerAUX_64);
5358                //fwrite(&wrd[0], (LongestLineInclusive+1+4), 1, fp_outRG);
5359              // [  //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
5360              // [  //r.14+
5361              memcpy( &wrdAUX[0], &LEAF[8 + 8 + 8], (LongestLineInclusive+1+4) );
5362              memcpy( &LEAF[8 + 8 + 8 + (LongestLineInclusive+1+4)], &wrdAUX[0], (LongestLineInclusive+1+4) );
5363              memcpy( &LEAF[8 + 8 + 8], &wrd[0], (LongestLineInclusive+1+4) );
5364                      if (BSTorBtree == 2) {
5365              fsetpos(fp_outRG, &PseudoLinkedPointerAUX_64);
5366              fwrite(&LEAF[0], 8+8+8+2*(LongestLineInclusive+1+4), 1, fp_outRG);
5367                      } else { // ########## 64bit memory manipulations [
5368              memcpy( (char *)PseudoLinkedPointerAUX_64, &LEAF[0], 8+8+8+2*(LongestLineInclusive+1+4) );
5369                      } // ########## 64bit memory manipulations ]
5370              // ]  //r.14+
5371
5372          }
5373        if (POffsetInLEAF == 8) // wrd > [LW][] so [LW][] -> [LW][wrd]
5374          {
5375 //                     memcpy( PseudoLinkedPointer+4+4+4+wrdlen, wrd, wrdlen );
5376              // [  //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one! Here NO need!
5377                PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1+4);
5378                      if (BSTorBtree == 2) {
5379                fsetpos(fp_outRG, &PseudoLinkedPointerAUX_64);
5380                fwrite(&wrd[0], (LongestLineInclusive+1+4), 1, fp_outRG);
5381                      } else { // ########## 64bit memory manipulations [
5382              memcpy( (char *)PseudoLinkedPointerAUX_64, &wrd[0], (LongestLineInclusive+1+4) );
5383                      } // ########## 64bit memory manipulations ]
5384              // ]  //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one! Here NO need!
5385          }
5386      }
5387
5388 if (SplitOccured != 0)
5389 {
5390 // ~  Second deal with the INNER NODE(S) i.e Case #3 & Case #4:
5391        while (StackPtr != 0 || SplitOccured != 0)
5392               {
5393          // 'PseudoLinkedPointerNEW' is new LEAF to be inserted
5394          // 'wrdUP' is NEW word to be inserted
5395          if (StackPtr != 0)
5396          {
5397             POffsetInLEAF = BSTstack[--StackPtr];
5398             PseudoLinkedPointer_64 = BSTstack[--StackPtr];
5399 //if ( *(char *)(PseudoLinkedPointer+4+4+4+wrdlen) != 0 ) // If LEAF is full: Case #4
5400              // [  //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
5401              //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1+4); //RW
5402              //fsetpos(fp_outRG, &PseudoLinkedPointerAUX_64);
5403              //fread(&SomeByte, 1, 1, fp_outRG);
5404              // ]  //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
5405              // [  //r.14+
5406              PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64;
5407                      if (BSTorBtree == 2) {
5408              fsetpos(fp_outRG, &PseudoLinkedPointerAUX_64);
5409              fread(&LEAF[0], 8+8+8+2*(LongestLineInclusive+1+4), 1, fp_outRG);
5410                      } else { // ########## 64bit memory manipulations [
5411              memcpy( &LEAF[0], (char *)PseudoLinkedPointerAUX_64, 8+8+8+2*(LongestLineInclusive+1+4) );
5412                      } // ########## 64bit memory manipulations ]
5413              memcpy( &SomeByte, &LEAF[8 + 8 + 8 + (LongestLineInclusive+1+4)], 1 );
5414              // ]  //r.14+
```

```
5415              if (SomeByte != 0 ) // RW exists
5416     { SplitOccured = 1;
5417 //                      memcpy( wrdUPold, wrdUP, wrdlen ); // LW up
5418 //                      PseudoLinkedPointerNEWold = PseudoLinkedPointerNEW;
5419 //                      memcpy( wrdUPold, wrdUP, (LongestLineInclusive+1+4) );
5420 //                      PseudoLinkedPointerNEWold_64 = PseudoLinkedPointerNEW_64;
5421             // ALlocate NEW LEAF:
5422 //            if( (unsigned long)(bufend[LetterOffset] - BufStart) + 2*wrdlen + 4 + 4 + 4 < GRMBLFoolAgain[(int)wrdlen] ) // +4 more for BST instead of LL; + more(see LEAF)
5423 //                      {
5424 //                      memcpy( &PseudoLinkedPointerNEW, &bufend[LetterOffset], 4 );
5425 //                      bufend[LetterOffset] = bufend[LetterOffset] + 4 + 4 + 4; // + 4 due to above commenting
5426 //                      bufend[LetterOffset] = bufend[LetterOffset] + 2*wrdlen;
5427 //                      if (MAXusedBuffer[wrdlen] < (unsigned long)(bufend[LetterOffset] - BufStart)) {MAXusedBuffer[wrdlen] = (unsigned long)(bufend[LetterOffset] - BufStart);}
5428 //                      }
5429             if( 8 + 8 + 8 + 2*(LongestLineInclusive+1+4) < size_in64_L14 - (BufEnd_64-(unsigned long long)pointerflush_64) ) // the longest wrdlen is LongestLineInclusive but actual is LongestLineInclusive(+ CR char)
5430                       {
5431               PseudoLinkedPointerNEW_64 = BufEnd_64;
5432               BufEnd_64 = BufEnd_64 + 8 + 8 + 8;
5433               BufEnd_64 = BufEnd_64 + 2*(LongestLineInclusive+1+4);
5434               // [  //r.14+
5435               //PseudoLinkedPointerAUX_64 = PseudoLinkedPointerNEW_64;
5436               //fsetpos(fp_outRG, &PseudoLinkedPointerAUX_64);
5437               //fread(&LEAFNEW[0], 8+8+8+2*(LongestLineInclusive+1+4), 1, fp_outRG);
5438               // In fact above three lines are slow, the only need is ZEROed LEAFNEW.
5439               memset(&LEAFNEW[0],0,8+8+8+2*(LongestLineInclusive+1+4));
5440               // ]  //r.14+
5441                         }
5442             else
5443             { printf( "\nLeprechaun: Failure! Increment 'Memory for each Letter' parameter(third one)!\n" );
5444       fprintf( fp_outLOG, "Word count: %s of them %s distinct\n", _ui64toaKAZEcomma(wORDcount, llTOaDigits, 10), _ui64toaKAZEcomma((unsigned long long)WORDcountDistinct, llTOaDigits2, 10) );
5445       fprintf( fp_outLOG, "Allocated memory: %s bytes\n", _ui64toaKAZEcomma(size_in64_L14, llTOaDigits, 10) );
5446       fprintf( fp_outLOG, "Total Attempts to Find/Put WORDs into B-trees order 3: %s\n", _ui64toaKAZEcomma(WORDcountAttemptsToPut, llTOaDigits, 10) );
5447       fprintf( fp_outLOG, "Used value for third parameter in KB: %lu\n", (unsigned long)Thunderwith );
5448       fprintf( fp_outLOG, "Leprechaun: Failure! Increment 'Memory for each Letter' parameter(third one)!\n\n");
5449               return( 1 );
5450             }
5451       if (POffsetInLEAF == 0) // wrdUPold < LW
5452          {
5453 //                      memcpy( wrdUP, PseudoLinkedPointer+4+4+4, wrdlen ); // LW up
5454 //                      memcpy( PseudoLinkedPointer+4+4+4, wrdUPold, wrdlen );   // wrdUPold go to OLD LEAF
5455 //                      memcpy( PseudoLinkedPointerNEW+4+4+4, PseudoLinkedPointer+4+4+4+wrdlen, wrdlen ); // RW go to NEW LEAF
5456 //                      *(char *)(PseudoLinkedPointer+4+4+4+wrdlen) = 0;    // RW mark unused in OLD LEAF
5457 //                      // [LP](PseudoLinkedPointerNEWold)[MP][RP](wrdUPold)[LW][RW]       ------
5458 //                      //                pair [LW] PseudoLinkedPointerNEW goes up              |
5459 //                      // PseudoLinkedPointer:                          PseudoLinkedPointerNEW: |
5460 //                      // [LP](PseudoLinkedPointerNEWold)[](wrdUPold)  [MP][RP][][RW]      <----
5461 //                      // no need to put zero in RP because logic is based on words existence:
5462 //                      memcpy( PseudoLinkedPointerNEW+0, PseudoLinkedPointer+4, 4 );
5463 //                      memcpy( PseudoLinkedPointerNEW+4, PseudoLinkedPointer+8, 4 );
5464 //                      memcpy( PseudoLinkedPointer+4, &PseudoLinkedPointerNEWold, 4 );
5465               // [  //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
5466               //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8;
5467               //fsetpos(fp_outRG, &PseudoLinkedPointerAUX_64);
5468               //fread(&wrdUP[0], (LongestLineInclusive+1+4), 1, fp_outRG);
5469                //fsetpos(fp_outRG, &PseudoLinkedPointerAUX_64);
5470                //fwrite(&wrdUPold[0], (LongestLineInclusive+1+4), 1, fp_outRG);
5471                 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1+4);
5472                 //fsetpos(fp_outRG, &PseudoLinkedPointerAUX_64);
5473                 //fread(&wrdAUX[0], (LongestLineInclusive+1+4), 1, fp_outRG);
5474                 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointerNEW_64 + 8 + 8 + 8;
5475                 //fsetpos(fp_outRG, &PseudoLinkedPointerAUX_64);
5476                 //fwrite(&wrdAUX[0], (LongestLineInclusive+1+4), 1, fp_outRG);
5477                  //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1+4);
5478                  //fsetpos(fp_outRG, &PseudoLinkedPointerAUX_64);
5479                  //fwrite(&OneChar_ieByte, 1, 1, fp_outRG); // Write ZERO ASCII code
5480               //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8;
5481               //fsetpos(fp_outRG, &PseudoLinkedPointerAUX_64);
5482               //fread(&PseudoLinkedPointerAUXdumbo_64, 8, 1, fp_outRG);
5483               //PseudoLinkedPointerAUX_64 = PseudoLinkedPointerNEW_64 + 0;
5484               //fsetpos(fp_outRG, &PseudoLinkedPointerAUX_64);
5485               //fwrite(&PseudoLinkedPointerAUXdumbo_64, 8, 1, fp_outRG);
5486                //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 16;
5487                //fsetpos(fp_outRG, &PseudoLinkedPointerAUX_64);
5488                //fread(&PseudoLinkedPointerAUXdumbo_64, 8, 1, fp_outRG);
5489                //PseudoLinkedPointerAUX_64 = PseudoLinkedPointerNEW_64 + 8;
5490                //fsetpos(fp_outRG, &PseudoLinkedPointerAUX_64);
5491                //fwrite(&PseudoLinkedPointerAUXdumbo_64, 8, 1, fp_outRG);
5492                 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8;
5493                 //fsetpos(fp_outRG, &PseudoLinkedPointerAUX_64);
5494                 //fwrite(&PseudoLinkedPointerNEWold_64, 8, 1, fp_outRG);
5495               // ]  //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
5496               // [  //r.14+
5497               memcpy( &wrdUP[0], &LEAF[8 + 8 + 8], (LongestLineInclusive+1+4) );
```



```
5498               memcpy( &LEAF[8 + 8 + 8], &wrdUPold[0], (LongestLineInclusive+1+4) );
5499               memcpy( &wrdAUX[0], &LEAF[8 + 8 + 8 + (LongestLineInclusive+1+4)], (LongestLineInclusive+1+4) );
5500               memcpy( &LEAFNEW[8 + 8 + 8], &wrdAUX[0], (LongestLineInclusive+1+4) );
5501               memcpy( &LEAF[8 + 8 + 8 + (LongestLineInclusive+1+4)], &OneChar_ieByte, 1 );
5502               memcpy( &PseudoLinkedPointerAUXdumbo_64, &LEAF[8], 8 );
5503               memcpy( &LEAFNEW[0], &PseudoLinkedPointerAUXdumbo_64, 8 );
5504               memcpy( &PseudoLinkedPointerAUXdumbo_64, &LEAF[16], 8 );
5505               memcpy( &LEAFNEW[8], &PseudoLinkedPointerAUXdumbo_64, 8 );
5506               memcpy( &LEAF[8], &PseudoLinkedPointerNEWold_64, 8 );
5507               PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64;
5508                         if (BSTorBtree == 2) {
5509               fsetpos(fp_outRG, &PseudoLinkedPointerAUX_64);
5510               fwrite(&LEAF[0], 8+8+8+2*(LongestLineInclusive+1+4), 1, fp_outRG);
5511                         } else { // ########## 64bit memory manipulations [
5512               memcpy( (char *)PseudoLinkedPointerAUX_64, &LEAF[0], 8+8+8+2*(LongestLineInclusive+1+4) );
5513                         } // ########## 64bit memory manipulations ]
5514               PseudoLinkedPointerAUX_64 = PseudoLinkedPointerNEW_64;
5515                         if (BSTorBtree == 2) {
5516               fsetpos(fp_outRG, &PseudoLinkedPointerAUX_64);
5517               fwrite(&LEAFNEW[0], 8+8+8+2*(LongestLineInclusive+1+4), 1, fp_outRG);
5518                         } else { // ########## 64bit memory manipulations [
5519               memcpy( (char *)PseudoLinkedPointerAUX_64, &LEAFNEW[0], 8+8+8+2*(LongestLineInclusive+1+4) );
5520                         } // ########## 64bit memory manipulations ]
5521               // ]  //r.14+
5522          }
5523       if (POffsetInLEAF == 8) // LW < wrdUPold < RW
5524          {
5525 //                      memcpy( wrdUP, wrdUPold, wrdlen );                      // wrdUPold up
5526 //                      memcpy( PseudoLinkedPointerNEW+4+4+4, PseudoLinkedPointer+4+4+4+wrdlen, wrdlen ); // RW go to NEW LEAF
5527 //                      *(char *)(PseudoLinkedPointer+4+4+4+wrdlen) = 0;    // RW mark unused in OLD LEAF
5528 //                      // [LP][MP](PseudoLinkedPointerNEWold)[RP][LW](wrdUPold)[RW]       ------
5529 //                      //                pair [wrdUPold] PseudoLinkedPointerNEW goes up          |
5530 //                      // PseudoLinkedPointer:  PseudoLinkedPointerNEW:                          |
5531 //                      // [LP][MP][][LW]        (PseudoLinkedPointerNEWold)[RP][][RW]      <----
5532 //                      // no need to put zero in RP because logic is based on words existence:
5533 //                      memcpy( PseudoLinkedPointerNEW+0, &PseudoLinkedPointerNEWold, 4 );
5534 //                      memcpy( PseudoLinkedPointerNEW+4, PseudoLinkedPointer+8, 4 );
5535               // [  //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
5536                         memcpy( wrdUP, wrdUPold, (LongestLineInclusive+1+4) );
5537                 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1+4);
5538                 //fsetpos(fp_outRG, &PseudoLinkedPointerAUX_64);
5539                 //fread(&wrdAUX[0], (LongestLineInclusive+1+4), 1, fp_outRG);
5540                 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointerNEW_64 + 8 + 8 + 8;
5541                 //fsetpos(fp_outRG, &PseudoLinkedPointerAUX_64);
5542                 //fwrite(&wrdAUX[0], (LongestLineInclusive+1+4), 1, fp_outRG);
5543                  //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1+4);
5544                  //fsetpos(fp_outRG, &PseudoLinkedPointerAUX_64);
5545                  //fwrite(&OneChar_ieByte, 1, 1, fp_outRG); // Write ZERO ASCII code
5546               //PseudoLinkedPointerAUX_64 = PseudoLinkedPointerNEW_64 + 0;
5547               //fsetpos(fp_outRG, &PseudoLinkedPointerAUX_64);
5548               //fwrite(&PseudoLinkedPointerNEWold_64, 8, 1, fp_outRG);
5549                //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 16;
5550                //fsetpos(fp_outRG, &PseudoLinkedPointerAUX_64);
5551                //fread(&PseudoLinkedPointerAUXdumbo_64, 8, 1, fp_outRG);
5552                //PseudoLinkedPointerAUX_64 = PseudoLinkedPointerNEW_64 + 8;
5553                //fsetpos(fp_outRG, &PseudoLinkedPointerAUX_64);
5554                //fwrite(&PseudoLinkedPointerAUXdumbo_64, 8, 1, fp_outRG);
5555               // ]  //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
5556               // [  //r.14+
5557               memcpy( &wrdAUX[0], &LEAF[8 + 8 + 8 + (LongestLineInclusive+1+4)], (LongestLineInclusive+1+4) );
5558               memcpy( &LEAFNEW[8 + 8 + 8], &wrdAUX[0], (LongestLineInclusive+1+4) );
5559               memcpy( &LEAF[8 + 8 + 8 + (LongestLineInclusive+1+4)], &OneChar_ieByte, 1 );
5560               memcpy( &LEAFNEW[0], &PseudoLinkedPointerNEWold_64, 8 );
5561               memcpy( &PseudoLinkedPointerAUXdumbo_64, &LEAF[16], 8 );
5562               memcpy( &LEAFNEW[8], &PseudoLinkedPointerAUXdumbo_64, 8 );
5563               PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64;
5564                         if (BSTorBtree == 2) {
5565               fsetpos(fp_outRG, &PseudoLinkedPointerAUX_64);
5566               fwrite(&LEAF[0], 8+8+8+2*(LongestLineInclusive+1+4), 1, fp_outRG);
5567                         } else { // ########## 64bit memory manipulations [
5568               memcpy( (char *)PseudoLinkedPointerAUX_64, &LEAF[0], 8+8+8+2*(LongestLineInclusive+1+4) );
5569                         } // ########## 64bit memory manipulations ]
5570               PseudoLinkedPointerAUX_64 = PseudoLinkedPointerNEW_64;
5571                         if (BSTorBtree == 2) {
5572               fsetpos(fp_outRG, &PseudoLinkedPointerAUX_64);
5573               fwrite(&LEAFNEW[0], 8+8+8+2*(LongestLineInclusive+1+4), 1, fp_outRG);
5574                         } else { // ########## 64bit memory manipulations [
5575               memcpy( (char *)PseudoLinkedPointerAUX_64, &LEAFNEW[0], 8+8+8+2*(LongestLineInclusive+1+4) );
5576                         } // ########## 64bit memory manipulations ]
5577               // ]  //r.14+
5578          }
5579       if (POffsetInLEAF == 16) // wrdUPold > RW
5580          {
5581 //                      memcpy( wrdUP, PseudoLinkedPointer+4+4+4+wrdlen, wrdlen ); // RW up
5582 //                      *(char *)(PseudoLinkedPointer+4+4+4+wrdlen) = 0;    // RW mark unused in OLD LEAF
5583 //                      memcpy( PseudoLinkedPointerNEW+4+4+4, wrdUPold, wrdlen );   // wrdUPold go to NEW LEAF
5584 //                      // [LP][MP][RP](PseudoLinkedPointerNEWold)[LW][RW](wrdUPold)       ------
5585 //                      //                pair [RW] PseudoLinkedPointerNEW goes up              |
```

```
5586 //                      // PseudoLinkedPointer:  PseudoLinkedPointerNEW:                        |
5587 //                      // [LP][MP][][LW]        [RP](PseudoLinkedPointerNEWold)[](wrdUPold) <---
5588 //                      // no need to put zero in RP because logic is based on words existence:
5589 //                      memcpy( PseudoLinkedPointerNEW+0, PseudoLinkedPointer+8, 4 );
5590 //                      memcpy( PseudoLinkedPointerNEW+4, &PseudoLinkedPointerNEWold, 4 );
5591                 // [  //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
5592                 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1+4);
5593                 //fsetpos(fp_outRG, &PseudoLinkedPointerAUX_64);
5594                 //fread(&wrdUP[0], (LongestLineInclusive+1+4), 1, fp_outRG);
5595                  //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1+4);
5596                  //fsetpos(fp_outRG, &PseudoLinkedPointerAUX_64);
5597                  //fwrite(&OneChar_ieByte, 1, 1, fp_outRG); // Write ZERO ASCII code
5598                   //PseudoLinkedPointerAUX_64 = PseudoLinkedPointerNEW_64 + 8 + 8 + 8;
5599                   //fsetpos(fp_outRG, &PseudoLinkedPointerAUX_64);
5600                   //fwrite(&wrdUPold[0], (LongestLineInclusive+1+4), 1, fp_outRG);
5601                 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 16;
5602                 //fsetpos(fp_outRG, &PseudoLinkedPointerAUX_64);
5603                 //fread(&PseudoLinkedPointerAUXdumbo_64, 8, 1, fp_outRG);
5604                 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointerNEW_64 + 0;
5605                 //fsetpos(fp_outRG, &PseudoLinkedPointerAUX_64);
5606                 //fwrite(&PseudoLinkedPointerAUXdumbo_64, 8, 1, fp_outRG);
5607                  //PseudoLinkedPointerAUX_64 = PseudoLinkedPointerNEW_64 + 8;
5608                  //fsetpos(fp_outRG, &PseudoLinkedPointerAUX_64);
5609                  //fwrite(&PseudoLinkedPointerNEWold_64, 8, 1, fp_outRG);
5610                 // ]  //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
5611                 // [  //r.14+
5612                 memcpy( &wrdUP[0], &LEAF[8 + 8 + 8 + (LongestLineInclusive+1+4)], (LongestLineInclusive+1+4) );
5613                 memcpy( &LEAF[8 + 8 + 8 + (LongestLineInclusive+1+4)], &OneChar_ieByte, 1 );
5614                 memcpy( &LEAFNEW[8 + 8 + 8], &wrdUPold[0], (LongestLineInclusive+1+4) );
5615                 memcpy( &PseudoLinkedPointerAUXdumbo_64, &LEAF[16], 8 );
5616                 memcpy( &LEAFNEW[0], &PseudoLinkedPointerAUXdumbo_64, 8 );
5617                 memcpy( &LEAFNEW[8], &PseudoLinkedPointerNEWold_64, 8 );
5618                 PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64;
5619                         if (BSTorBtree == 2) {
5620                 fsetpos(fp_outRG, &PseudoLinkedPointerAUX_64);
5621                 fwrite(&LEAF[0], 8+8+8+2*(LongestLineInclusive+1+4), 1, fp_outRG);
5622                         } else { // ########## 64bit memory manipulations [
5623                 memcpy( (char *)PseudoLinkedPointerAUX_64, &LEAF[0], 8+8+8+2*(LongestLineInclusive+1+4) );
5624                         } // ########## 64bit memory manipulations ]
5625                 PseudoLinkedPointerAUX_64 = PseudoLinkedPointerNEW_64;
5626                         if (BSTorBtree == 2) {
5627                 fsetpos(fp_outRG, &PseudoLinkedPointerAUX_64);
5628                 fwrite(&LEAFNEW[0], 8+8+8+2*(LongestLineInclusive+1+4), 1, fp_outRG);
5629                         } else { // ########## 64bit memory manipulations [
5630                 memcpy( (char *)PseudoLinkedPointerAUX_64, &LEAFNEW[0], 8+8+8+2*(LongestLineInclusive+1+4) );
5631                         } // ########## 64bit memory manipulations ]
5632                 // ]  //r.14+
5633         }
5634     }
5635 else // If LEAF is not full: Case #3
5636     { SplitOccured = 0;
5637       if (POffsetInLEAF == 0) // wrdUP < [LW][] so [LW][] -> [][LW] -> [wrdUP][LW]
5638         {
5639 //                      memcpy( PseudoLinkedPointer+4+4+4+wrdlen, PseudoLinkedPointer+4+4+4, wrdlen );
5640 //                      memcpy( PseudoLinkedPointer+4+4+4, wrdUP, wrdlen );
5641 //                              // [LP][MP][] -> [LP][][MP] -> [LP][np][MP]
5642 //                      memcpy( PseudoLinkedPointer+8, PseudoLinkedPointer+4, 4 );
5643 //                      memcpy( PseudoLinkedPointer+4, &PseudoLinkedPointerNEW, 4 );
5644                 // [  //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
5645                 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8;
5646                 //fsetpos(fp_outRG, &PseudoLinkedPointerAUX_64);
5647                 //fread(&wrdAUX[0], (LongestLineInclusive+1+4), 1, fp_outRG);
5648                 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1+4);
5649                 //fsetpos(fp_outRG, &PseudoLinkedPointerAUX_64);
5650                 //fwrite(&wrdAUX[0], (LongestLineInclusive+1+4), 1, fp_outRG);
5651                  //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8;
5652                  //fsetpos(fp_outRG, &PseudoLinkedPointerAUX_64);
5653                  //fwrite(&wrdUP[0], (LongestLineInclusive+1+4), 1, fp_outRG);
5654                 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8;
5655                 //fsetpos(fp_outRG, &PseudoLinkedPointerAUX_64);
5656                 //fread(&PseudoLinkedPointerAUXdumbo_64, 8, 1, fp_outRG);
5657                 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 16;
5658                 //fsetpos(fp_outRG, &PseudoLinkedPointerAUX_64);
5659                 //fwrite(&PseudoLinkedPointerAUXdumbo_64, 8, 1, fp_outRG);
5660                  //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8;
5661                  //fsetpos(fp_outRG, &PseudoLinkedPointerAUX_64);
5662                  //fwrite(&PseudoLinkedPointerNEW_64, 8, 1, fp_outRG);
5663                 // ]  //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
5664                 // [  //r.14+
5665                 memcpy( &wrdAUX[0], &LEAF[8 + 8 + 8], (LongestLineInclusive+1+4) );
5666                 memcpy( &LEAF[8 + 8 + 8 + (LongestLineInclusive+1+4)], &wrdAUX[0], (LongestLineInclusive+1+4) );
5667                 memcpy( &LEAF[8 + 8 + 8], &wrdUP[0], (LongestLineInclusive+1+4) );
5668                 memcpy( &PseudoLinkedPointerAUXdumbo_64, &LEAF[8], 8 );
5669                 memcpy( &LEAF[8 + 8], &PseudoLinkedPointerAUXdumbo_64, 8 );
5670                 memcpy( &LEAF[8], &PseudoLinkedPointerNEW_64, 8 );
5671                         if (BSTorBtree == 2) {
5672                 fsetpos(fp_outRG, &PseudoLinkedPointerAUX_64);
5673                 fwrite(&LEAF[0], 8+8+8+2*(LongestLineInclusive+1+4), 1, fp_outRG);
```



```
5674                         } else { // ########## 64bit memory manipulations [
5675                 memcpy( (char *)PseudoLinkedPointerAUX_64, &LEAF[0], 8+8+8+2*(LongestLineInclusive+1+4) );
5676                         } // ########## 64bit memory manipulations ]
5677                 // ]  //r.14+
5678         }
5679       if (POffsetInLEAF == 8) // wrdUP > [LW][] so [LW][] -> [LW][wrdUP]
5680         {
5681 //                      memcpy( PseudoLinkedPointer+4+4+4+wrdlen, wrdUP, wrdlen );
5682 //                              // [LP][MP][] -> [LP][MP][np]
5683 //                      memcpy( PseudoLinkedPointer+8, &PseudoLinkedPointerNEW, 4 );
5684                 // [  //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
5685                 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1+4);
5686                 //fsetpos(fp_outRG, &PseudoLinkedPointerAUX_64);
5687                 //fwrite(&wrdUP[0], (LongestLineInclusive+1+4), 1, fp_outRG);
5688                  //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 16;
5689                  //fsetpos(fp_outRG, &PseudoLinkedPointerAUX_64);
5690                  //fwrite(&PseudoLinkedPointerNEW_64, 8, 1, fp_outRG);
5691                 // ]  //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
5692                 // [  //r.14+
5693                 memcpy( &LEAF[8 + 8 + 8 + (LongestLineInclusive+1+4)], &wrdUP[0], (LongestLineInclusive+1+4) );
5694                 memcpy( &LEAF[8 + 8], &PseudoLinkedPointerNEW_64, 8 );
5695                         if (BSTorBtree == 2) {
5696                 fsetpos(fp_outRG, &PseudoLinkedPointerAUX_64);
5697                 fwrite(&LEAF[0], 8+8+8+2*(LongestLineInclusive+1+4), 1, fp_outRG);
5698                         } else { // ########## 64bit memory manipulations [
5699                 memcpy( (char *)PseudoLinkedPointerAUX_64, &LEAF[0], 8+8+8+2*(LongestLineInclusive+1+4) );
5700                         } // ########## 64bit memory manipulations ]
5701                 // ]  //r.14+
5702         }
5703           break;
5704     }
5705         }
5706         else // Empty stack means ROOT and more over ROOT is already splitted(Case #4 is off)
5707         {
5708      // If LEAF is not full: Case #3
5709      // THIS IS WHERE A NEW(SECOND) LEAF 'PseudoLinkedPointerROOT' must be allocated:
5710 //           if( (unsigned long)(bufend[LetterOffset] - BufStart) + 2*wrdlen + 4 + 4 + 4 < GRMBLFoolAgain[(int)wrdlen] ) // +4 more for BST instead of LL; + more(see LEAF)
5711 //                   {
5712 //                   memcpy( &PseudoLinkedPointerROOT, &bufend[LetterOffset], 4 );
5713 //                   bufend[LetterOffset] = bufend[LetterOffset] + 4 + 4 + 4; // + 4 due to above commenting
5714 //                   memcpy( bufend[LetterOffset], wrdUP, wrdlen );
5715 //                   bufend[LetterOffset] = bufend[LetterOffset] + 2*wrdlen;
5716 //                   if (MAXusedBuffer[wrdlen] < (unsigned long)(bufend[LetterOffset] - BufStart)) {MAXusedBuffer[wrdlen] = (unsigned long)(bufend[LetterOffset] - BufStart);}
5717 //                   }
5718           if( 8 + 8 + 8 + 2*(LongestLineInclusive+1+4) < size_in64_L14 - (BufEnd_64-(unsigned long long)pointerflush_64) ) // the longest wrdlen is LongestLineInclusive but actual is LongestLineInclusive(+ CR char)
5719                   {
5720             PseudoLinkedPointerROOT_64 = BufEnd_64;
5721             BufEnd_64 = BufEnd_64 + 8 + 8 + 8;
5722             BufEnd_64 = BufEnd_64 + 2*(LongestLineInclusive+1+4);
5723             PseudoLinkedPointerAUX_64 = PseudoLinkedPointerROOT_64 + 8 + 8 + 8;
5724                     if (BSTorBtree == 2) {
5725             fsetpos(fp_outRG, &PseudoLinkedPointerAUX_64);
5726             fwrite(&wrdUP[0], (LongestLineInclusive+1+4), 1, fp_outRG);
5727                     } else { // ########## 64bit memory manipulations [
5728             memcpy( (char *)PseudoLinkedPointerAUX_64, &wrdUP[0], (LongestLineInclusive+1+4) );
5729                     } // ########## 64bit memory manipulations ]
5730                   }
5731           else
5732           { printf( "\nLeprechaun: Failure! Increment 'Memory for each Letter' parameter(third one)!\n" );
5733       fprintf( fp_outLOG, "Word count: %s of them %s distinct\n", _ui64toaKAZEcomma(WORDcount, llTOaDigits, 10), _ui64toaKAZEcomma((unsigned long long)WORDcountDistinct, llTOaDigits2, 10) );
5734       fprintf( fp_outLOG, "Allocated memory: %s bytes\n", _ui64toaKAZEcomma(size_in64_L14, llTOaDigits, 10) );
5735       fprintf( fp_outLOG, "Total Attempts to Find/Put WORDs into B-trees order 3: %s\n", _ui64toaKAZEcomma(WORDcountAttemptsToPut, llTOaDigits, 10) );
5736       fprintf( fp_outLOG, "Used value for third parameter in KB: %lu\n", (unsigned long)Thunderwith );
5737       fprintf( fp_outLOG, "Leprechaun: Failure! Increment 'Memory for each Letter' parameter(third one)!\n\n");
5738             return( 1 );
5739           }
5740           // Here                    --  'PseudoLinkedPointerROOT' --
5741           //                         |          (wrdUP)          |
5742           // 'PseudoLinkedPointer' <--                           --> 'PseudoLinkedPointerNEW'
5743           //          (LW)                                              (RW)
5744 //          memcpy( PseudoLinkedPointerROOT, &PseudoLinkedPointer, 4 );      // LP
5745 //          memcpy( PseudoLinkedPointerROOT+4, &PseudoLinkedPointerNEW, 4 ); // MP
5746 //          // Here must NEW ROOT be updated i.e. HASH table(SLOT) must point it:
5747 //          memcpy( BufStart+Slot, &PseudoLinkedPointerROOT, 4 );
5748       PseudoLinkedPointerAUX_64 = PseudoLinkedPointerROOT_64;
5749                   if (BSTorBtree == 2) {
5750       fsetpos(fp_outRG, &PseudoLinkedPointerAUX_64);
5751       fwrite(&PseudoLinkedPointer_64, 8, 1, fp_outRG);
5752                   } else { // ########## 64bit memory manipulations [
5753             memcpy( (char *)PseudoLinkedPointerAUX_64, &PseudoLinkedPointer_64, 8 );
5754                   } // ########## 64bit memory manipulations ]
5755        PseudoLinkedPointerAUX_64 = PseudoLinkedPointerROOT_64 + 8;
5756                   if (BSTorBtree == 2) {
```

```
        fsetpos(fp_outRG, &PseudoLinkedPointerAUX_64);
        fwrite(&PseudoLinkedPointerNEW_64, 8, 1, fp_outRG);
                        } else { // ########## 64bit memory manipulations [
              memcpy( (char *)PseudoLinkedPointerAUX_64, &PseudoLinkedPointerNEW_64, 8 );
                        } // ########## 64bit memory manipulations ]
    memcpy( BufStart+Slot, &PseudoLinkedPointerROOT_64, 8 );
          break; //because it is ROOT without split
        }
              } // while
} //if (SplitOccured != 0)
// 3] Insert Iterative ]
} //if (FoundInLinkedList == 0)
} //if (DoNotInsertFlag == 0) { // This line comes since r15FIXFIX+
} //if ( REUSE != 2 ) { //  REUSE [ // This line comes since r16
// ########################################## B-tree order 3 64bit ]
} //if ( (Slot>>HashChunkSizeInBITS) == RipPasses ) {

// External Btrees ]
} //if (BSTorBtree != 2) {
                        } // if ( ( PLE_words == 4 ) && ( wrdlen <= 31 ) ) {
                        } // if( 1 <= wrdlen && wrdlen <= 31 )
/*
The Most Stupid Bug I have ever made [
              else {
              PLE_words_INITflag = 1; // This line fixes the stupidity done since first quadrupleton r1, namely to initialize the sequence
 when a word longer than 31 is spotted - it is wrong to slide it.
              }
The Most Stupid Bug I have ever made ]
*/
//r.15fix [
//This line was intended BUT placed before the fragment 'if( 1 <= wrdlen && wrdlen <= 31 )' [
//                      if( wrdlen > 31 ) PLE_words_INITflag = 1; // Not when wrdlen == 0 as the above buggy fragment!
//This line was intended BUT placed before the fragment 'if( 1 <= wrdlen && wrdlen <= 31 )' ]
//r.15fix ]
if ( PLE_words_INITflag == 1 ) { PLE_words = 0; PLE_words_INITflag = 0; } // Quadruple!
              wrdlen = 0;
                        // This fragment is MIRRORed: #1 copy ]
                }
                //else if( workbyte >= 'A' &&  workbyte <= 'Z' )
                else if( workbyte <= 'Z' )
    {
                        if( wrdlen < 31 )
                        //if( wrdlen < LongestLineInclusive )
                        { wrd[ wrdlen ] = workbyte + 32 ; }
                        wrdlen++;
    }
                else if( workbyte >= 'a' &&  workbyte <= 'z' )
    {
                        if( wrdlen < 31 )
                        //if( wrdlen < LongestLineInclusive )
                        { wrd[ wrdlen ] = workbyte; }
                        wrdlen++;
    }
    else
                {
                        // This fragment is MIRRORed: #2 copy [
                goto ElStupido;
                        // This fragment is MIRRORed: #2 copy ]
    }
        } // i 'for'
        //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
//++Melnitchka;
//Melnitchka = Melnitchka % 4;
//if (Melnitchka == 0){ printf( "|; Word count: %s of them %s distinct; Done: %lu/64\n", _ui64toaKAZEcomma(WORDcount, llTOaDigits, 10),
 _ui64toaKAZEcomma((unsigned long long)WORDcountDistinct, llTOaDigits2, 10), 64 ); }
//if (Melnitchka == 1){ printf( "/; Word count: %s of them %s distinct; Done: %lu/64\n", _ui64toaKAZEcomma(WORDcount, llTOaDigits, 10),
 _ui64toaKAZEcomma((unsigned long long)WORDcountDistinct, llTOaDigits2, 10), 64 ); }
//if (Melnitchka == 2){ printf( "-; Word count: %s of them %s distinct; Done: %lu/64\n", _ui64toaKAZEcomma(WORDcount, llTOaDigits, 10),
 _ui64toaKAZEcomma((unsigned long long)WORDcountDistinct, llTOaDigits2, 10), 64 ); }
//if (Melnitchka == 3){ printf( "\\; Word count: %s of them %s distinct; Done: %lu/64\n", _ui64toaKAZEcomma(WORDcount, llTOaDigits, 10),
 _ui64toaKAZEcomma((unsigned long long)WORDcountDistinct, llTOaDigits2, 10), 64 ); }
Melnitchka = Melnitchka & 3; // 0 1 2 3: 00 01 10 11
                        (void) time(&t4);
                        if (t4 <= t1) {t4 = t1; t4++;}
printf( "%s; %sP/s; Phrase count: %s of them %s distinct; Done: %lu/64\n", Auberge[Melnitchka++], _ui64toaKAZEzerocomma(WORDcount/((int) t4-
 t1), llTOaDigits3, 10)+(26-10), _ui64toaKAZEcomma(WORDcount, llTOaDigits, 10), _ui64toaKAZEcomma((unsigned long long)WORDcountDistinct,
 llTOaDigits2, 10), 64 );

fclose( fp_inLINE );
} // ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ IT IS a FILENAME not a METACOMMAND ]
        LINE10len = 0;
LINE10[ LINE10len ] = 0;
                  }
                }
        } // k 'for'

(void) time(&t3);
if (t3 <= t1) {t3 = t1; t3++;}
```



```
printf( "Bytes per second performance: %sB/s\n", _ui64toaKAZEcomma(FilesLEN/((int) t3-t1), llTOaDigits, 10) );  // Rev. 12+
printf( "Phrases per second performance: %sP/s\n", _ui64toaKAZEcomma(WORDcount/((int) t3-t1), llTOaDigits, 10) );  // Rev. 12+
printf("Time for putting phrases into trees: %d second(s)\n", (int) t3-t1);

if (BSTorBtree < 2) {
        // FLUSH FLUSH FLUSH FLUSH FLUSH FLUSH FLUSH FLUSH FLUSH FLUSH FLUSH
printf("Flushing unsorted words ...\n");
if( ( fp_out = fopen( argv[2], "wb+" ) ) == NULL )
{ printf( "Leprechaun: Can't create file %s \n", argv[2] ); return( 1 ); }
ZEROS[0] = 0; ZEROS[1] = 0; ZEROS[2] = 0; ZEROS[3] = 0;
CRdLFa[0] = 13; CRdLFa[1] = 10;

for( i = 0; i < 806; i++ )
{ //BufStart = pointerflush + i * LetterBuffer; // OLD
  BufStart = pointerflush + (i / 31) * WHOLEletter_BufferSize + OffsetsInBuffer[i % 31];
//  for( j = 0; j < NumberOfSLOTs; j++ )
//  {
//                    Slot = j<<2;
//                    memcpy( &PseudoLinkedPointer, BufStart+Slot, 4 );
//                    while (PseudoLinkedPointer != 0)
//                    { memcpy( &PseudoLinkedPointerNEW, PseudoLinkedPointer, 4 );
//                      memcpy( PseudoLinkedPointer, ZEROS, 4 );
//                      PseudoLinkedPointer = PseudoLinkedPointerNEW;
//                    }
//  }
//  // Start of COUPLES [OFFSET: 4byte(ZEROS)][WORD:up to 31bytes]
//  //fwrite(BufStart+(NumberOfSLOTs+1)*4, bufend[i] - (BufStart+(NumberOfSLOTs+1)*4), 1, fp_out );
//  /* Follows STATE OF UGLINESS: */
//  Flushing = BufStart+(NumberOfSLOTs+1)*4 + 4; // '+ 4' in order to skip first 4 zeros
//  //in case of current buffer not have been used then NOT entering in this cycle
//  while(Flushing < bufend[i])
//  { if (*Flushing != 0) {fwrite(Flushing, 1, 1, fp_out ); TotalWLchars++;}
//    // Below 'Flushing-1' works due to skipped first 4 zeros!
//    if (*(Flushing-1) != 0 && *Flushing == 0) {fwrite(CRdLFa, 2, 1, fp_out);}
//    //last word must be suffixed with 1310 too
//    if (Flushing == bufend[i]-1) {fwrite(CRdLFa, 2, 1, fp_out );}
//    Flushing++;
//  }

  for( j = 0; j < NumberOfSLOTs; j++ )
  {
                  Slot = j<<2;
                  memcpy( &PseudoLinkedPointer, BufStart+Slot, 4 );
                  if (PseudoLinkedPointer != 0)
                  {
NumberOfTrees++;
if (BSTorBtree == 0)
{
// ========================================================== BST traverse [
    // DONE JOB:
    // Must be written BST traverse ! with simulated stack i.e. non-recursive.
    // ...
                                  // /*
                                  // Given a binary search tree, print out
                                  // its data elements in increasing
                                  // sorted order.
                                  // */
                                  // void printTree(struct node* node) {
                                  // if (node == NULL) return;
                                  // printTree(node->left);
                                  // printf("%d ", node->data);
                                  // printTree(node->right);
                                  // }

    // FUTURE JOB:
    // I need functions:
    //                BST_LeafNumber() // greater the better
    //                BST_NodeNumber() // 'BSTcurrent' below
    //                BST_Peak()      // i.e. levels, root has height = 1
    //                BST_PeakIB()    // IBBST(Ideal Balanced BST) has 1 + lgNodeNumber height
    // I need 'Ideal Balancing BST FRAGMENT' with simulated stack:
    // I need 'Ideal Balancing BST FRAGMENT' to be executed when Peak() >= PeakIB()<<1:

// --------------------------------------------------------------------- [
  BSTcurrentNode = 0; BSTcurrentPeak = 0; BSTcurrentLeaf = 0;
                  BSTcurrentPeakMAX = 0; // Height of current BST
  StackPtr = 0;
  while ( 2==2 ) {
      while (PseudoLinkedPointer != 0)
            {
              if (StackPtr > 8192*3-1-3) { printf( "\nLeprechaun: Failure! BST simulated stack overflow, too high BST!\n" ); return( 13 );}
              memcpy( &PseudoLinkedPointerNEWleft, PseudoLinkedPointer, 4 );
              PseudoLinkedPointer = PseudoLinkedPointer + 4;
              memcpy( &PseudoLinkedPointerNEWright, PseudoLinkedPointer, 4 );
              BSTstack[StackPtr] = PseudoLinkedPointer + 4; ++StackPtr; //ptr to wrd
              BSTstack[StackPtr] = PseudoLinkedPointerNEWright; ++StackPtr;
              BSTstack[StackPtr] = PseudoLinkedPointerNEWleft; ++StackPtr; //needed for stats not for recursion
      // BST stats [
```

```
                if (PseudoLinkedPointerNEWleft == 0 && PseudoLinkedPointerNEWright == 0) {BSTcurrentLeaf++; BSTsTotalLEAFS++;}
                BSTcurrentPeak++;
                if (BSTcurrentPeakMAX < BSTcurrentPeak) BSTcurrentPeakMAX = BSTcurrentPeak;
              BSTstack[StackPtr] = BSTcurrentPeak; ++StackPtr; //needed for stats not for recursion
        // BST stats ]
              PseudoLinkedPointer = PseudoLinkedPointerNEWright; // choose right instead of 'PseudoLinkedPointerNEWleft' because of stats
print
            }
        if (StackPtr == 0) break;
        BSTcurrentPeak = BSTstack[--StackPtr]; // level of the node(1 is root) needed only for stats(print)
        PseudoLinkedPointerNEWleft = BSTstack[--StackPtr]; // left pointer needed only for stats(print)
        PseudoLinkedPointerNEWright = BSTstack[--StackPtr]; // right pointer
        memcpy( wrd, BSTstack[--StackPtr], i%31+1 );
        fwrite(wrd, i%31+1, 1, fp_out); TotalWLchars = TotalWLchars + i%31+1;
        fwrite(CRdLFa, 2, 1, fp_out);
        BSTcurrentNode++;
        PseudoLinkedPointer = PseudoLinkedPointerNEWleft;
  }
// ------------------------------------------------------------------------ ]
        // BST stats [
        if (BSTwithMAXnode < BSTcurrentNode) {
           BSTwithMAXnode = BSTcurrentNode;
           BSTwithMAXnodePEAK = BSTcurrentPeakMAX;
           BSTwithMAXnodeLEAF = BSTcurrentLeaf;
           BSTcurrentNodeMAXQUANTITY = 0;
        }
        if (BSTwithMAXnode == BSTcurrentNode) BSTcurrentNodeMAXQUANTITY++;
        if (BSTwithMAXpeak < BSTcurrentPeakMAX) {
           BSTwithMAXpeak = BSTcurrentPeakMAX;
           BSTwithMAXpeakNODE = BSTcurrentNode;
           BSTwithMAXpeakLEAF = BSTcurrentLeaf;
           BSTcurrentPeakMAXQUANTITY = 0;      iBSTwithMAXpeak=i; jBSTwithMAXpeak=j;
        }
        if (BSTwithMAXpeak == BSTcurrentPeakMAX) BSTcurrentPeakMAXQUANTITY++;
        if (BSTwithMAXleaf < BSTcurrentLeaf) {
           BSTwithMAXleaf = BSTcurrentLeaf;
           BSTwithMAXleafNODE = BSTcurrentNode;
           BSTwithMAXleafPEAK = BSTcurrentPeakMAX;
           BSTcurrentLeafMAXQUANTITY = 0;
        }
        if (BSTwithMAXleaf == BSTcurrentLeaf) BSTcurrentLeafMAXQUANTITY++;
        // BST stats ]
// ========================================================= BST traverse ]
} else
{
// $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$ B-tree order 3 traverse [
     // DONE JOB:
     // Must be written B-tree traverse ! with simulated stack i.e. non-recursive.
     // ...
  StackPtr = 0;
  while ( 2==2 ) {
        while (PseudoLinkedPointer != 0)
              {
                if (StackPtr > 8192*3-1) { printf( "\nLeprechaun: Failure! B-tree simulated stack overflow, too high B-tree!\n" ); return( 13
);}
                BSTstack[StackPtr] = PseudoLinkedPointer; ++StackPtr; //ptr to Rwrd
if ( *(char *)(PseudoLinkedPointer + 4 + 4 + 4 + i%31+1) == 0 ) {memcpy( PseudoLinkedPointer + 4 + 4, &BufStart[NumberOfSLOTs*4], 4 );}
                memcpy( &PseudoLinkedPointerNEWleft, PseudoLinkedPointer, 4 );
                memcpy( &PseudoLinkedPointerNEWmiddle, PseudoLinkedPointer + 4, 4 );
                memcpy( &PseudoLinkedPointerNEWright, PseudoLinkedPointer + 4 + 4, 4 );
// Give first from right to left non-zero PTR
                if (PseudoLinkedPointerNEWright !=0 )
                { memcpy( PseudoLinkedPointer + 4 + 4, &BufStart[NumberOfSLOTs*4], 4 );
                  PseudoLinkedPointer = PseudoLinkedPointerNEWright;
                }
                else if (PseudoLinkedPointerNEWmiddle !=0 )
                { memcpy( PseudoLinkedPointer + 4, &BufStart[NumberOfSLOTs*4], 4 );
                  PseudoLinkedPointer = PseudoLinkedPointerNEWmiddle;
                }
                else if (PseudoLinkedPointerNEWleft !=0 )
                { memcpy( PseudoLinkedPointer, &BufStart[NumberOfSLOTs*4], 4 );
                  PseudoLinkedPointer = PseudoLinkedPointerNEWleft;
                }
                else
                {
                  PseudoLinkedPointer = 0;
                }
              }
        if (StackPtr == 0) break;
        PseudoLinkedPointer = BSTstack[--StackPtr];
                memcpy( &PseudoLinkedPointerNEWleft, PseudoLinkedPointer, 4 );
                memcpy( &PseudoLinkedPointerNEWmiddle, PseudoLinkedPointer + 4, 4 );
                memcpy( &PseudoLinkedPointerNEWright, PseudoLinkedPointer + 4 + 4, 4 );
        if (PseudoLinkedPointerNEWleft+PseudoLinkedPointerNEWmiddle+PseudoLinkedPointerNEWright == 0) // One LEAF is PRINTED when LP=0 MP=0
RP=0
            {
        memcpy( wrd, PseudoLinkedPointer + 4 + 4 + 4, i%31+1 );
        fwrite(wrd, i%31+1, 1, fp_out); TotalWLchars = TotalWLchars + i%31+1;
```



```
        fwrite(CRdLFa, 2, 1, fp_out);
        if ( *(char *)(PseudoLinkedPointer + 4 + 4 + 4 + i%31+1) != 0 )
             { memcpy( wrd, PseudoLinkedPointer + 4 + 4 + 4, i%31+1 );
               fwrite(wrd, i%31+1, 1, fp_out); TotalWLchars = TotalWLchars + i%31+1;
               fwrite(CRdLFa, 2, 1, fp_out);
             }
        PseudoLinkedPointer = 0;
            }
  }
// $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$ B-tree order 3 traverse ]
}
                     }
  } // j

} // i

if (BSTorBtree == 0)
{
// ~~~~~ Longest path ~~~~~ [
i=iBSTwithMAXpeak; j=jBSTwithMAXpeak;
  BufStart = pointerflush + (i / 31) * WHOLEletter_BufferSize + OffsetsInBuffer[i % 31];
                     Slot = j<<2;
                     memcpy( &PseudoLinkedPointer, BufStart+Slot, 4 );
                     if (PseudoLinkedPointer != 0)
                     {
// ---------------------------------------------------------------------- [
  BSTcurrentNode = 0; BSTcurrentPeak = 0; BSTcurrentLeaf = 0;
                     BSTcurrentPeakMAX = 0; // Height of current BST
  StackPtr = 0;
        // BST print [
                fprintf( fp_outLOG, "A(not always THE) Binary-Search-Tree with the longest path(height, PEAK, number of levels):\n" );
        // BST print ]
  while ( 2==2 ) {
        while (PseudoLinkedPointer != 0)
              {
                if (StackPtr > 8192*3-1-3) { printf( "\nLeprechaun: Failure! BST simulated stack overflow, too high BST!\n" ); return( 13 );}
                memcpy( &PseudoLinkedPointerNEWleft, PseudoLinkedPointer, 4 );
                PseudoLinkedPointer = PseudoLinkedPointer + 4;
                memcpy( &PseudoLinkedPointerNEWright, PseudoLinkedPointer, 4 );
                BSTstack[StackPtr] = PseudoLinkedPointer + 4; ++StackPtr; //ptr to wrd
                BSTstack[StackPtr] = PseudoLinkedPointerNEWright; ++StackPtr;
                BSTstack[StackPtr] = PseudoLinkedPointerNEWleft; ++StackPtr; //needed for stats not for recursion
        // BST stats [
                  if (PseudoLinkedPointerNEWleft == 0 && PseudoLinkedPointerNEWright == 0) {BSTcurrentLeaf++;} //BSTsTotalLEAFS++;} // REMOVED
to avoid mess in TOTAL stats
                  BSTcurrentPeak++;
                  if (BSTcurrentPeakMAX < BSTcurrentPeak) BSTcurrentPeakMAX = BSTcurrentPeak;
                BSTstack[StackPtr] = BSTcurrentPeak; ++StackPtr; //needed for stats not for recursion
        // BST stats ]
                PseudoLinkedPointer = PseudoLinkedPointerNEWright; // choose right instead of 'PseudoLinkedPointerNEWleft' because of stats
print
              }
        if (StackPtr == 0) break;
        BSTcurrentPeak = BSTstack[--StackPtr]; // level of the node(1 is root) needed only for stats(print)
        PseudoLinkedPointerNEWleft = BSTstack[--StackPtr]; // left pointer needed only for stats(print)
        PseudoLinkedPointerNEWright = BSTstack[--StackPtr]; // right pointer
        memcpy( wrd, BSTstack[--StackPtr], i%31+1 );
        //fwrite(wrd, i%31+1, 1, fp_out); TotalWLchars = TotalWLchars + i%31+1;
        //fwrite(CRdLFa, 2, 1, fp_out);
        BSTcurrentNode++;
        PseudoLinkedPointer = PseudoLinkedPointerNEWleft;
        // BST print [
                for( k = 0; k < BSTcurrentPeak; k++ ) fprintf( fp_outLOG, "%c", ' ' );
                  if (PseudoLinkedPointerNEWleft == 0) fprintf( fp_outLOG, "[" ); else fprintf( fp_outLOG, "]" );
                for( k = 0; k < i%31+1; k++ ) fprintf( fp_outLOG, "%c", *(char *)(wrd+k) );
                  if (PseudoLinkedPointerNEWright == 0) fprintf( fp_outLOG, "]" ); else fprintf( fp_outLOG, "[" );
                  if (BSTcurrentPeak == 1) fprintf( fp_outLOG, " ROOT" );
                  fprintf( fp_outLOG, "\n" );
        // BST print ]
  }
// ---------------------------------------------------------------------- ]
                     }
fprintf( fp_outLOG, "Above Binary-Search-Tree with MaxPEAK = %s has NODEs = %s and LEAFs = %s\n", _ui64toaKAZEcomma(BSTwithMAXpeak,
llTOaDigits2, 10), _ui64toaKAZEcomma(BSTwithMAXpeakNODE, llTOaDigits3, 10), _ui64toaKAZEcomma(BSTwithMAXpeakLEAF, llTOaDigits, 10));
                fprintf( fp_outLOG, "Legend:\n" );
                fprintf( fp_outLOG, "At left side of the word - '[' means no left successor\n" );
                fprintf( fp_outLOG, "At left side of the word - ']' means left successor exists\n" );
                fprintf( fp_outLOG, "At right side of the word - ']' means no right successor\n" );
                fprintf( fp_outLOG, "At right side of the word - '[' means right successor exists\n" );
// ~~~~~ Longest path ~~~~~ ]

        // BST stats [
PEAKibBST=1+floorLog2(BSTwithMAXnode);
//PEAKibBST=1;
//while (BSTwithMAXnode>>PEAKibBST) PEAKibBST++;
        // BST stats ]
}
```

```
6096 (void) time(&t2);
6097 if (t2 <= t1) {t2 = t1; t2++;}
6098 printf("Time for making unsorted wordlist: %d second(s)\n", (int) t2-t1);
6099 fprintf( fp_outLOG, "Bytes per second performance: %sB/s\n", _ui64toaKAZEcomma(FilesLEN/((int) t3-t1), llTOaDigits, 10) );  // Rev. 12+
6100 fprintf( fp_outLOG, "Words per second performance: %sW/s\n", _ui64toaKAZEcomma(WORDcount/((int) t3-t1), llTOaDigits, 10) );  // Rev. 12+
6101 fprintf( fp_outLOG, "Input File with a list of TEXTual Files: %s\n", argv[1] );
6102 fprintf( fp_outLOG, "Size of all TEXTual Files: %s\n", _ui64toaKAZEcomma(FilesLEN, llTOaDigits, 10) );
6103 fprintf( fp_outLOG, "Word count: %s of them %s distinct\n", _ui64toaKAZEcomma(WORDcount, llTOaDigits, 10), _ui64toaKAZEcomma((unsigned long
     long)WORDcountDistinct, llTOaDigits2, 10) );
6104 fprintf( fp_outLOG, "Number Of Files: %lu\n", NumberOfFiles );
6105 fprintf( fp_outLOG, "Number Of Lines: %lu\n", NumberOfLines );
6106 fprintf( fp_outLOG, "Allocated memory in MB: %lu\n", (unsigned long)(memory_size>>20)+1 );
6107 NumberOfHashCollisions = WORDcountDistinct - NumberOfTrees;
6108 fprintf( fp_outLOG, "Number Of Trees(GREATER THE BETTER): %lu\n", NumberOfTrees );
6109 fprintf( fp_outLOG, "Forest population(Hash Function Quality regarding Collisions i.e. Hash Table Utilization): %lu%s\n",
     (NumberOfTrees*100)/(26*31*8192), "%\0" );
6110 fprintf( fp_outLOG, "Number Of Hash Collisions(Distinct WORDs - Number Of Trees): %lu\n", NumberOfHashCollisions );
6111
6112 if (BSTorBtree == 0)
6113 {
6114 fprintf( fp_outLOG, "Maximum Attempts to Find/Put a WORD into a Binary-Search-Tree: '%s'\n", _ui64toaKAZEcomma(BSTwithMAXpeak, llTOaDigits,
     10) );
6115 fprintf( fp_outLOG, "Total Attempts to Find/Put WORDs into Binary-Search-Trees: %s\n", _ui64toaKAZEcomma(WORDcountAttemptsToPut, llTOaDigits,
     10) );
6116 fprintf( fp_outLOG, "Total Number of LEAFs in Binary-Search-Trees(GREATER THE BETTER): %s\n", _ui64toaKAZEcomma(BSTsTotalLEAFs, llTOaDigits,
     10) );
6117 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, llTOaDigits, 10), _ui64toaKAZEcomma(PEAKibBST, llTOaDigits2, 10), _ui64toaKAZEcomma(BSTwithMAXnode,
     llTOaDigits3, 10));
6118 fprintf( fp_outLOG, "Binary-Search-Tree(1st out of %s) with MaxNODEs = %s has PEAK = %s and LEAFs = %s\n",
     _ui64toaKAZEcomma(BSTcurrentNodeMAXqUANTITY, llTOaDigits4, 10), _ui64toaKAZEcomma(BSTwithMAXnode, llTOaDigits2, 10),
     _ui64toaKAZEcomma(BSTwithMAXnodePEAK, llTOaDigits3, 10), _ui64toaKAZEcomma(BSTwithMAXnodeLEAF, llTOaDigits, 10));
6119 fprintf( fp_outLOG, "Binary-Search-Tree(1st out of %s) with MaxPEAK = '%s' has NODEs = %s and LEAFs = %s\n",
     _ui64toaKAZEcomma(BSTcurrentPeakMAXqUANTITY, llTOaDigits4, 10), _ui64toaKAZEcomma(BSTwithMAXpeak, llTOaDigits2, 10),
     _ui64toaKAZEcomma(BSTwithMAXpeakNODE, llTOaDigits3, 10), _ui64toaKAZEcomma(BSTwithMAXpeakLEAF, llTOaDigits, 10));
6120 fprintf( fp_outLOG, "Binary-Search-Tree(1st out of %s) with MaxLEAFs = %s has NODEs = %s and PEAK = %s\n",
     _ui64toaKAZEcomma(BSTcurrentLeafMAXqUANTITY, llTOaDigits4, 10), _ui64toaKAZEcomma(BSTwithMAXleaf, llTOaDigits2, 10),
     _ui64toaKAZEcomma(BSTwithMAXleafNODE, llTOaDigits3, 10), _ui64toaKAZEcomma(BSTwithMAXleafPEAK, llTOaDigits, 10));
6121 } else
6122 {
6123 fprintf( fp_outLOG, "Total Attempts to Find/Put WORDs into B-trees order 3: %s\n", _ui64toaKAZEcomma(WORDcountAttemptsToPut, llTOaDigits, 10)
     );
6124 }
6125
6126 for( k = 1; k < 32; k++ )
6127 { 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
6128   if ( MAXusedBufferABS < (31 * ((MAXusedBuffer[k]>>10)+1)) / GRMBLhill[(int)k] ) {MAXusedBufferABS = 1+(31 * ((MAXusedBuffer[k]>>10)+1)) /
     GRMBLhill[(int)k];}
6129   Utiliza1 = Utiliza1 + (MAXusedBuffer[k]>>10)+1;
6130   Utiliza2 = Utiliza2 + (((GRMBLhill[(int)k] * LetterBuffer)/31)>>10)+1;
6131 }
6132 fprintf( fp_outLOG, "Total pseudo(including hash table) memory utilization: %s%s\n", _ui64toaKAZEzerocomma((Utiliza1*100)/Utiliza2,
     llTOaDigits, 10)+(26-2), "%\0" ); // 26 are all 26-DESIRED=24
6133 fprintf( fp_outLOG, "Total real(wordlist's words VS allocated block) memory utilization: %s/1000\n", _i64toaKAZE(((unsigned long
     long)TotalWLchars*1000)/memory_size, llTOaDigits, 10) ); // 26 are all 26-DESIRED=24
6134 fprintf( fp_outLOG, "Used value for third parameter in KB: %lu\n", (unsigned long)Thunderwith );
6135 fprintf( fp_outLOG, "Use next time as third parameter: %lu-\n", MAXusedBufferABS ); // 26 are all 26-DESIRED=24
6136 fprintf( fp_outLOG, "Time for making unsorted wordlist: %d second(s)\n", (int) t2-t1);
6137
6138 // EXIT EXIT EXIT EXIT EXIT EXIT EXIT EXIT EXIT EXIT EXIT EXIT EXIT EXIT EXIT
6139 printf( "Deallocated memory in MB: %lu\n", (memory_size>>20)+1 );
6140 free(pointerflushUNALIGN);
6141 fclose(fp_out);
6142 fclose(fp_outLOG);
6143 // SORT SORT SORT SORT SORT SORT SORT SORT SORT SORT SORT SORT SORT SORT SORT
6144    //printf("Uploading unsorted wordlist ...\n");
6145    if ((nlines = readlines(argv[2], &backup)) >= 0)
6146    {  //printf("Number of words(lines) uploaded: %lu\n", nlines);
6147       //printf("Note1: Press 'Ctrl+C' to abort sorting, unsorted wordlist(second parameter)\n");
6148       //printf("       will remain intact(unless flushing is in progress) because of\n");
6149       //printf("       pointers-to-data are being sorted not the data itself.\n");
6150       //printf("Note2: In near future 'InsertionX26Sort' will be replaced with 'QuickX26Sort':\n");
6151       //printf("       which is much faster than 'QuickSort' applied for data-at-once!\n");
6152
6153       // !!!??? I AM DISAPPOINTED x26 is just an illusion !!!???
6154       // X26 X26 X26 X26 X26 X26 X26 X26 X26 X26 X26 X26 X26 X26 X26 X26 X26
6155 // argc is 4|5|6 due to eventual missing BufferSize
6156 if( argc == 4 ) // not 5 due to eventual missing BufferSize
6157     k_FIX = 3;
6158 if( argc == 5 || argc == 6 )
6159     k_FIX = 4;
6160 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')
6161 { printf("Sorting(with 'MultiKeyQuickSortX26Sort' by J. Bentley and R. Sedgewick) ...\n");
6162       /* ???!!! what an unexpected behavior! I have been hit: for SOED5.HTM(259,835 distinct words): InsertionX26Sort gives 46s, InsertionSort
```



```
     gives 28s */
6163       for( k = 0; k < 26; k++ )
6164       { printf( "Sort pass %s/26 ...\r", _ui64toaKAZEzerocomma(k+1, llTOaDigits, 10)+(26-2));
6165         HEADOffetFromStartBUKVA = TAILOffetFromStartBUKVA;
6166         while( (TAILOffetFromStartBUKVA < nlines) && (*backup[TAILOffetFromStartBUKVA] - 'a' == k) )
6167         { TAILOffetFromStartBUKVA++;
6168         }
6169         if (HEADOffetFromStartBUKVA != TAILOffetFromStartBUKVA)
6170         { mkqsort_main(backup + HEADOffetFromStartBUKVA, TAILOffetFromStartBUKVA - HEADOffetFromStartBUKVA + 0); // backup[0..nlines-1]
6171         }
6172       }
6173 }
6174 else
6175 {
6176 if (*argv[k_FIX] == 'A' || *argv[k_FIX] == 'a')
6177    { printf("Sorting(with 'InsertionSort') ...");
6178      InsertSortKAZE(backup, nlines, 0); // backup[0..nlines-1]
6179    }
6180 if (*argv[k_FIX] == 'B' || *argv[k_FIX] == 'b')
6181    { printf("Sorting(with 'InsertionX26Sort') ...\n");
6182       /* ???!!! what an unexpected behavior! I have been hit: for SOED5.HTM(259,835 distinct words): InsertionX26Sort gives 46s, InsertionSort
     gives 28s */
6183       for( k = 0; k < 26; k++ )
6184       { printf( "Sort pass %s/26 ...\r", _ui64toaKAZEzerocomma(k+1, llTOaDigits, 10)+(26-2));
6185         HEADOffetFromStartBUKVA = TAILOffetFromStartBUKVA;
6186         while( (TAILOffetFromStartBUKVA < nlines) && (*backup[TAILOffetFromStartBUKVA] - 'a' == k) )
6187         { TAILOffetFromStartBUKVA++;
6188         }
6189         if (HEADOffetFromStartBUKVA != TAILOffetFromStartBUKVA)
6190         { InsertSortKAZE(backup + HEADOffetFromStartBUKVA, TAILOffetFromStartBUKVA - HEADOffetFromStartBUKVA + 0, 0); // backup[0..nlines-1]
6191         }
6192       }
6193    }
6194 if (*argv[k_FIX] == 'C' || *argv[k_FIX] == 'c')
6195    { printf("Sorting(with 'MultiKeyQuickSortSort' by J. Bentley and R. Sedgewick) ...");
6196      mkqsort_main(backup, nlines); // backup[0..nlines-1]
6197    }
6198 if (*argv[k_FIX] == 'D' || *argv[k_FIX] == 'd')
6199    { printf("Sorting(with 'MultiKeyQuickSortX26Sort' by J. Bentley and R. Sedgewick) ...\n");
6200       /* ???!!! what an unexpected behavior! I have been hit: for SOED5.HTM(259,835 distinct words): InsertionX26Sort gives 46s, InsertionSort
     gives 28s */
6201       for( k = 0; k < 26; k++ )
6202       { printf( "Sort pass %s/26 ...\r", _ui64toaKAZEzerocomma(k+1, llTOaDigits, 10)+(26-2));
6203         HEADOffetFromStartBUKVA = TAILOffetFromStartBUKVA;
6204         while( (TAILOffetFromStartBUKVA < nlines) && (*backup[TAILOffetFromStartBUKVA] - 'a' == k) )
6205         { TAILOffetFromStartBUKVA++;
6206         }
6207         if (HEADOffetFromStartBUKVA != TAILOffetFromStartBUKVA)
6208         { mkqsort_main(backup + HEADOffetFromStartBUKVA, TAILOffetFromStartBUKVA - HEADOffetFromStartBUKVA + 0); // backup[0..nlines-1]
6209         }
6210       }
6211    }
6212 }
6213       // X26 X26 X26 X26 X26 X26 X26 X26 X26 X26 X26 X26 X26 X26 X26 X26 X26
6214
6215       printf("\nFlushing sorted words ...\n");
6216       if( ( fp_out = fopen( argv[2], "wb+" ) ) == NULL )
6217       { printf( "Leprechaun: Can't create file %s \n", argv[2] ); return( 1 ); }
6218       for( j = 0; j < nlines; j++ )
6219       { //slot = KuxHash3plus(backup[j]);
6220         //fprintf(fp_out, "Hashcode: %s - ", _ui64toaKAZEzerocomma(Slot, llTOaDigits, 10)+(26-5));
6221         fprintf(fp_out, "%s", backup[j]);
6222         fwrite(CRdLFa, 2, 1, fp_out );
6223       }
6224       (void) time(&t3);
6225       if (t3 <= t2) {t3 = t2; t3++;}
6226       printf("Time for sorting unsorted wordlist: %d second(s)\n", (int) t3-t2);
6227
6228 /*
6229 // Hash benchmarking ---------------------------------------------- [
6230
6231 // 5[
6232 clocks1 = clock();
6233    for (Bozan=0; Bozan < (1<<4); Bozan++) // 16 times, at end >>4
6234 {
6235       for( j = 0; j < nlines; j++ )
6236       { //slot = KuxHash3plus(backup[j]);
6237         //fprintf(fp_out, "Hashcode: %s - ", _ui64toaKAZEzerocomma(Slot, llTOaDigits, 10)+(26-5));
6238               Slot = FNV1A_Hash_4_OCTETS(backup[j], (strlen(backup[j])>>2)); //13+++
6239       }
6240 }
6241 clocks2 = clock();
6242 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))  );
6243 // 5]
6244
6245 // 1[
6246 clocks1 = clock();
```

```
    for (Bozan=0; Bozan < (1<<4); Bozan++) // 16 times, at end >>4
{
      for( j = 0; j < nlines; j++ )
      { //Slot = KuxHash3plus(backup[j]);
        //fprintf(fp_out, "Hashcode: %s - ", _ui64toaKAZEzerocomma(Slot, llTOaDigits, 10)+(26-5));
// To make it EVEN !!!
//wrdlen = strlen(backup[j]);
//if (strlen(backup[j]) != 0)
              Slot = FNV1A_Hash(backup[j]); //13+++
      }
}
clocks2 = clock();
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))  );
// 1]

// 2[
clocks1 = clock();
    for (Bozan=0; Bozan < (1<<4); Bozan++) // 16 times, at end >>4
{
      for( j = 0; j < nlines; j++ )
      { //Slot = KuxHash3plus(backup[j]);
        //fprintf(fp_out, "Hashcode: %s - ", _ui64toaKAZEzerocomma(Slot, llTOaDigits, 10)+(26-5));
              Slot = FNV1A_Hash_4_OCTETS_31(backup[j], (strlen(backup[j])>>2)); //13+++
      }
}
clocks2 = clock();
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))  );
// 2]

// 4[
clocks1 = clock();
    for (Bozan=0; Bozan < (1<<4); Bozan++) // 16 times, at end >>4
{
      for( j = 0; j < nlines; j++ )
      { //Slot = KuxHash3plus(backup[j]);
        //fprintf(fp_out, "Hashcode: %s - ", _ui64toaKAZEzerocomma(Slot, llTOaDigits, 10)+(26-5));
// To make it EVEN !!!
//wrdlen = strlen(backup[j]);
//if (strlen(backup[j]) != 0)
              Slot = KuxHash3plus(backup[j]); //13++
      }
}
clocks2 = clock();
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))  );
// 4]

// 6[
clocks1 = clock();
    for (Bozan=0; Bozan < (1<<4); Bozan++) // 16 times, at end >>4
{
      for( j = 0; j < nlines; j++ )
      { //Slot = KuxHash3plus(backup[j]);
        //fprintf(fp_out, "Hashcode: %s - ", _ui64toaKAZEzerocomma(Slot, llTOaDigits, 10)+(26-5));
wrdlen = strlen(backup[j]);
if (wrdlen<=19) // 4x4+3=19 i.e. last contains 7 clashes
              Slot = FNV1A_Hash_Granularity(backup[j], wrdlen>>2, 2); //13+++++
else            // 2x8+4=20 i.e. first contains 6 clashes
              Slot = FNV1A_Hash_Granularity(backup[j], wrdlen>>3, 3); //13+++++
      } // Conclusion: two functions > 64 bytes lead to horrible slowness, so unite them in one: fit in the cache line.
}
clocks2 = clock();
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))  );
// 6]

// Hash benchmarking --------------------------------------------- ]
*/

if( ( fp_outLOG = fopen( "Leprechaun.LOG", "a+" ) ) == NULL )
{ printf( "Leprechaun: Can't open file Leprechaun.LOG.\n" ); return( 1 ); }
fprintf( fp_outLOG, "Time for sorting unsorted wordlist: %d second(s)\n\n", (int) t3-t2);
printf( "Leprechaun: Done.\n" );
      return 0;
    }
    else
    {  printf("Leprechaun: Input file too large, wordlist remains unsorted!\n");
       return 1;
    }
// SORT SORT SORT SORT SORT SORT SORT SORT SORT SORT SORT SORT SORT SORT SORT
} else { //if (BSTorBtree != 2) {
// External Btrees [

// r16 [
if (REUSE==2) {
    fclose(fp_out);
```



```
}
// r16 ]

    if ( REUSE == 0 ) { // r16FIX <*><*><*><*><*><*><*><*><*><*><*><*><*><*><*><*><*><*><*><*><*><*><*> [

(void) time(&t1);
WORDcountBOTTOM = 0;
//printf("Flushing UNsorted phrases ...\r");
//if( ( fp_out = fopen( argv[2], "wb+" ) ) == NULL ) // Commented since r.14++++ because passes need concatenation.

if( ( fp_out = fopen( argv[2], "ab+" ) ) == NULL )
{ printf( "Leprechaun: Can't create file %s \n", argv[2] ); return( 1 ); }
CRdLFa[0] = 13; CRdLFa[1] = 10;

BufStart = pointerflush;
//   for( j = 0; j < 28*28*28*28*28; j++ )
for( j = 0; j < (1<<HashInBITS); j++ )
{
                              if ((j & ((1<<14)-1)) == 0) {
                    (void) time(&t3);
                    if (t3 <= t1) {t3 = t1; t3++;}
                    printf("Flushing UNsorted phrases: %s%%; Shaking trees performance: %sP/s\r", _ui64toaKAZEzerocomma(((long
long)j*100)/((1<<HashInBITS)), llTOaDigits, 10)+(26-3), _ui64toaKAZEzerocomma(WORDcountBOTTOM/((int) t3-t1), llTOaDigits2, 10)+(26-10));
                    }
                   Slot = j<<3;
                   memcpy( &PseudoLinkedPointer_64, BufStart+Slot, 8 );
                   if (PseudoLinkedPointer_64 != 0)
                   {
NumberOfTrees++;

// $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$ B-tree order 3 traverse 64bit [
    // DONE JOB:
    // Must be written B-tree traverse ! with simulated stack i.e. non-recursive.
    // ...
  StackPtr = 0;
  while ( 2==2 ) {
       while (PseudoLinkedPointer_64 != 0)
             {
               if (StackPtr > 8192*3-1) { printf( "\nLeprechaun: Failure! B-tree simulated stack overflow, too high B-tree!\n" ); return( 13
);}
               BSTstack[StackPtr] = PseudoLinkedPointer_64; ++StackPtr; //ptr to Rwrd
//if ( *(char *)(PseudoLinkedPointer + 4 + 4 + 4 + i%31+1) == 0 ) {memcpy( PseudoLinkedPointer + 4 + 4, &BufStart[NumberOfSLOTs*4], 4 );}
             // [  //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
    //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1+4); //RW
    //fsetpos(fp_outRG, &PseudoLinkedPointerAUX_64);
    //fread(&SomeByte, 1, 1, fp_outRG);
             // ]  //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
             // [  //r.14+
             PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64;
                       if (BSTorBtree == 2) {
             fsetpos(fp_outRG, &PseudoLinkedPointerAUX_64);
             fread(&LEAF[0], 8+8+8+2*(LongestLineInclusive+1+4), 1, fp_outRG);
                       } else { // ########## 64bit memory manipulations [
             memcpy( &LEAF[0], (char *)PseudoLinkedPointerAUX_64, 8+8+8+2*(LongestLineInclusive+1+4) );
                       } // ########## 64bit memory manipulations ]
             memcpy( &SomeByte, &LEAF[8 + 8 + 8 + (LongestLineInclusive+1+4)], 1 );
             // ]  //r.14+
    if (SomeByte == 0 ) // RW exists not
     {
      PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8;
                       if (BSTorBtree == 2) {
      fsetpos(fp_outRG, &PseudoLinkedPointerAUX_64);
      fwrite(&NULLs_64, 8, 1, fp_outRG);
                       } else { // ########## 64bit memory manipulations [
             memcpy( (char *)PseudoLinkedPointerAUX_64, &NULLs_64, 8 );
                       } // ########## 64bit memory manipulations ]
             // [  //r.14+
             memcpy( &LEAF[8 + 8], &NULLs_64, 8 );
             // ]  //r.14+
     }
//               memcpy( &PseudoLinkedPointerNEWleft, PseudoLinkedPointer, 4 );
//               memcpy( &PseudoLinkedPointerNEWmiddle, PseudoLinkedPointer + 4, 4 );
//               memcpy( &PseudoLinkedPointerNEWright, PseudoLinkedPointer + 4 + 4, 4 );
             // [  //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
      //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64;
      //fsetpos(fp_outRG, &PseudoLinkedPointerAUX_64);
      //fread(&PseudoLinkedPointerNEWleft_64, 8, 1, fp_outRG);
       //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8;
       //fsetpos(fp_outRG, &PseudoLinkedPointerAUX_64);
       //fread(&PseudoLinkedPointerNEWmiddle_64, 8, 1, fp_outRG);
        //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8;
        //fsetpos(fp_outRG, &PseudoLinkedPointerAUX_64);
        //fread(&PseudoLinkedPointerNEWright_64, 8, 1, fp_outRG);
             // ]  //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
             // [  //r.14+
             memcpy( &PseudoLinkedPointerNEWleft_64, &LEAF[0], 8 );
             memcpy( &PseudoLinkedPointerNEWmiddle_64, &LEAF[8], 8 );
             memcpy( &PseudoLinkedPointerNEWright_64, &LEAF[8+8], 8 );
```

```c
                // ]  //r.14+
// Give first from right to left non-zero PTR
                if (PseudoLinkedPointerNEWright_64 !=0 )
//                { memcpy( PseudoLinkedPointer + 4 + 4, &BufStart[NumberOfSLOTs*4], 4 );
//                  PseudoLinkedPointer = PseudoLinkedPointerNEWright;
//                }
                {
      PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8;
                        if (BSTorBtree == 2) {
      fsetpos(fp_outRG, &PseudoLinkedPointerAUX_64);
      fwrite(&NULLs_64, 8, 1, fp_outRG);
                        } else { // ########## 64bit memory manipulations [
               memcpy( (char *)PseudoLinkedPointerAUX_64, &NULLs_64, 8 );
                        } // ########## 64bit memory manipulations ]
       PseudoLinkedPointer_64 = PseudoLinkedPointerNEWright_64;
                }
                else if (PseudoLinkedPointerNEWmiddle_64 !=0 )
//                { memcpy( PseudoLinkedPointer + 4, &BufStart[NumberOfSLOTs*4], 4 );
//                  PseudoLinkedPointer = PseudoLinkedPointerNEWmiddle;
//                }
                {
      PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8;
                        if (BSTorBtree == 2) {
      fsetpos(fp_outRG, &PseudoLinkedPointerAUX_64);
      fwrite(&NULLs_64, 8, 1, fp_outRG);
                        } else { // ########## 64bit memory manipulations [
               memcpy( (char *)PseudoLinkedPointerAUX_64, &NULLs_64, 8 );
                        } // ########## 64bit memory manipulations ]
       PseudoLinkedPointer_64 = PseudoLinkedPointerNEWmiddle_64;
                }
                else if (PseudoLinkedPointerNEWleft_64 !=0 )
//                { memcpy( PseudoLinkedPointer, &BufStart[NumberOfSLOTs*4], 4 );
//                  PseudoLinkedPointer = PseudoLinkedPointerNEWleft;
//                }
                {
      PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64;
                        if (BSTorBtree == 2) {
      fsetpos(fp_outRG, &PseudoLinkedPointerAUX_64);
      fwrite(&NULLs_64, 8, 1, fp_outRG);
                        } else { // ########## 64bit memory manipulations [
               memcpy( (char *)PseudoLinkedPointerAUX_64, &NULLs_64, 8 );
                        } // ########## 64bit memory manipulations ]
       PseudoLinkedPointer_64 = PseudoLinkedPointerNEWleft_64;
                }
                else
                {
                  PseudoLinkedPointer_64 = 0;
                }
            }
     if (LevelsInCorona_Not_Counting_ROOT < StackPtr) LevelsInCorona_Not_Counting_ROOT = StackPtr; //r.14
         if (StackPtr == 0) break;
         PseudoLinkedPointer_64 = BSTstack[--StackPtr];
//                memcpy( &PseudoLinkedPointerNEWleft, PseudoLinkedPointer, 4 );
//                memcpy( &PseudoLinkedPointerNEWmiddle, PseudoLinkedPointer + 4, 4 );
//                memcpy( &PseudoLinkedPointerNEWright, PseudoLinkedPointer + 4 + 4, 4 );
               // [  //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
        //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64;
        //fsetpos(fp_outRG, &PseudoLinkedPointerAUX_64);
        //fread(&PseudoLinkedPointerNEWleft_64, 8, 1, fp_outRG);
         //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8;
         //fsetpos(fp_outRG, &PseudoLinkedPointerAUX_64);
         //fread(&PseudoLinkedPointerNEWmiddle_64, 8, 1, fp_outRG);
          //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8;
          //fsetpos(fp_outRG, &PseudoLinkedPointerAUX_64);
          //fread(&PseudoLinkedPointerNEWright_64, 8, 1, fp_outRG);
               // ]  //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
               // [  //r.14+
               PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64;
                        if (BSTorBtree == 2) {
               fsetpos(fp_outRG, &PseudoLinkedPointerAUX_64);
               fread(&LEAF[0], 8+8+8+2*(LongestLineInclusive+1+4), 1, fp_outRG);
                        } else { // ########## 64bit memory manipulations [
               memcpy( &LEAF[0], (char *)PseudoLinkedPointerAUX_64, 8+8+8+2*(LongestLineInclusive+1+4) );
                        } // ########## 64bit memory manipulations ]
               memcpy( &PseudoLinkedPointerNEWleft_64, &LEAF[0], 8 );
               memcpy( &PseudoLinkedPointerNEWmiddle_64, &LEAF[8], 8 );
               memcpy( &PseudoLinkedPointerNEWright_64, &LEAF[8+8], 8 );
               // ]  //r.14+
         if (PseudoLinkedPointerNEWleft_64 + PseudoLinkedPointerNEWmiddle_64 + PseudoLinkedPointerNEWright_64 == 0) // One LEAF is PRINTED when
LP=0 MP=0 RP=0
            {
//         memcpy( wrd, PseudoLinkedPointer + 4 + 4 + 4, i%31+1 );
//         fwrite(wrd, i%31+1, 1, fp_out);
//         fwrite(CRdLFa, 2, 1, fp_out);
//         if ( *(char *)(PseudoLinkedPointer + 4 + 4 + 4 + i%31+1) != 0 )
//           { memcpy( wrd, PseudoLinkedPointer + 4 + 4 + 4 + i%31+1, i%31+1 );
//             fwrite(wrd, i%31+1, 1, fp_out);
//             fwrite(CRdLFa, 2, 1, fp_out);
```



```c
//             }
//        PseudoLinkedPointer = 0;
               // [  //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
     //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8;
     //fsetpos(fp_outRG, &PseudoLinkedPointerAUX_64);
     //fread(&wrd[0], (LongestLineInclusive+1+4), 1, fp_outRG);
               // ]  //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
               // [  //r.14+
               memcpy( &wrd[0], &LEAF[8 + 8 + 8], (LongestLineInclusive+1+4) );
               // ]  //r.14+
               // Counter [
               memcpy( &CounterOccurrencies, &wrd[(LongestLineInclusive+1+4)-4], 4 );
               if (CounterOccurrencies<9999999) CounterOccurrencies++; // Starting from ZERO! Because when insertion happened there was no
setting counter to 1.
               // Counter ]
if (*argv[k_FIX] == 'Y' || *argv[k_FIX] == 'Z') // || *argv[k_FIX] == 'W'
        fprintf(fp_out, "%s\t%s\r\n", _ui64toaKAZEzerocomma(CounterOccurrencies, llTOaDigits2, 10)+(26-9), wrd); WORDcountBOTTOM++;
if (*argv[k_FIX] == 'y' || *argv[k_FIX] == 'z') // || *argv[k_FIX] == 'w'
        fprintf(fp_out, "%s\r\n", wrd); WORDcountBOTTOM++;
               //fwrite(CRdLFa, 2, 1, fp_out);
               // [  //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
     //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1+4); //RW
     //fsetpos(fp_outRG, &PseudoLinkedPointerAUX_64);
     //fread(&SomeByte, 1, 1, fp_outRG);
               // ]  //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
               // [  //r.14+
               memcpy( &SomeByte, &LEAF[8 + 8 + 8 + (LongestLineInclusive+1+4)], 1 );
               // ]  //r.14+
     if (SomeByte != 0 ) // RW exists
     {
               // [  //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
     //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1+4);
     //fsetpos(fp_outRG, &PseudoLinkedPointerAUX_64);
     //fread(&wrd[0], (LongestLineInclusive+1+4), 1, fp_outRG);
               // ]  //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
               // [  //r.14+
               memcpy( &wrd[0], &LEAF[8 + 8 + 8 + (LongestLineInclusive+1+4)], (LongestLineInclusive+1+4) );
               // ]  //r.14+
               // Counter [
               memcpy( &CounterOccurrencies, &wrd[(LongestLineInclusive+1+4)-4], 4 );
               if (CounterOccurrencies<9999999) CounterOccurrencies++; // Starting from ZERO! Because when insertion happened there was no
setting counter to 1.
               // Counter ]
if (*argv[k_FIX] == 'Y' || *argv[k_FIX] == 'Z') // || *argv[k_FIX] == 'W'
               fprintf(fp_out, "%s\t%s\r\n", _ui64toaKAZEzerocomma(CounterOccurrencies, llTOaDigits2, 10)+(26-9), wrd); WORDcountBOTTOM++;
if (*argv[k_FIX] == 'y' || *argv[k_FIX] == 'z') // || *argv[k_FIX] == 'w'
               fprintf(fp_out, "%s\r\n", wrd); WORDcountBOTTOM++;
                //fwrite(CRdLFa, 2, 1, fp_out);
     }
      PseudoLinkedPointer_64 = 0;
    NumberOfLEAFs++; //r.14
           }
  }
// $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$ B-tree order 3 traverse 64bit ]

                     }
}

fclose(fp_out);
(void) time(&t2);
if (t2 <= t1) {t2 = t1; t2++;}
printf("Flushing UNsorted phrases: %s%%; Shaking trees performance: %sP/s\n", _ui64toaKAZEzerocomma(100, llTOaDigits, 10)+(26-3),
_ui64toaKAZEzerocomma(WORDcountBOTTOM/((int) t2-t1), llTOaDigits2, 10)+(26-10));
printf("Time for shaking phrases from trees: %d second(s)\n", (int) t2-t1);

// TO DO ...
//printf( "Leprechaun: THE DUMP NOT SORTED?! HASH sucks also - these two issues ought to be fixed in r.14. This is revision 14-.\n" );

     } //if ( REUSE == 0 ) { // r16FIX <*><*><*><*><*><*><*><*><*><*><*><*><*><*><*><*><*><*><*><*><*><*><*><*> ]

fprintf( fp_outLOG, "Number Of Hash Collisions(Distinct WORDs - Number Of Trees): %s\n", _ui64toaKAZEcomma(WORDcountDistinct - NumberOfTrees,
llTOaDigits, 10) );
fprintf( fp_outLOG, "Number Of Trees(GREATER THE BETTER): %s\n", _ui64toaKAZEcomma(NumberOfTrees, llTOaDigits, 10) );
fprintf( fp_outLOG, "Number Of LEAFs(littler THE BETTER) not counting ROOT LEAFs: %s\n", _ui64toaKAZEcomma(NumberOfLEAFs-NumberOfTrees,
llTOaDigits, 10) );
fprintf( fp_outLOG, "Highest Tree not counting ROOT Level i.e. CORONA levels(littler THE BETTER): %s\n",
_ui64toaKAZEcomma(LevelsInCorona_Not_Counting_ROOT-1, llTOaDigits, 10) );

fprintf( fp_outLOG, "Used value for third parameter in KB: %s\n", _ui64toaKAZEcomma(Thunderwith, llTOaDigits, 10) );
fprintf( fp_outLOG, "Use next time as third parameter: %s\n", _ui64toaKAZEcomma((((BufEnd_64-(unsigned long long)pointerflush_64)+1)>>10)+1,
llTOaDigits, 10) );
fprintf( fp_outLOG, "Total Attempts to Find/Put WORDs into B-trees order 3: %s\n", _ui64toaKAZEcomma(WORDcountAttemptsToPut, llTOaDigits, 10)
);

// External Btrees ]
                        if (BSTorBtree == 2) {
fclose(fp_outRG);
// r16
```

```
6584 if ( (REUSE == 1) && ((HashInBITS-HashChunkSizeInBITS)==0) ) { // Multiple-passes shouldn't be dumped - it is meaningless, dump when only one
     pass.
6585     if( ( fp_outRG = fopen( "Leprechaun_64bit.hsh", "wb+" ) ) == NULL )
6586     { printf( "Leprechaun: Can't create file 'Leprechaun_64bit.hsh'.\n" ); return( 1 ); }
6587     fwrite(pointerflushUNALIGN,  (1<<HashInBITS)*8 + 1 + 64 , 1, fp_outRG);
6588 fclose(fp_outRG);
6589 }
6590                         } else { // ########## 64bit memory manipulations [
6591 free(pointerflushUNALIGN_64);
6592                         } // ########## 64bit memory manipulations ]
6593 free(pointerflushUNALIGN);
6594 fclose(fp_outLOG);
6595 printf( "Leprechaun: Current pass done.\n" );
6596
6597 // 14++++ [
6598  TotalMemoryNeededForOnePass += (((BufEnd_64-(unsigned long long)pointerflush_64)+1)>>10)+1;
6599  WORDcountDistinctTOTAL += WORDcountDistinct;
6600  RipPasses++;
6601  if (RipPasses <= (1<<(HashInBITS-HashChunkSizeInBITS))-1) goto WhyTheHellForIsNotWorking;
6602 //} // for( RipPasses = 1-1; RipPasses <= (1<<(HashInBITS-HashChunkSizeInBITS))-1; RipPasses++ )
6603 // 14++++ ]
6604 (void) time(&tMainE);
6605 if (tMainE <= tMainB) {tMainE = tMainB; tMainE++;} // This line fixes a bug in r.15
6606 printf( "\nTotal memory needed for one pass: %sKB\n", _ui64toaKAZEcomma(TotalMemoryNeededForOnePass, llTOaDigits2, 10) );
6607 printf( "Total distinct phrases: %s\n", _ui64toaKAZEcomma(WORDcountDistinctTOTAL, llTOaDigits2, 10) );
6608 printf("Total time: %d second(s)\n", (int) tMainE-tMainB);
6609 printf( "Total performance: %sP/s i.e. phrases per second\n", _ui64toaKAZEcomma(WORDcount/((int) tMainE-tMainB), llTOaDigits2, 10) );
6610
6611 printf( "Leprechaun: Done.\n" );
6612 exit(0);
6613 } //if (BSTorBtree != 2) {
6614 } // main()
6615
6616 /*
6617 TO BE DONE: Ideal Balancing BST [
6618
6619 link rotR(link h)
6620   { link x = h->l; h->l = x->r; x->r = h;
6621     return x; }
6622
6623 link rotL(link h)
6624   { link x = h->r; h->r = x->l; x->l = h;
6625     return x; }
6626
6627 link partR(link h, int k)
6628   { int t = h->l->N;
6629     if (t > k )
6630       { h->l = partR(h->l, k); h = rotR(h); }
6631     if (t < k )
6632       { h->r = partR(h->r, k-t-1); h = rotL(h); }
6633     return h;
6634   }
6635
6636 link balanceR(link h)
6637   {
6638     if (h->N < 2) return h;
6639     h = partR(h, h->N/2);
6640     h->l = balanceR(h->l);
6641     h->r = balanceR(h->r);
6642     return h;
6643   }
6644
6645 TO BE DONE: Ideal Balancing BST ]
6646 */
6647
6648 /*
6649 #include <stdlib.h>
6650 #include "Item.h"
6651 typedef struct STnode* link;
6652 struct STnode { Item item; link l, r; int N };
6653 static link head, z;
6654 link NEW(Item item, link l, link r, int N)
6655   { link x = malloc(sizeof *x);
6656     x->item = item; x->l = l; x->r = r; x->N = N;
6657     return x;
6658   }
6659 void STinit()
6660   { head = (z = NEW(NULLitem, 0, 0, 0)); }
6661 int STcount() { return head->N; }
6662 Item searchR(link h, Key v)
6663   { Key t = key(h->item);
6664     if (h == z) return NULLitem;
6665     if eq(v, t) return h->item;
6666     if less(v, t) return searchR(h->l, v);
6667             else return searchR(h->r, v);
6668   }
6669 Item STsearch(Key v)
6670   { return searchR(head, v); }
```



```
6671 link insertR(link h, Item item)
6672   { Key v = key(item), t = key(h->item);
6673     if (h == z) return NEW(item, z, z, 1);
6674     if less(v, t)
6675          h->l = insertR(h->l, item);
6676     else h->r = insertR(h->r, item);
6677     (h->N)++; return h;
6678   }
6679 void STinsert(Item item)
6680   { head = insertR(head, item); }
6681 */
6682
6683 /*
6684 int count(link h)
6685   {
6686     if (h == NULL) return 0;
6687     return count(h->l) + count(h->r) + 1;
6688   }
6689
6690 int height(link h)
6691   { int u, v;
6692     if (h == NULL) return -1;
6693     u = height(h->l); v = height(h->r);
6694     if (u > v) return u+1; else return v+1;
6695   }
6696
6697 void printnode(char c, int h)
6698   { int i;
6699     for (i = 0; i < h; i++) printf("  ");
6700     printf("%c\n", c);
6701   }
6702
6703 void show(link x, int h)
6704   {
6705     if (x == NULL) { printnode("*", h); return; }
6706     show(x->r, h+1);
6707     printnode(x->item, h);
6708     show(x->l, h+1);
6709   }
6710 */
```