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

LEPRECHAUN

AN ENGLISH-WORDLIST RIPPER, REVISION 13+++++++

Free download at *www.sanmayce.com* – on Intel *Merom-1M* 2166 MHz it rips **wikipedia** at 2,860,880**++** words per second.

```
/*
This is source of Leprechaun revision 13_7pluses, copyleft Sanmayce, 2010-Nov-16.

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

Linux compile(uncomment #include <io.h> line, ignore warnings):
gcc -D_FILE_OFFSET_BITS=64 -m32 -static -O3 -mtune=generic Leprechaun.c -o Leprechaun_r13_7pluses_generic_32bits.elf

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

[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)
```

```
0174 #define PTR_ALIGNED_4(a)    ((PTR_LINEAR(a) & 3) == 0)
0175 */
0176
0177 //__declspec(align(64)) int BigArray[1024]; // Windows syntax
0178 //or
0179 //int BigArray[1024] __attribute__((aligned(64))); // Linux syntax
0180
0181 #if defined(_WIN32_ENVIRONMENT_)
0182 __declspec(align(64))
0183 #else
0184 //__attribute__((aligned(64)));
0185 #endif /* defined(_WIN32_ENVIRONMENT_)  */
0186
0187 typedef unsigned short WORD; // As for 'With *(DWORD*), a buffer overrun is possible at the end of a memory page.' I knew about it but was
     fooled by assembly code generated by VS2010 which translates it to a word access:
0188 //; 792 : hash32 = FNV_32A_OP32(hash32, *(UINT*)p&0xFFFF);
0189
0190 typedef unsigned int UINT;
0191 typedef unsigned int DWORD;
0192
0193 /*
0194 Enter-the-BESTer or an alchemical clash of pairs of primes.
0195
0196 When an x-bit hash where x < 16 and is not a power of 2 is needed,
0197 here comes 'FNV1A_Hash_4_OCTETS': a slightly tuned FNV1A hash for a huge(22,202,980) wordlist of latin-letters-words.
0198
0199 Two improvements for the generic(base) FNV1A hash:
0200 - first, better speed: by reducing 'imul' instructions when string is 4++ chars
0201 - second, better dispersion: by experimenting(superficially-lite test done, so far) with 'FNV1_32_PRIME'
0202
0203 Or more concretely:
0204 - For FNV1_32_INIT = 2166136261
0205 - Giving to 'FNV1_32_PRIME' all primes between 2 and 11987
0206 - Shifting by 16bits instead of 13bits, when 8192 slots are used
0207
0208 C code:
0209 typedef unsigned char u_int8_t;
0210 typedef unsigned long u_int32_t;
0211
0212 #define FNV1_32_INIT ((u_int32_t)2166136261)
0213 #define FNV1_32_PRIME ((u_int32_t)1607)
0214
0215 #define FNV_32A_OP(hash, octet) \
0216     (((u_int32_t)(hash) ^ (u_int8_t)(octet)) * FNV1_32_PRIME)
0217
0218 #define FNV_32A_OP32(hash, octet) \
0219     (((u_int32_t)(hash) ^ (u_int32_t)(octet)) * FNV1_32_PRIME)
0220
0221 0800 // Invoking: FNV1A_Hash_4_OCTETS(wrd, wrdlen>>2) // = 0,1,2,3,4,5,6,7 [1..31]
0222 0801 int FNV1A_Hash_4_OCTETS(char *str, int wrdlen_QUADRUPLETS)
0223 0802 {
0224 0803 u_int32_t hash;
0225 0804 char *p;
0226 0805
0227 0806 hash = FNV1_32_INIT;
0228 0807 p=str;
0229 0808
0230 0809 // The goal of stage #1: to reduce number of 'imul's.
0231 0810
0232 0811 // Stage #1:
0233 0812 for (; wrdlen_QUADRUPLETS != 0; --wrdlen_QUADRUPLETS) {
0234 0813     hash = FNV_32A_OP32(hash, (unsigned long)*(long *)p); // mov edi, DWORD PTR [eax]
0235 0814     p=p+4; // add eax, 4
0236 0815 }
0237 0816
0238 0817 // Stage #2:
0239 0818 for (; *p; ++p) {
0240 0819     hash = FNV_32A_OP(hash, *p); // mov dl, BYTE PTR [eax]
0241 0820 }
0242 0821
0243 0822   //return ((hash>>13) ^ hash) & 8191; // (((u_int32_t)1<<(x))-1) where x=13
0244 0823   return ((hash>>16) ^ hash) & 8191; // 00..8191 i.e. 2^13=8192
0245 0824 }
0246
0247 Assembler code:
0248 _FNV1A_Hash_4_OCTETS PROC NEAR
0249 ; Line 812
0250 mov  edx, DWORD PTR _wrdlen_QUADRUPLETS$[esp-4]
0251 test edx, edx
0252 mov  eax, DWORD PTR _str$[esp-4]
0253 push esi
0254 mov  esi, DWORD PTR _FNV1_32_PRIME
0255 mov  ecx, -2128831035
0256 je   SHORT $L1612
0257 push edi
0258 npad 7
0259 $L1610:
0260 ; Line 813
```



```
0261 mov  edi, DWORD PTR [eax]
0262 xor  edi, ecx
0263 imul edi, esi
0264 ; Line 814
0265 add  eax, 4
0266 dec  edx
0267 mov  ecx, edi
0268 jne  SHORT $L1610
0269 pop  edi
0270 $L1612:
0271 ; Line 818
0272 mov  dl, BYTE PTR [eax]
0273 test dl, dl
0274 je   SHORT $L1619
0275 $L1617:
0276 ; Line 819
0277 movzx        edx, dl
0278 xor  edx, ecx
0279 imul edx, esi
0280 inc  eax
0281 mov  ecx, edx
0282 mov  dl, BYTE PTR [eax]
0283 test dl, dl
0284 jne  SHORT $L1617
0285 $L1619:
0286 ; Line 823
0287 mov  eax, ecx
0288 shr  eax, 16
0289 xor  eax, ecx
0290 and  eax, 8191
0291 pop  esi
0292 ; Line 824
0293 ret  0
0294 _FNV1A_Hash_4_OCTETS ENDP
0295
0296
0297 So, 'FNV1A_Hash_4_OCTETS' calculates faster and gives better distribution(3549448 for 1607), which is 0.6% better(less collisions), than
     generic 'FNV1A_Hash' with 3527916.
0298
0299 FNV proves to be great, dealing with 4x8bits(four octets) at once doesn't hurt distribution at all, I was amazed by consistency(stable
     behaviour) of 'FNV1A_Hash_4_OCTETS'.
0300
0301 I want to make a total clash of all possible pairs 'FNV1_32_INIT' & 'FNV1_32_PRIME' in order to lessen even a few thousand collisions.
0302 This is critical for speed performance e.g. when 30,974,750,142 words, the case of wikipedia-en-html.tar, must be hashed.
0303 The current obstacle is needed-time: each filling (26 slots x 31 sub-slots x 8192 sub-sub-slots) executes in 32-36 seconds for each pair.
0304 Such an easy task, but I can't see how to get done, it is not hard but slow even with 15 times faster testbed.
0305
0306 Between 1..1166136247 there are 58,834,113 primes (inclusive).
0307 Between 1..16777619 there are 1,077,891 primes (inclusive).
0308 Or 58834113*1077891 = 63,416,760,895,683 pairs or 2,010,932 years needed at one-pair-per-second rate.
0309
0310 Finding THE best pair in my opinion is a total alchemy, due to the very nature of hashing: which is mainly alchemical and partly scientific.
0311 Since the magnum corpus of words is static-enough, THE pair is worthy to be found.
0312
0313 It doesn't take a think-tank to see the superiority of FNV, Fowler/Noll/Vo did reveal a thing of beauty.
0314
0315 Performance of 'FNV1A_Hash_4_OCTETS': 10236 words/clock or 105 MB/s|3,549,448 used slots (best)
0316
0317 CASE #1: with 'if (strlen(backup[j]) != 0)' before each execution
0318 Performance of 'KuxHash3plus' aka '2in1': 8076 words/clock or 82 MB/s|3,410,463 used slots (worst)
0319 Performance of 'FNV1A_Hash': 8079 words/clock or 83 MB/s|3,527,916 used slots
0320 Performance of 'FNV1A_Hash_SHIFTless_XORless': 8109 words/clock or 83 MB/s|3,540,323 used slots
0321
0322 CASE #2: without 'if (strlen(backup[j]) != 0)' before each execution
0323 Performance of 'KuxHash3plus' aka '2in1': 11673 words/clock or 119 MB/s|3,410,463 used slots (worst)
0324 Performance of 'FNV1A_Hash': 11558 words/clock or 118 MB/s|3,527,916 used slots
0325 Performance of 'FNV1A_Hash_SHIFTless_XORless': 11570 words/clock or 118 MB/s|3,540,323 used slots
0326
0327 Note:
0328 The 'strlen' overhead(CASE #1) is necessary due to priorly(before hash invocation) needed len-of-string for 'FNV1A_Hash_4_OCTETS'.
0329 Almost always, that is the case, since parsing of incoming text must know length of words/lines/files.
0330 In case of not knowing this length: ((119-105)/105)*100% = 13% degradation is unacceptable.
0331 The 'strlen' is an awful brake.
0332 Also whether the code overhead(one additional cycle) of 'FNV1A_Hash_4_OCTETS' is so successful(as a trade-off) or the testbed is deceiving I
     do not know, here I am not so sure regardless of notorious delays caused by 'imul' and 'div' instructions.
0333 */
0334
0335 /*
0336 FNV1_32_PRIME: //?: 16777619
0337
0338 Above Binary-Search-Tree with MaxPEAK = 61 has NODEs = 61 and LEAFs = 1
0339 Words per second performance: 1,046,822w/s
0340 Input File with a list of TEXTual Files: Leprechaun_vs_Wikipedia_en-WORDS.lst
0341 Size of all TEXTual Files: 146,973,879
0342 Word count: 12,561,874 of them 12,561,874 distinct
0343 Number Of Trees(GREATER THE BETTER): 2775839
0344
0345 Above Binary-Search-Tree with MaxPEAK = 39 has NODEs = 72 and LEAFs = 15
```

```
Words per second performance: 1,356,588W/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 Trees(GREATER THE BETTER): 3539690

FNV1_32_PRIME: //3549448: 1607

Above Binary-Search-Tree with MaxPEAK = 61 has NODEs = 61 and LEAFs = 1
Words per second performance: 1,046,822W/s
Input File with a list of TEXTual Files: Leprechaun_vs_Wikipedia_en-WORDS.lst
Size of all TEXTual Files: 146,973,879
Word count: 12,561,874 of them 12,561,874 distinct
Number Of Trees(GREATER THE BETTER): 2783970

Above Binary-Search-Tree with MaxPEAK = 38 has NODEs = 50 and LEAFs = 11
Words per second performance: 1,410,851W/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 Trees(GREATER THE BETTER): 3549395

FNV1_32_PRIME: //3550132: 175757909

Above Binary-Search-Tree with MaxPEAK = 60 has NODEs = 60 and LEAFs = 1
Words per second performance: 966,298W/s
Input File with a list of TEXTual Files: Leprechaun_vs_Wikipedia_en-WORDS.lst
Size of all TEXTual Files: 146,973,879
Word count: 12,561,874 of them 12,561,874 distinct
Number Of Trees(GREATER THE BETTER): 2784479

Above Binary-Search-Tree with MaxPEAK = 39 has NODEs = 64 and LEAFs = 12
Words per second performance: 1,410,851W/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 Trees(GREATER THE BETTER): 3550115

FNV1_32_PRIME: //3550687: 201887489

Above Binary-Search-Tree with MaxPEAK = 60 has NODEs = 60 and LEAFs = 1
Words per second performance: 966,298W/s
Input File with a list of TEXTual Files: Leprechaun_vs_Wikipedia_en-WORDS.lst
Size of all TEXTual Files: 146,973,879
Word count: 12,561,874 of them 12,561,874 distinct
Number Of Trees(GREATER THE BETTER): 2784377

Above Binary-Search-Tree with MaxPEAK = 40 has NODEs = 55 and LEAFs = 11
Words per second performance: 1,356,588W/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 Trees(GREATER THE BETTER): 3550528

FNV1_32_PRIME: //3550733: 172783361

Above Binary-Search-Tree with MaxPEAK = 59 has NODEs = 59 and LEAFs = 1
Words per second performance: 1,046,822W/s
Input File with a list of TEXTual Files: Leprechaun_vs_Wikipedia_en-WORDS.lst
Size of all TEXTual Files: 146,973,879
Word count: 12,561,874 of them 12,561,874 distinct
Number Of Trees(GREATER THE BETTER): 2786362

Above Binary-Search-Tree with MaxPEAK = 38 has NODEs = 70 and LEAFs = 17
Words per second performance: 1,410,851W/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 Trees(GREATER THE BETTER): 3550746

FNV1_32_PRIME: //3550929: 204312319

Above Binary-Search-Tree with MaxPEAK = 61 has NODEs = 61 and LEAFs = 1
Words per second performance: 966,298W/s
Input File with a list of TEXTual Files: Leprechaun_vs_Wikipedia_en-WORDS.lst
Size of all TEXTual Files: 146,973,879
Word count: 12,561,874 of them 12,561,874 distinct
Number Of Trees(GREATER THE BETTER): 2785581

Above Binary-Search-Tree with MaxPEAK = 37 has NODEs = 55 and LEAFs = 12
Words per second performance: 1,356,588W/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 Trees(GREATER THE BETTER): 3550886

Leprechaun_Microsoft.exe: FNV1_32_PRIME: //3551736: 107712257

```



```
Above Binary-Search-Tree with MaxPEAK = 61 has NODEs = 61 and LEAFs = 1
Words per second performance: 1,046,822W/s
Input File with a list of TEXTual Files: Leprechaun_vs_Wikipedia_en-WORDS.lst
Size of all TEXTual Files: 146,973,879
Word count: 12,561,874 of them 12,561,874 distinct
Number Of Trees(GREATER THE BETTER): 2786515

Above Binary-Search-Tree with MaxPEAK = 36 has NODEs = 64 and LEAFs = 15
Words per second performance: 1,356,588W/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 Trees(GREATER THE BETTER): 3551744

Leprechaun_Intel.exe: FNV1_32_PRIME: //3551736: 107712257

Above Binary-Search-Tree with MaxPEAK = 61 has NODEs = 61 and LEAFs = 1
Words per second performance: 1,256,187W/s
Input File with a list of TEXTual Files: Leprechaun_vs_Wikipedia_en-WORDS.lst
Size of all TEXTual Files: 146,973,879
Word count: 12,561,874 of them 12,561,874 distinct
Number Of Trees(GREATER THE BETTER): 2786515

Above Binary-Search-Tree with MaxPEAK = 36 has NODEs = 64 and LEAFs = 15
Words per second performance: 1,603,240W/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 Trees(GREATER THE BETTER): 3551744

Wow: 1,603,240W/s vs 1,356,588W/s respectively Leprechaun_Intel.exe vs Leprechaun_Microsoft.exe, i.e. 18% betterment, no joke!

Alchemical search for best PRIME-PAIR revision uses next line:
Slot = FNV1A_Hash_4_OCTETS(wrd, wrdlen>>2)<<2; //13++++
This revision uses next lines:
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+++++

! 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.

Leprechaun.exe: FNV1_32_PRIME: //3551961: 428904191

Above Binary-Search-Tree with MaxPEAK = 60 has NODEs = 60 and LEAFs = 1
Words per second performance: 966,298W/s
Input File with a list of TEXTual Files: Leprechaun_vs_Wikipedia_en-WORDS.lst
Size of all TEXTual Files: 146,973,879
Word count: 12,561,874 of them 12,561,874 distinct
Number Of Trees(GREATER THE BETTER): 2786383

Above Binary-Search-Tree with MaxPEAK = 39 has NODEs = 71 and LEAFs = 16
Words per second performance: 1,410,851W/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 Trees(GREATER THE BETTER): 3551503

Leprechaun.exe: FNV1_32_PRIME: //3552103: 588411137

Above Binary-Search-Tree with MaxPEAK = 6 has NODEs = 6 and LEAFs = 1
Size of all TEXTual Files: 4,067,439
Word count: 358,798 of them 351,116 distinct
Number Of Trees(GREATER THE BETTER): 310622
Total Number of LEAFs in Binary-Search-Trees(GREATER THE BETTER): 310,622

Above Binary-Search-Tree with MaxPEAK = 60 has NODEs = 60 and LEAFs = 1
Size of all TEXTual Files: 146,973,879
Word count: 12,561,874 of them 12,561,874 distinct
Number Of Trees(GREATER THE BETTER): 2786485
Total Number of LEAFs in Binary-Search-Trees(GREATER THE BETTER): 2,786,485

Above Binary-Search-Tree with MaxPEAK = 39 has NODEs = 62 and LEAFs = 15
Size of all TEXTual Files: 415,982,896
Word count: 35,271,297 of them 22,202,980 distinct
Number Of Trees(GREATER THE BETTER): 3551956
Total Number of LEAFs in Binary-Search-Trees(GREATER THE BETTER): 8,072,131

Leprechaun.exe: FNV1_32_PRIME: //3552039: 602173697 !!!GOODEST so far!!!

Above Binary-Search-Tree with MaxPEAK = 6 has NODEs = 6 and LEAFs = 1
Size of all TEXTual Files: 4,067,439
Word count: 358,798 of them 351,116 distinct
Number Of Trees(GREATER THE BETTER): 310948
Total Number of LEAFs in Binary-Search-Trees(GREATER THE BETTER): 310,948

Above Binary-Search-Tree with MaxPEAK = 63 has NODEs = 63 and LEAFs = 1
```

```
0521 Size of all TEXTual Files: 146,973,879
0522 Word count: 12,561,874 of them 12,561,874 distinct
0523 Number Of Trees(GREATER THE BETTER): 2786806
0524 Total Number of LEAFs in Binary-Search-Trees(GREATER THE BETTER): 2,786,806
0525
0526 Above Binary-Search-Tree with MaxPEAK = 36 has NODEs = 52 and LEAFs = 9
0527 Input File with a list of TEXTual Files: Leprechaun_vs_Wikipedia_LATIN-WORDS.lst
0528 Size of all TEXTual Files: 415,982,896
0529 Word count: 35,271,297 of them 22,202,980 distinct
0530 Number Of Trees(GREATER THE BETTER): 3552296
0531 Total Number of LEAFs in Binary-Search-Trees(GREATER THE BETTER): 8,072,899
0532
0533 Between 1 and 602392027 at step 100 following FNV1_32_PRIMEs(for FNV1_32_INIT=2166136261) give(FNV1A_Hash_4_OCTETS) dispersion:
0534 3550022: 423779327
0535 3550028: 513793537
0536 3550053: 434840321
0537 3550067: 437062229
0538 3550080: 420344321
0539 3550090: 304777471
0540 3550097: 496547839
0541 3550129: 390809599
0542 3550132: 175757909
0543 3550163: 353712127
0544 3550231: 334434817
0545 3550237: 272789761
0546 3550247: 590341121
0547 3550255: 358814207
0548 3550277: 437182721
0549 3550326: 521795327
0550 3550347: 311867393
0551 3550447: 456137729
0552 3550458: 418208767
0553 3550516: 602048767
0554 3550525: 513597697
0555 3550526: 347283199
0556 3550528: 598773503
0557 3550592: 598139137
0558 3550598: 242448127
0559 3550611: 571481087
0560 3550628: 457012993
0561 3550664: 482822143
0562 3550666: 249098753
0563 3550687: 201887489
0564 3550702: 489976063
0565 3550710: 272961023
0566 3550733: 172783361
0567 3550734: 431562497
0568 3550929: 204312319
0569 3550984: 562853633
0570 3550991: 551362303
0571 3551359: 332820737
0572 3551484: 354126079
0573 3551514: 407138561
0574 3551523: 442058753
0575 3551701: 449230849
0576 3551736: 107712257
0577 3551961: 428904191
0578 3552039: 602173697
0579 3552103: 588411137
0580 */
0581
0582 // Windows: ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
0583 //  _CRTIMP size_t __cdecl fread(void *, size_t, size_t, FILE *);
0584 //  _CRTIMP size_t __cdecl fwrite(const void *, size_t, size_t, FILE *);
0585 //  _CRTIMP int __cdecl fgetpos(FILE *, fpos_t *);
0586 //  _CRTIMP int __cdecl fsetpos(FILE *, const fpos_t *);
0587
0588 //  _CRTIMP __int64 __cdecl _lseeki64(int, __int64, int);
0589 //  _CRTIMP __int64 __cdecl _telli64(int);
0590 //  _CRTIMP __int64 __cdecl _filelengthi64(int);
0591 //  above 3 are in 'io.h'
0592
0593 //  _CRTIMP int __cdecl fseek(FILE *, long, int);
0594 //  _CRTIMP long __cdecl ftell(FILE *);
0595 //  _CRTIMP int __cdecl fclose(FILE *);
0596
0597 //  #ifndef _SIZE_T_DEFINED
0598 //  #ifdef  _WIN64
0599 //  typedef unsigned __int64    size_t;
0600 //  #else
0601 //  typedef _W64 unsigned int   size_t;
0602 //  #endif
0603 //  #define _SIZE_T_DEFINED
0604 //  #endif
0605
0606 //  typedef __int64 fpos_t;
0607
0608 // Linux: ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```



```
0609 //  size_t fread (void *data, size_t size, size_t count, FILE *stream)
0610 //  size_t fwrite (const void *data, size_t size, size_t count, FILE *stream)
0611 //  int fgetpos (FILE *stream, fpos_t *position)
0612 //  int fsetpos (FILE *stream, const fpos_t *position)
0613
0614 //  FILE * fopen64 (const char *filename, const char *opentype)
0615 //  int fseeko64 (FILE *stream, off64_t offset, int whence)
0616 //  off64_t ftello64 (FILE *stream)
0617 //  int fclose (FILE *stream)
0618
0619 //  off_t lseek (int filedes, off_t offset, int whence)
0620 //  above 1 is in 'unistd.h'
0621
0622
0623 // ============= MUST work both for Windows and Linux =============
0624 #define _WIN32_ENVIRONMENT_
0625 //#define _POSIX_ENVIRONMENT_
0626
0627
0628 #ifndef NULL
0629 #ifdef __cplusplus
0630 #define NULL 0
0631 #else
0632 #define NULL ((void*)0)
0633 #endif
0634 #endif
0635
0636 // To do #1: Put this 31 in MAXWl: 'int MAXWl = 31;'
0637 // To do #2: No need of flushing unsorted words to file: make backup[] array
0638 //           instead of writing. And mostly sort 26 times!
0639 // HEAVY BUG in r.7: unsigned long Hill(unsigned long n)
0640 //                   is NOT identical with
0641 //                   unsigned long GRMBLhill[32]; // 00 not used, only 01..31
0642 //                   BECAUSE DUMBEST DUMB Array GRMBLhill expects 'int' not
0643 //                   'unsigned long' !!!
0644
0645 #include <stdio.h>
0646 #include <ctype.h>
0647 #include <time.h>
0648 #if defined(_WIN32_ENVIRONMENT_)
0649 #include <io.h> // needed for Windows' 'lseeki64' and 'telli64'
0650 //Above line must be commented in order to compile with Intel C compiler: an error "can't find io.h" occurs.
0651 #else
0652 #endif /* defined(_WIN32_ENVIRONMENT_)  */
0653
0654 typedef unsigned char char_t;
0655 typedef char_t *string;
0656
0657 clock_t clocks1, clocks2;
0658 int Bozan;
0659
0660 typedef unsigned char u_int8_t; //FNV only
0661 typedef unsigned long u_int32_t; //FNV only
0662 typedef unsigned long long u_int64_t; //FNV only
0663
0664 // SINHA fragment[
0665
0666 #define swapKAZE(a, b) { t = *(a); *(a) = *(b); *(b) = t; }
0667
0668 static void InsertSortKAZE(string *a, int n, int d) //void inssort(unsigned char **a, int n, int d)
0669 {   string *pi, *pj, s, t;                          //unsigned char **pi, **pj, *s, *t;
0670     for (pi = a + 1; --n > 0; pi++)
0671         for (pj = pi; pj > a; pj--) {
0672             /* Inline strcmp: break if *(pj-1) <= *pj */
0673             for (s=*(pj-1)+d, t=*pj+d; *s==*t && *s!=0; s++, t++)
0674                 ;
0675             if (*s <= *t)
0676                 break;
0677             swapKAZE(pj, pj-1);
0678     }
0679 }
0680
0681 //int cmpit(unsigned char **h1, unsigned char **h2)
0682 //{
0683 //    return( scmp(*h1, *h2) );
0684 //}
0685
0686 //int scmp( unsigned char *s1, unsigned char *s2 )
0687 //{
0688 //    while( *s1 != '\0' && *s1 == *s2 )
0689 //    {
0690 //        s1++;
0691 //        s2++;
0692 //    }
0693 //    return( *s1-*s2 );
0694 //}
0695
0696 //static void simplesort(string a[], int n, int b)
```

```
0697 //{
0698 //   int i, j;
0699 //   string tmp;
0700 //
0701 //   for (i = 1; i < n; i++)
0702 //       for (j = i; j > 0 && scmp(a[j-1]+b, a[j]+b) > 0; j--)
0703 //           { tmp = a[j]; a[j] = a[j-1]; a[j-1] = tmp; }
0704 //}
0705
0706 // SINHA fragment]
0707
0708 // mkqsort.c BEGIN ****************************************************************
0709 /*
0710    Multikey quicksort, a radix sort algorithm for arrays of character
0711    strings by Bentley and Sedgewick.
0712
0713    J. Bentley and R. Sedgewick. Fast algorithms for sorting and
0714    searching strings. In Proceedings of 8th Annual ACM-SIAM Symposium
0715    on Discrete Algorithms, 1997.
0716
0717    http://www.CS.Princeton.EDU/~rs/strings/index.html
0718
0719    The code presented in this file has been tested with care but is
0720    not guaranteed for any purpose. The writer does not offer any
0721    warranties nor does he accept any liabilities with respect to
0722    the code.
0723
0724    Ranjan Sinha, 1 jan 2003.
0725
0726    School of Computer Science and Information Technology,
0727    RMIT University, Melbourne, Australia
0728    rsinha@cs.rmit.edu.au
0729
0730 */
0731
0732 //#include "sortstring.h"
0733
0734 /* MULTIKEY QUICKSORT */
0735
0736 #ifndef min
0737 #define min(a, b) ((a)<=(b) ? (a) : (b))
0738 #endif
0739
0740
0741 // ------------------------------ BTREE [
0742 #define false -1
0743 #define true 0
0744
0745 struct nodeBTREE {
0746 int data;
0747 struct nodeBTREE* left;
0748 struct nodeBTREE* right;
0749 };
0750
0751 // ------------------------------ BTREE ]
0752
0753
0754 /* ssort2 -- Faster Version of Multikey Quicksort */
0755
0756 void vecswap2(unsigned char **a, unsigned char **b, int n)
0757 {   while (n-- > 0) {
0758         unsigned char *t = *a;
0759         *a++ = *b;
0760         *b++ = t;
0761     }
0762 }
0763
0764 #define swap2(a, b) { t = *(a); *(a) = *(b); *(b) = t; }
0765 #define ptr2char(i) (*(*(i) + depth))
0766
0767 unsigned char **med3func(unsigned char **a, unsigned char **b, unsigned char **c, int depth)
0768 {   int va, vb, vc;
0769     if ((va=ptr2char(a)) == (vb=ptr2char(b)))
0770         return a;
0771     if ((vc=ptr2char(c)) == va || vc == vb)
0772         return c;
0773     return va < vb ?
0774           (vb < vc ? b : (va < vc ? c : a ) )
0775         : (vb > vc ? b : (va < vc ? a : c ) );
0776 }
0777 #define med3(a, b, c) med3func(a, b, c, depth)
0778
0779 void inssort(unsigned char **a, int n, int d)
0780 {   unsigned char **pi, **pj, *s, *t;
0781     for (pi = a + 1; --n > 0; pi++)
0782         for (pj = pi; pj > a; pj--) {
0783             /* Inline strcmp: break if *(pj-1) <= *pj */
0784             for (s=*(pj-1)+d, t=*pj+d; *s==*t && *s!=0; s++, t++)
```



```
0785                 ;
0786             if (*s <= *t)
0787                 break;
0788             swap2(pj, pj-1);
0789     }
0790 }
0791
0792 void mkqsort(unsigned char **a, int n, int depth)
0793 {   int d, r, partval;
0794     unsigned char **pa, **pb, **pc, **pd, **pl, **pm, **pn, *t;
0795     if (n < 20) {
0796         inssort(a, n, depth);
0797         return;
0798     }
0799     pl = a;
0800     pm = a + (n/2);
0801     pn = a + (n-1);
0802     if (n > 30) { /* On big arrays, pseudomedian of 9 */
0803         d = (n/8);
0804         pl = med3(pl, pl+d, pl+2*d);
0805         pm = med3(pm-d, pm, pm+d);
0806         pn = med3(pn-2*d, pn-d, pn);
0807     }
0808     pm = med3(pl, pm, pn);
0809     swap2(a, pm);
0810     partval = ptr2char(a);
0811     pa = pb = a + 1;
0812     pc = pd = a + n-1;
0813     for (;;) {
0814         while (pb <= pc && (r = ptr2char(pb)-partval) <= 0) {
0815             if (r == 0) { swap2(pa, pb); pa++; }
0816             pb++;
0817         }
0818         while (pb <= pc && (r = ptr2char(pc)-partval) >= 0) {
0819             if (r == 0) { swap2(pc, pd); pd--; }
0820             pc--;
0821        }
0822         if (pb > pc) break;
0823         swap2(pb, pc);
0824         pb++;
0825         pc--;
0826     }
0827     pn = a + n;
0828     r = min(pa-a, pb-pa);    vecswap2(a,  pb-r, r);
0829     r = min(pd-pc, pn-pd-1); vecswap2(pb, pn-r, r);
0830     if ((r = pb-pa) > 1)
0831         mkqsort(a, r, depth);
0832     if (ptr2char(a + r) != 0)
0833         mkqsort(a + r, pa-a + pn-pd-1, depth+1);
0834     if ((r = pd-pc) > 1)
0835         mkqsort(a + n-r, r, depth);
0836 }
0837
0838 void mkqsort_main(unsigned char **a, int n) { mkqsort(a, n, 0); }
0839 // mkqsort.c END ******************************************************************
0840
0841 // Why Sinha uses int instead of long??!!
0842 static int readlines(char *file_name, string **lines)
0843 {
0844    int nlines = 0;
0845    size_t size;
0846    FILE *in_file;
0847    string basep, cur, next;
0848    string *ASbackup;
0849
0850    if (!(in_file = fopen(file_name, "rb"))) {
0851       printf( "Leprechaun: Can't open file %s \n", file_name );
0852       exit(-1);
0853    }
0854    fseek(in_file, 0, SEEK_END);
0855    size = ftell(in_file);
0856    fseek(in_file, 0, SEEK_SET);
0857    if (!(basep = (string) malloc(size*sizeof(char_t)))) return -1;
0858    printf( "Allocated memory for words in MB: %lu\n", ((size*sizeof(char_t))>>20)+1 );
0859    if (fread(basep, 1, size, in_file) < size) {
0860       printf( "Leprechaun: Can't read file %s \n", file_name );
0861       exit(-1);
0862    }
0863    fclose(in_file);
0864
0865 // GET nlines:
0866    cur = basep;
0867    while (cur < basep + size) {
0868       next = cur;
0869       while ((next < basep + size) && (*next != '\n')) {next++;}
0870       *--next = '\0';          // This is ala DOS i.e. Windows
0871                                // 1310 not 10(\n=10)
0872       cur = next + 2;
```

```
      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?!
```

```
1049   }
1050   return h; // 00..255 i.e. 2^8=256
1051 }
1052
1053 int KuxHash2(char *str)
1054 { int h = 0;
1055   unsigned long h2 = 0; // must be long: 31*'z'=31*122
1056   int max31 = 0;
1057   while (str[max31])
1058   { h = h ^ str[max31]; // 00..255 i.e. 2^8=256
1059     //h2 = h2 + str[max31++];       // [113s]
1060     h2 = h2 + max31 * str[max31++];
1061   }
1062   h=h<<4; // 00..15 i.e. 2^4=16
1063   //h = h|( str[0] ^ str[max31-1] ); // [111s] a..z: each XOR each gives 00..31
1064   h = h|( h2%((1<<4)-1) );
1065   return h; // 00..4095 i.e. 2^12=4096
1066 }
1067
1068 //OSHO test - Attempts to Find/Put a WORD into linked list count: 32,011,937
1069 int KuxHash3(char *str)
1070 { int h = 0;
1071   unsigned long h2 = 0; // must be long: 31*'z'=31*122
1072   int max31 = 0;
1073   while (str[max31])
1074   { h = h ^ str[max31]; // 00..255 i.e. 2^8=256
1075     //h2 = h2 + str[max31++];       // [113s]
1076     h2 = h2 + str[max31++] * (max31+1);
1077   }
1078 // Result is: 7bits in 'h' and 32bits in 'h2'.
1079
1080                         //printf("%s:\n ",str);
1081                         //printf("%d ",h);
1082   h=h<<6; // 00..15 i.e. 00-05+7bits=13bits
1083                         //printf("%d ",h);
1084                         //printf("%d ",h2);
1085   //h = h|( str[0] ^ str[max31-1] ); // [111s] a..z: each XOR each gives 00..31
1086   h = h|( h2%((1<<6)-1) ); // 64-1=63=9*7; 61 is prime
1087                         //printf("%d \n",h);
1088   return h; // 00..8191 i.e. 2^13=8192
1089 }
1090
1091 //OSHO test - Attempts to Find/Put a WORD into a linked list count: 31,927,285
1092 int KuxHash3plus(char *str)
1093 { int h = 0;
1094   unsigned long h2 = 0; // must be long: 31*'z'=31*122
1095   int max31 = 0;
1096   while (str[max31])
1097   { h = h ^ str[max31]; // 00..255 i.e. 2^8=256
1098     //h2 = h2 + str[max31++];       // [113s]
1099     h2 = h2 + str[max31++] * (max31+1);
1100   }
1101 // Result is: 7bits in 'h' and 32bits in 'h2'.
1102
1103                         //printf("%s:\n ",str);
1104                         //printf("%d ",h);
1105 // a in ASCII is 097 = 0110 0001
1106 // z in ASCII is 122 = 0111 1010
1107 // Above two lines show that bits 8-7-6 are always 0-1-1 so need for low 5 bits.
1108   //h=h<<8; // 00..15 i.e. 5bits + 00-07bits=13bits
1109                         //printf("%d ",h);
1110                         //printf("%d ",h2);
1111   //h = h|( str[0] ^ str[max31-1] ); // [111s] a..z: each XOR each gives 00..31
1112   h = (( h<<8 )|( h2%(251) ))&8191; // 251 prime
1113                         //printf("%d \n",h);
1114   return h; // 00..8191 i.e. 2^13=8192
1115 }
1116
1117 /*
1118 PUBLIC        _KuxHash3plus
1119 ; Function compile flags: /Ogty
1120 _TEXT         SEGMENT
1121 _str$ = 8                                                       ; size = 4
1122 _KuxHash3plus PROC NEAR
1123 ; Line 511
1124 mov  ecx, DWORD PTR _str$[esp-4]
1125 mov  dl, BYTE PTR [ecx]
1126 push esi
1127 xor  esi, esi
1128 xor  eax, eax
1129 test dl, dl
1130 je   SHORT $L1561
1131 push ebx
1132 push edi
1133 mov  edi, 1
1134 sub  edi, ecx
1135 npad 8
1136 $L1560:
```



```
1137 ; Line 512
1138 movsx        edx, BYTE PTR [ecx]
1139 ; Line 514
1140 lea  ebx, DWORD PTR [edi+ecx]
1141 imul ebx, edx
1142 xor  esi, edx
1143 mov  dl, BYTE PTR [ecx+1]
1144 add  eax, ebx
1145 inc  ecx
1146 test dl, dl
1147 jne  SHORT $L1560
1148 pop  edi
1149 pop  ebx
1150 $L1561:
1151 ; Line 527
1152 xor  edx, edx
1153 mov  ecx, 251                              ; 000000fbH
1154 div  ecx
1155 shl  esi, 8
1156 mov  eax, edx
1157 ; Line 529
1158 or   eax, esi
1159 and  eax, 8191                             ; 00001fffH
1160 pop  esi
1161 ; Line 530
1162 ret  0
1163 _KuxHash3plus ENDP
1164 _TEXT         ENDS
1165 */
1166
1167 //OSHO test - Attempts to Find/Put a WORD into a linked list count: 32,021,975
1168 int KuxHash4(char *str)
1169 {
1170   int h2 = 0;
1171     for (; *str != 0; str++) {
1172       //h2 = (127*h2 + *str) % (8192-1); // 2^13-1 = 8191 is Mersenne prime
1173       h2 = ((h2<<7) + *str) % (8192-1); // 2^13-1 = 8191 is Mersenne prime
1174     }
1175
1176   return h2; // 00..8191 i.e. 2^13=8192
1177 }
1178 /*
1179 int hash(char *v, int M)
1180   { int h = 0, a = 127;
1181     for (; *v != 0; v++)
1182       h = (a*h + *v) % M;
1183     return h;
1184   }
1185
1186 int hashU(char *v, int M)
1187   { int h, a = 31415, b = 27183;
1188     for (h = 0; *v != 0; v++, a = a*b % (M-1))
1189         h = (a*h + *v) % M;
1190     return (h < 0) ? (h + M) : h;
1191   }
1192 */
1193
1194 // Kaze: My appreciation of FNV is far beyound C code optimization, it is alchemical, and why not, magical.
1195
1196 /*
1197 FNV hash history
1198     The basis of the FNV hash algorithm was taken from an idea sent as reviewer comments to the IEEE POSIX P1003.2 committee
1199 by Glenn Fowler and Phong Vo back in 1991. In a subsequent ballot round: Landon Curt Noll improved on their algorithm.
1200 Some people tried this hash and found that it worked rather well. In an EMail message to Landon, they named it
1201 the   Fowler/Noll/Vo'' or FNV hash.
1202     FNV hashes are designed to be fast while maintaining a low collision rate. The FNV speed allows one to quickly hash
1203 lots of data while maintaining a reasonable collision rate. The high dispersion of the FNV hashes makes them well suited
1204 for hashing nearly identical strings such as URLs, hostnames, filenames, text, IP addresses, etc.
1205 */
1206
1207 /* NOTE: u_int64_t is a 64 bit unsigned type */
1208 /* NOTE: u_int32_t is a 32 bit unsigned type */
1209 /* NOTE: u_int16_t is a 16 bit unsigned type */
1210 /* NOTE: u_int8_t is a 8 bit unsigned type */
1211
1212 //typedef unsigned char u_int8_t; //FNV only
1213 //typedef unsigned long u_int32_t; //FNV only
1214 //typedef unsigned long long u_int64_t; //FNV only
1215
1216 // 32 bit FNV_prime = 2^24 + 2^8 + 0x93 = 16777619
1217 // 64 bit FNV_prime = 2^40 + 2^8 + 0xb3 = 1099511628211
1218
1219 // 32 bit offset_basis = 2166136261
1220 // 64 bit offset_basis = 14695981039346656037
1221
1222 #define FNV1_64_INIT ((u_int64_t)14695981039346656037)
1223 #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) // wrdlen>>0=wrdlen
int FNV1A_Hash_Granularity(char *str, int wrdlen_granulated, int Granularity)
{
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;

hash32 = FNV1_32_INIT;
p=str;

for(; wrdlen >= 4; wrdlen -= 4, p += 4) {
hash32 = FNV_32A_OP32(hash32, (UINT)*(UINT *)p);
}
if (wrdlen & -2) {
    hash32 = FNV_32A_OP32(hash32, *(UINT*)p&0xFFFF);
    p++;p++;
}
if (wrdlen & 1)
    hash32 = FNV_32A_OP(hash32, *p);

return hash32 ^ (hash32 >> 16);
}
*/

/*
Results for 'FNV1A_Hash_Jester':
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): 3537352
Forest population(Hash Function Quality regarding Collisions i.e. Hash Table Utilization): 53%
```

```
Number Of Hash Collisions(Distinct WORDs - Number Of Trees): 18665628
Maximum Attempts to Find/Put a WORD into a Binary-Search-Tree: '37'
Total Attempts to Find/Put WORDs into Binary-Search-Trees: 117,243,563
Total Number of LEAFs in Binary-Search-Trees(GREATER THE BETTER): 8,063,361
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 2) with MaxNODEs = 87 has PEAK = 27 and LEAFs = 23
Binary-Search-Tree(1st out of 1) with MaxPEAK = '37' has NODEs = 66 and LEAFs = 18
Binary-Search-Tree(1st out of 3) with MaxLEAFs = 27 has NODEs = 84 and PEAK = 27
*/
UINT FNV1A_Hash_Jester(const char *str, unsigned int wrdlen)
{
const UINT PRIME = 709607;
UINT hash32 = 2166136261;
const char *p = str;

// Idea comes from Igor Pavlov's 7zCRC, thanks.
/*
for(; wrdlen && ((unsigned)(ptrdiff_t)p&3); wrdlen -= 1, p++) {
    hash32 = (hash32 ^ *p) * PRIME;
}
*/
for(; wrdlen >= 2*sizeof(DWORD); wrdlen -= 2*sizeof(DWORD), p += 2*sizeof(DWORD)) {
    hash32 = (hash32 ^ *(DWORD *)p) * PRIME;
    hash32 = (hash32 ^ *(DWORD *)(p+4)) * PRIME;
}
// Cases: 0,1,2,3,4,5,6,7
if (wrdlen & sizeof(DWORD)) {
    hash32 = (hash32 ^ *(DWORD*)p) * PRIME;
    p += sizeof(DWORD);
}
if (wrdlen & sizeof(WORD)) {
    hash32 = (hash32 ^ *(WORD*)p) * PRIME;
    p += sizeof(WORD);
}
if (wrdlen & 1)
    hash32 = (hash32 ^ *p) * PRIME;

return (hash32 ^ (hash32 >> 16)) & 8191;
}

/*
Results for 'FNV1A_Hash_Jesteress':
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): 3537293
Forest population(Hash Function Quality regarding Collisions i.e. Hash Table Utilization): 53%
Number Of Hash Collisions(Distinct WORDs - Number Of Trees): 18665687
Maximum Attempts to Find/Put a WORD into a Binary-Search-Tree: '40'
Total Attempts to Find/Put WORDs into Binary-Search-Trees: 117,526,680
Total Number of LEAFs in Binary-Search-Trees(GREATER THE BETTER): 8,051,512
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 = 25 and LEAFs = 23
Binary-Search-Tree(1st out of 1) with MaxPEAK = '40' has NODEs = 49 and LEAFs = 8
Binary-Search-Tree(1st out of 1) with MaxLEAFs = 28 has NODEs = 72 and PEAK = 21
*/
#define ROL(x, n) (((x) << (n)) | ((x) >> (32-(n))))
UINT FNV1A_Hash_Jesteress(const char *str, unsigned int wrdlen)
{
const UINT PRIME = 709607;
UINT hash32 = 2166136261;
const char *p = str;

// Idea comes from Igor Pavlov's 7zCRC, thanks.
/*
for(; wrdlen && ((unsigned)(ptrdiff_t)p&3); wrdlen -= 1, p++) {
    hash32 = (hash32 ^ *p) * PRIME;
}
*/
for(; wrdlen >= 2*sizeof(DWORD); wrdlen -= 2*sizeof(DWORD), p += 2*sizeof(DWORD)) {
    hash32 = (hash32 ^ (ROL(*(DWORD *)p,5)^*(DWORD *)(p+4))) * PRIME;
}
// Cases: 0,1,2,3,4,5,6,7
if (wrdlen & sizeof(DWORD)) {
    hash32 = (hash32 ^ *(DWORD*)p) * PRIME;
    p += sizeof(DWORD);
}
if (wrdlen & sizeof(WORD)) {
    hash32 = (hash32 ^ *(WORD*)p) * PRIME;
    p += sizeof(WORD);
}
if (wrdlen & 1)
    hash32 = (hash32 ^ *p) * PRIME;
```



```

return (hash32 ^ (hash32 >> 16)) & 8191;
}


/*
UINT NextPowerOfTwo(UINT x) {
// Henry Warren, "Hacker's Delight", ch. 3.2
x--;
x |= (x >> 1);
x |= (x >> 2);
x |= (x >> 4);
x |= (x >> 8);
x |= (x >> 16);
return x + 1;
}

UINT NextLog2(UINT x) {
// Henry Warren, "Hacker's Delight", ch. 5.3
if(x <= 1) return x;
x--;
UINT n = 0;
UINT y;
y = x >>16; if(y) {n += 16; x = y;}
y = x >> 8; if(y) {n +=  8; x = y;}
y = x >> 4; if(y) {n +=  4; x = y;}
y = x >> 2; if(y) {n +=  2; x = y;}
y = x >> 1; if(y) return n + 2;
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);
}


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 WORDcountAttemptsToPut;
      int Thunderwith;
        unsigned long NumberOfFiles, WORDcountDistinct;
        unsigned long long NumberOfLines;  // rev. 12+
      unsigned long WHOLEletter_BufferSize;
        unsigned long memory_size, LetterBuffer, j, k, LINE10len, wrdlen;
        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;

      const int NumberOfSLOTs = 4096*2; // Since r.12+ in rev.12 it was 4096
  unsigned long StackPtr;
  unsigned long BSTstack [65536*3]; // BST in worst case could become a LL.
unsigned long NumberOfTrees=0, NumberOfHashCollisions=0;
unsigned long iBSTwithMAXpeak, jBSTwithMAXpeak;
  unsigned int PEAKiBST;
  unsigned long BSTsTotalLEAFs=0; // ?! MADHOUSE: if BSTcurrent is not zeroed here then INSANE values of BSTcurrent occur below where 'break' is ?!
  unsigned long BSTwithMAXnode=0, BSTcurrentNode=0; // ?! MADHOUSE: if BSTcurrent is not zeroed here then INSANE values of BSTcurrent occur below where 'break' is ?!
```

```
2277   unsigned long                  BSTcurrentNodeMAXqUANTITY=0; // ?! MADHOUSE: if BSTcurrent is not zeroed here then INSANE values of BSTcurrent occur below where 'break' is ?!
2278   unsigned long BSTwithMAXnodePEAK=1, BSTwithMAXnodeLEAF=1; // ?! MADHOUSE: if BSTcurrent is not zeroed here then INSANE values of BSTcurrent occur below where 'break' is ?!
2279   unsigned long BSTwithMAXpeak=0, BSTcurrentPeak=0; // ?! MADHOUSE: if BSTcurrent is not zeroed here then INSANE values of BSTcurrent occur below where 'break' is ?!
2280   unsigned long                  BSTcurrentPeakMAX=0; // ?! MADHOUSE: if BSTcurrent is not zeroed here then INSANE values of BSTcurrent occur below where 'break' is ?!
2281   unsigned long                  BSTcurrentPeakMAXqUANTITY=0; // ?! MADHOUSE: if BSTcurrent is not zeroed here then INSANE values of BSTcurrent occur below where 'break' is ?!
2282   unsigned long BSTwithMAXpeakNODE=1, BSTwithMAXpeakLEAF=1; // ?! MADHOUSE: if BSTcurrent is not zeroed here then INSANE values of BSTcurrent occur below where 'break' is ?!
2283   unsigned long BSTwithMAXleaf=0, BSTcurrentLeaf=0; // ?! MADHOUSE: if BSTcurrent is not zeroed here then INSANE values of BSTcurrent occur below where 'break' is ?!
2284   unsigned long                  BSTcurrentLeafMAXqUANTITY=0; // ?! MADHOUSE: if BSTcurrent is not zeroed here then INSANE values of BSTcurrent occur below where 'break' is ?!
2285   unsigned long BSTwithMAXleafNODE=1, BSTwithMAXleafPEAK=1; // ?! MADHOUSE: if BSTcurrent is not zeroed here then INSANE values of BSTcurrent occur below where 'break' is ?!
2286
2287       char *pointerflush, *pointerflushUNALIGN, *BufStart, *Flushing;
2288       unsigned long PseudoLinkedPointer, PseudoLinkedPointerNEW, PseudoLinkedPointerROOT, PseudoLinkedPointerNEWold;
2289       unsigned long PseudoLinkedPointerNEWleft, PseudoLinkedPointerNEWright;
2290       unsigned long PseudoLinkedPointerNEWmiddle;
2291       char *bufend[ 806 ]; // 'a'=0, ... 'z'=25 - 26 letters x 31 lengths
2292       long bufNumberOfWords[ 806 ]; // 'a'=0, ... 'z'=25 - 26 letters x 31 lengths
2293       // long bufNoWpS[ 806 ][ 8192 ]; // ?! crashes below when an attempt to use it occur
2294       char wrd[32];  // 0..30, 31 = 0
2295       char wrdUP[32];  // 0..30, 31 = 0
2296       char wrdUPold[32];  // 0..30, 31 = 0
2297       char LINE10[257]; // 000..255, 256 = 0
2298       char ZEROS[4]; // 0..3, 0 = 0, 1 = 0, 2 = 0, 3 = 0
2299       char CRdLFa[2];  // 0..1, 0 = 13, 1 = 10
2300       char workbyte;
2301       char workK[1024*128];
2302       long workKoffset = -1;
2303       int FoundInLinkedList, Slot;
2304       unsigned long OffsetsInBuffer[31]; // 00..30
2305       unsigned long MAXusedBuffer[32]; // 00 not used, only 01..31
2306       unsigned long GRMBLhill[32]; // 00..31
2307       unsigned long GRMBLFoolAgain[32]; // 00..31
2308       int Melnitchka;
2309       unsigned long MAXusedBufferABS = 0;
2310       unsigned long Utiliza1 = 0;
2311       unsigned long Utiliza2 = 0;
2312       unsigned long TotalWLchars = 0;
2313
2314 /* minimum signed 64 bit value */
2315 #define _I64_MIN    (-9223372036854775807i64 - 1)
2316 /* maximum signed 64 bit value */
2317 #define _I64_MAX      9223372036854775807i64
2318 /* maximum unsigned 64 bit value */
2319 #define _UI64_MAX     0xffffffffffffffffui64
2320
2321 /* minimum signed 128 bit value */
2322 #define _I128_MIN   (-170141183460469231731687303715884105727i128 - 1)
2323 /* maximum signed 128 bit value */
2324 #define _I128_MAX     170141183460469231731687303715884105727i128
2325 /* maximum unsigned 128 bit value */
2326 #define _UI128_MAX    0xffffffffffffffffffffffffffffffffui128
2327
2328       char llTOaDigits[27]; // 9,223,372,036,854,775,807: 1(sign or carry)+19(digits)+1('\0')+6(,)
2329       // below duplicates are needed because of one_line_invoking need different buffers.
2330       char llTOaDigits2[27]; // 9,223,372,036,854,775,807: 1(sign or carry)+19(digits)+1('\0')+6(,)
2331       char llTOaDigits3[27]; // 9,223,372,036,854,775,807: 1(sign or carry)+19(digits)+1('\0')+6(,)
2332       char llTOaDigits4[27]; // 9,223,372,036,854,775,807: 1(sign or carry)+19(digits)+1('\0')+6(,)
2333       unsigned long HEADOffetFromStartBUKVA = 0;
2334       unsigned long TAILOffetFromStartBUKVA = 0;
2335       int BSTorBtree = 0;
2336       int SplitOccured;
2337       int POffsetInLEAF;
2338 char *Auberge[4] = {"|\0","/\0","-\0","\\\0"};
2339 int hashAlfalfa, iAlfalfa;
2340
2341 // INIT INIT INIT INIT INIT INIT INIT INIT INIT INIT INIT INIT INIT INIT INIT
2342 puts( "Leprechaun(Fast Greedy Word-Ripper), rev. 13_7pluses, written by Svalqyatchx." );
2343 puts( "Leprechaun: 'Oh, well, didn't you hear? Bigger is good, but jumbo is dear.'" );
2344 puts( "Kaze: Let's see what a 3-way hash + 6,602,752 Binary-Search-Trees can give us," );
2345 puts( "      also the performance of a 3-way hash + 6,602,752 B-Trees of order 3." );
2346 //puts( "Note1: Compiled with Microsoft C v. 13.10.3077: 'cl /Ox /TcLeprechaun.c'." );
2347 //puts( "Note2: This WORDLISTER makes as output pseudo(unsorted)_wordlist_CRLF_file." );
2348 if( argc != 3 && argc != 4 && argc != 5 && argc != 6 ) // +1 for program name
2349 {
2350   puts( "" );
2351 puts( "'The Little Monster' short notes:" );
2352 puts( "Note1: I wish to thank to R.N. Horspool, Ranjan Sinha, Dmitry Shkarin," );
2353 puts( "       Michael Abrash, J. Bentley, R. Sedgewick, Igor Pavlov, Lasse Reinhold," );
2354 puts( "       Landon Noll, Peter Kankowski for sharing their knowledge to public." );
2355 puts( "Note2: Run it without parameters to get usage and short notes." );
```



```
2356 puts( "Note3: This simple amateurish(more over I am not versed well neither in C nor" );
2357 puts( "       in mathematics nor in english language, but I am persistent in INDEXING" );
2358 puts( "       GBs of english TEXTS) tool is written in ANSI C(at least its source is" );
2359 puts( "       compileable for CL(Windows) and GCC(Linux)), and its purpose is to" );
2360 puts( "       create a WordList for a group of files(given via filelist)." );
2361 puts( "       Its name comes(according to Heritage Dictionary) from 'low corpus' or" );
2362 puts( "       'little body', in fact from amazing movie saga 'Leprechaun 1-2-3-4-5-6'" );
2363 puts( "       starring by Warwick Davis." );
2364 puts( "Note4: Only words up to 31 chars are proceeded - the reason is 'DDT'(the" );
2365 puts( "       longest word in Heritage Dictionary 3rd edition) or" );
2366 puts( "       'dichlorodiphenyltrichloroethane'." );
2367 puts( "Note5: Cursor hiding in C - mission impossible for me." );
2368 puts( "Note6: By default(third parameter is 1023) allocated memory is 393MB." );
2369 puts( "       Due to 'malloc()' limitation under WINDOWS, maximum value of third" );
2370 puts( "       parameter is 5174 which is 1988MB allocated block." );
2371 puts( "Note7: File Leprechaun.LOG is a log, where new statistics are appended." );
2372 puts( "Note8: Revision 12+ can handle files larger than 4GB." );
2373 puts( "Note9: Revision 12++ has a buffered 'fread()' - therefore I/O READ-BURST SPEED" );
2374 puts( "       is the first(worst) bottleneck, as a result r.12++ is much-much faster;" );
2375 puts( "       the second(worse) bottleneck: the linked lists - the b-trees" );
2376 puts( "       might be the answer; the third(bad) bottleneck: the amateurish author." );
2377 puts( "NoteA: Revision 12+++ has an improved(2 bits were used doltishly) main hash" );
2378 puts( "       function - therefore less collisions, for example:" );
2379 puts( "       for file 'wikipedia-de-html.tar' 42,291,855,360 bytes with" );
2380 puts( "       5,750,179,678 words of them 7,375,373 distinct attempts to Find/Put" );
2381 puts( "       a WORD into a linked list are 6,117,675,470(r.12++) and 5,845,989,790" );
2382 puts( "       (r.12+++); also two 'if' sections were moved because they were executed" );
2383 puts( "       unnecessarily many times." );
2384 puts( "NoteB: Revision 13 uses BSTs instead of LLs, that is Linked-Lists were" );
2385 puts( "       replaced by Binary-Search-Trees, as a result for 22,202,980 distinct" );
2386 puts( "       words(out of 35,271,297) r.12+++ needs 225,548,268 total attempts to" );
2387 puts( "       Find/Put WORDs into linked lists where r.13 needs 121,674,042 total" );
2388 puts( "       attempts to Find/Put WORDs into Binary-Search-Trees. But this is a" );
2389 puts( "       significant boost in performance only for wordlists of million words." );
2390 puts( "NoteC: Revision 13+ gives only more statistics. Future revisions could lessen" );
2391 puts( "       number of attempts to Find/Put WORDs into Binary-Search-Trees" );
2392 puts( "       furthermore by making them at some point Perfectly-Balanced. But" );
2393 puts( "       for huge amount(multi-(m|b)illion) of distinct words the b-tree family" );
2394 puts( "       must come in, until then this is the leprechaunish niche." );
2395 puts( "NoteD: Revision 13++ has a little fix(2 unnecessary ZEROings, when a new word" );
2396 puts( "       is inserted, were deleted) and a fixed bug(13+ adds stupidly the" );
2397 puts( "       highest BST to the wordlist). Also B-Tree of order 3 is added as a" );
2398 puts( "       searching method. Main goal of B-Tree is to reduce number of" );
2399 puts( "       comparisons but at nasty cost: a precious time wasted to construct it" );
2400 puts( "       and twice more memory, i.e. one step forward two backward: this tree is" );
2401 puts( "       more effective than BST in cases of 2++ billion/million" );
2402 puts( "       different/distinct words." );
2403 puts( "       The improvement which comes from using B-Tree of order 3 is about 200%" );
2404 puts( "       much more pleasing than I expected, for wikipedia-en-html.tar.wrd with" );
2405 puts( "       12,561,874 distinct words Total Attempts to Find/Put WORDs into:" );
2406 puts( "       Binary-Search-Trees was 61,895,043 while for" );
2407 puts( "       B-trees order 3 was 19,295,791." );
2408 puts( "NoteE: Revision 13+++ has a faster(not heavily tested yet) and with" );
2409 puts( "       better(0.6% to 1.1%) dispersion Fowler/Noll/Vo hash," );
2410 puts( "       so called FNV1a hash. Revision 13+++++ boosting: Leprechaun_Intel.exe" );
2411 puts( "       gives 1,256,187W/s for wikipedia-en-html.tar.wrd with FNV1_32_PRIME:" );
2412 puts( "       107712257 with 3,551,736 dispersion for 'FNV1a_Hash_Granularity'." );
2413 puts( "NoteF: For old r.12+ a USB connected HDD crippled test:" );
2414 puts( "       for 'H:\\>Leprechaun.exe static.wikipedia.org_downloads_2008-06_en.lst" );
2415 puts( "       wikipedia-en-html.tar.wrd 5400'" );
2416 puts( "       where 223,674,511,360 wikipedia-en-html.tar" );
2417 puts( "       on laptop Toshiba Pentium T3400 2166 MHz with" );
2418 puts( "       Motherboard Name:                                Toshiba Satellite L305" );
2419 puts( "       CPU Type:         Mobile DualCore Intel Pentium, 2166 MHz (13 x 167)" );
2420 puts( "       CPU Alias:                                                  Merom-1M" );
2421 puts( "       L1 Code Cache:                                            32 KB per core" );
2422 puts( "       L1 Data Cache:                                            32 KB per core" );
2423 puts( "       L2 Cache:                        1 MB (On-Die, ECC, ASC, Full-Speed)" );
2424 puts( "       Bus Type:                                               Dual DDR2 SDRAM" );
2425 puts( "       Bus Width:                                                   128-bit" );
2426 puts( "       Real Clock:                                          333 MHz (DDR)" );
2427 puts( "       Effective Clock:                                            666 MHz" );
2428 puts( "       EVEREST v5.00.1650 Memory Copy:       3725MB/s with timings 5-5-5-13" );
2429 puts( "       result is logged to 'Leprechaun.LOG':" );
2430 puts( "  Bytes per second performance: 20,658,955B/s" );
2431 puts( "  Words per second performance: 2,860,880W/s" );
2432 puts( "  Input File with a list of TEXTual Files:" );
2433 puts( "   static.wikipedia.org_downloads_2008-06_en.lst" );
2434 puts( "  Size of all TEXTual Files: 223,674,511,360" );
2435 puts( "  Word count: 30,974,750,142 of them 12,561,874 distinct" );
2436 puts( "  Number Of Files: 1" );
2437 puts( "  Number Of Lines: 2088618575" );
2438 puts( "  Allocated memory in MB: 1920" );
2439 puts( "  Words with length 01 occupy 0,033KB of 0,349KB given i.e. 09% utilization" );
2440 puts( "  Words with length 02 occupy 0,033KB of 0,349KB given i.e. 09% utilization" );
2441 puts( "  Words with length 03 occupy 0,037KB of 0,697KB given i.e. 05% utilization" );
2442 puts( "  Words with length 04 occupy 0,151KB of 0,871KB given i.e. 17% utilization" );
2443 puts( "  Words with length 05 occupy 0,744KB of 1,568KB given i.e. 47% utilization" );
```

```
puts( "  Words with length 06 occupy 1,470KB of 3,136KB given i.e. 46% utilization" );
puts( "  Words with length 07 occupy 2,605KB of 5,923KB given i.e. 43% utilization" );
puts( "  Words with length 08 occupy 3,296KB of 6,968KB given i.e. 47% utilization" );
puts( "  Words with length 09 occupy 3,714KB of 6,968KB given i.e. 53% utilization" );
puts( "  Words with length 10 occupy 3,483KB of 6,968KB given i.e. 49% utilization" );
puts( "  Words with length 11 occupy 3,235KB of 5,923KB given i.e. 54% utilization" );
puts( "  Words with length 12 occupy 2,691KB of 4,181KB given i.e. 64% utilization" );
puts( "  Words with length 13 occupy 2,230KB of 3,484KB given i.e. 64% utilization" );
puts( "  Words with length 14 occupy 1,718KB of 3,484KB given i.e. 49% utilization" );
puts( "  Words with length 15 occupy 1,357KB of 2,613KB given i.e. 51% utilization" );
puts( "  Words with length 16 occupy 1,063KB of 2,613KB given i.e. 40% utilization" );
puts( "  Words with length 17 occupy 0,814KB of 1,742KB given i.e. 46% utilization" );
puts( "  Words with length 18 occupy 0,617KB of 1,742KB given i.e. 35% utilization" );
puts( "  Words with length 19 occupy 0,485KB of 1,742KB given i.e. 27% utilization" );
puts( "  Words with length 20 occupy 0,402KB of 1,742KB given i.e. 23% utilization" );
puts( "  Words with length 21 occupy 0,327KB of 1,742KB given i.e. 18% utilization" );
puts( "  Words with length 22 occupy 0,274KB of 1,742KB given i.e. 15% utilization" );
puts( "  Words with length 23 occupy 0,224KB of 1,394KB given i.e. 16% utilization" );
puts( "  Words with length 24 occupy 0,190KB of 1,394KB given i.e. 13% utilization" );
puts( "  Words with length 25 occupy 0,162KB of 1,394KB given i.e. 11% utilization" );
puts( "  Words with length 26 occupy 0,136KB of 1,220KB given i.e. 11% utilization" );
puts( "  Words with length 27 occupy 0,119KB of 1,046KB given i.e. 11% utilization" );
puts( "  Words with length 28 occupy 0,107KB of 0,871KB given i.e. 12% utilization" );
puts( "  Words with length 29 occupy 0,091KB of 0,697KB given i.e. 13% utilization" );
puts( "  Words with length 30 occupy 0,080KB of 0,523KB given i.e. 15% utilization" );
puts( "  Words with length 31 occupy 0,076KB of 0,523KB given i.e. 14% utilization" );
puts( "  Total pseudo(including hash table) memory utilization: 42%" );
puts( "  Total real(wordlist's words VS allocated block) memory utilization: 60/1000" );
puts( "  Used value for third parameter in KB: 5400" );
puts( "  Use next time as third parameter: 3475-" );
puts( "  Time for making unsorted wordlist: 10827 second(s)" );
puts( "  Time for sorting unsorted wordlist: 10 second(s)" );

  puts( "" );
  puts( "Usage: Leprechaun InFile OutFile [BufferSize] [SortMethod] [TreeMethod]" );
  puts( "      <InFile>: Input file with files for Leprechauning, in WINDOWS console" );
  puts( "                you can create it by 'E:\\KAZEHOME>dir *.txt/s/b>Leprechaun.lst'" );
  puts( "      <OutFile>: Output WORDLIST(sorted since r.9, CRLF) file" );
  puts( "      <BufferSize>: Optional Dynamic RAM buffer in KB, default(and minimum" );
  puts( "                    in the same time) is 1023, i.e. omit or specify greater one" );
  puts( "      <SortMethod>: Optional Sort Method, default is 'D'," );
  puts( "                    A - InsertionSort" );
  puts( "                    B - InsertionX26Sort" );
  puts( "                    C - MultiKeyQuickSortSort by J. Bentley, R. Sedgewick" );
  puts( "                    D - MultiKeyQuickSortX26Sort' by J. Bentley, R. Sedgewick" );
  puts( "      <TreeMethod>: Optional Tree Method, default is 'X'," );
  puts( "                    X - Binary-Search-Trees" );
  puts( "                    Y - B-Trees of order 3" );
  puts( "" );
puts( "Have a nice Leprechauning." );
puts( "For contacts: sanmayce@sanmayce.com" );
puts( "Sanmayce Svalqyatchx 'Kaze', 2005 Feb 07(rev. 13_7pluses: 2010 Nov 16)." );
  return( 1 );
}

  GRMBLhill[0]=0;
  GRMBLhill[1]=1;
  GRMBLhill[2]=1;
  GRMBLhill[3]=1;
  GRMBLhill[4]=4;
  GRMBLhill[5]=11;
  GRMBLhill[6]=22;
  GRMBLhill[7]=37;
  GRMBLhill[8]=47;
  GRMBLhill[9]=53;
  GRMBLhill[10]=50;
  GRMBLhill[11]=46;
  GRMBLhill[12]=38;
  GRMBLhill[13]=32;
  GRMBLhill[14]=25;
  GRMBLhill[15]=20;
  GRMBLhill[16]=18;
  GRMBLhill[17]=14;
  GRMBLhill[18]=10;
  GRMBLhill[19]=8;
  GRMBLhill[20]=7;
  GRMBLhill[21]=6;
  GRMBLhill[22]=5;
  GRMBLhill[23]=4;
  GRMBLhill[24]=3;
  GRMBLhill[25]=3;
  GRMBLhill[26]=2;
  GRMBLhill[27]=2;
  GRMBLhill[28]=2;
  GRMBLhill[29]=2;
  GRMBLhill[30]=1;
  GRMBLhill[31]=1;

```



```
if( ( fp_in = fopen( argv[1], "rb" ) ) == NULL )
{ printf( "Leprechaun: Can't open file %s \n", argv[1] ); return( 1 ); }

fseek( fp_in, 0L, SEEK_END );
size_in = ftell( fp_in );
fseek( fp_in, 0L, SEEK_SET );
printf( "Size of input file with files for Leprechauning: %lu\n", size_in );

if( ( fp_outLOG = fopen( "Leprechaun.LOG", "a+" ) ) == NULL )
{ printf( "Leprechaun: Can't open file Leprechaun.LOG.\n" ); return( 1 ); }

// argc is 4|5|6 due to eventual missing BufferSize
if( argc == 4 ) // not 6 due to eventual missing BufferSize and SortMethod
    k = 3;
if( argc == 5)  // not 6 due to eventual missing BufferSize or SortMethod
    k = 4;
if( argc == 6 )
    k = 5;
if (*argv[k] == 'Y' || *argv[k] == 'y') BSTorBtree = 1;

if( argc == 4 || argc == 5 || argc == 6 ) Thunderwith = atoi( argv[3] );
else Thunderwith = 527; // for r.12:   527=17*31 this is minimum because of 4096*1*4=16KB+ needed for each buffer!
                        // for r.12+: 1023=33*31 this is minimum because of 4096*2*4=32KB+ needed for each buffer!
if (Thunderwith < 1023) {Thunderwith = 1023;}
LetterBuffer = Thunderwith * 1024;
  WHOLEletter_BufferSize = 0;
  for( i = 1; i <= 31; i++ )
  { OffsetsInBuffer[i-1] = 0;
    for( j = 1; j <= i; j++ )
    { OffsetsInBuffer[i-1] = OffsetsInBuffer[i-1] + (GRMBLhill[(int)(j-1)] * LetterBuffer)/31;
    }
    WHOLEletter_BufferSize = WHOLEletter_BufferSize + (GRMBLhill[(int)i] * LetterBuffer)/31; // Make soon 32 in order to shift >>5
    GRMBLFoolAgain[(int)i] = (GRMBLhill[(int)i] * LetterBuffer)/31;
  }
  memory_size = 26 * WHOLEletter_BufferSize + 1 + 64;
printf( "Allocating memory %luMB ... ", (memory_size>>20)+1 );
pointerflushUNALIGN = (char *)malloc( memory_size );
if( pointerflushUNALIGN == NULL )
{ 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;
  }
}

// PROGRAM PROGRAM PROGRAM PROGRAM PROGRAM PROGRAM PROGRAM PROGRAM PROGRAM
(void) time(&t1);
        Melnitchka = 0;
        WORDcount = 0; // Total word count i.e. for all files!
        WORDcountDistinct = 0;
        NumberOfFiles = 0;
        NumberOfLines = 0;
        FilesLEN = 0;
        LINE1Olen = 0;

        for( k = 0; k < size_in; k++ )
{
                fread( &workbyte, 1, 1, fp_in );
                if( workbyte != 10 )
                { if( workbyte != 13 ) // NON UNIX
                  { if( LINE1Olen < 255 ) { LINE1O[ LINE1Olen ] = workbyte; }
                    LINE1Olen++;
                  }
                  else
                  {
                  }
    }
    else
                { if( 1 <= LINE1Olen && LINE1Olen <= 255 )
                  { LINE1O[ LINE1Olen ] = 0;
```

```
if( ( fp_inLINE = fopen( LINE10, "rb" ) ) == NULL )
{ printf( "Leprechaun: Can't open file %s \n", LINE10 ); return( 1 ); }

//fseek( fp_inLINE, 0L, SEEK_END );  //Rev. 12
//size_inLINE = ftell( fp_inLINE );  //Rev. 12
//fseek( fp_inLINE, 0L, SEEK_SET );  //Rev. 12

#if defined(_WIN32_ENVIRONMENT_)
   // 64bit:
_lseeki64( fileno(fp_inLINE), 0L, SEEK_END );
size_inLINESIXFOUR = _telli64( fileno(fp_inLINE) );
_lseeki64( fileno(fp_inLINE), 0L, SEEK_SET );
#else
   // 64bit:
fseeko( fp_inLINE, 0L, SEEK_END );
size_inLINESIXFOUR = ftello( fp_inLINE );
fseeko( fp_inLINE, 0L, SEEK_SET );
#endif /* defined(_WIN32_ENVIRONMENT_)  */

printf( "Size of Input TEXTual file: %s\n", _ui64toaKAZEcomma(size_inLINESIXFOUR, llTOaDigits, 10) );
FilesLEN = FilesLEN + size_inLINESIXFOUR;
NumberOfFiles++;

        //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
wrdlen = 0;
        for( i = 0; i < size_inLINESIXFOUR; i++ )
{

                // ~~~~~~~~~~~~ Buffering fread [
                if (workKoffset == -1) {
                        if (i + 1024*128 < size_inLINESIXFOUR) {
                                fread( &workk[0], 1, 1024*128, fp_inLINE );
                                workKoffset = 0;
                                workbyte = workk[workKoffset];
                        } else
                        fread( &workbyte, 1, 1, fp_inLINE );
                } else {
                        workKoffset++;
                        workbyte = workk[workKoffset];
                        if (workKoffset == 1024*128 - 1) workKoffset = -1;
                }
                // ~~~~~~~~~~~~ Buffering fread ]

//                if( isalpha( workbyte ) )
//                {
//                        if( wrdlen < 31 )
//                        { wrd[ wrdlen ] = tolower( workbyte ); }
//                        wrdlen++;
//                }

                if ( workbyte < 'A' ) // Most characters are under alphabet - only one if
                {
ElStupido:
                        // This fragment is MIRRORed: #1 copy [
                        if (workbyte == 10) {NumberOfLines++;}
                        if( 1 <= wrdlen && wrdlen <= 31 )
                {
                                wrd[ wrdlen ] = 0;
                                // OTKACHAM: 1<<17-1 gives 65536 i.e. '-' have had high priority than '<<'
                                //Next line gives error due to mix of '&' and 'double'
                                if ((++WORDcount & ((1<<20)-1)) == 0)
                                { //_ui64toaKAZEzerocomma(WORDcount, llTOaDigits, 10);
                                  //printf( "Word count: %s(%lu/128 done)\r", llTOaDigits, ((long long)i*100) / size_inLINESIXFOUR );
//++Melnitchka;
//Melnitchka = Melnitchka % 4;
//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 ); }
//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 ); }
//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 ); }
//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 ); }
Melnitchka = Melnitchka & 3; // 0 1 2 3: 00 01 10 11
printf( "%s; Word count: %s of them %s distinct; Done: %lu/64\r", Auberge[Melnitchka++], _ui64toaKAZEcomma(WORDcount, llTOaDigits, 10), _ui64toaKAZEcomma((unsigned long long)WORDcountDistinct, llTOaDigits2, 10), ((long long)i<<6) / size_inLINESIXFOUR );
                                }
  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 != 1)
{
// @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ 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 < 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_______

```

```
2945 /*
2946 ; Line 1397
2947 jmp $L2139
2948 $L2042:
2949 ; Line 1408
2950 test edx, edx
2951 jne SHORT $L2110
2952 ; Line 1410
2953 mov ecx, DWORD PTR _bufend$[esp+esi*4+892340]
2954 mov edi, DWORD PTR _GRMBLFoolAgain$[esp+ebx*4+892340]
2955 lea edx, DWORD PTR _bufend$[esp+esi*4+892340]
2956 lea esi, DWORD PTR [ebx+ebx+12]
2957 sub esi, ebp
2958 add esi, ecx
2959 cmp esi, edi
2960 mov DWORD PTR tv4122[esp+892340], edx
2961 jae $L2113
2962 ; Line 1412
2963 mov DWORD PTR [eax+ebp], ecx
2964 ; Line 1413
2965 lea eax, DWORD PTR [ecx+12]
2966 ; Line 1436
2967 jmp $L2749
2968 $L2110:
2969 ; Line 1437
2970 mov DWORD PTR _FoundInLinkedList$[esp+892340], 0
2971 npad 11
2972 $L2141:
2973 ; Line 1438
2974 mov eax, DWORD PTR _FoundInLinkedList$[esp+892340]
2975 test eax, eax
2976 jne $L2142
2977 ; Line 1445
2978 lea ebp, DWORD PTR [edx+12]
2979 mov ecx, ebx
2980 lea edi, DWORD PTR _wrd$[esp+892340]
2981 mov esi, ebp
2982 xor eax, eax
2983 repe cmpsb
2984 je  SHORT $L2682
2985 sbb eax, eax
2986 sbb eax, -1
2987 $L2682:
2988 test eax, eax
2989 jle SHORT $L2143
2990 ; Line 1447
2991 mov edx, DWORD PTR [edx]
2992 ; Line 1449
2993 jmp $L2153
2994 $L2143:
2995 mov ecx, ebx
2996 lea edi, DWORD PTR _wrd$[esp+892340]
2997 mov esi, ebp
2998 xor eax, eax
2999 repe cmpsb
3000 je  SHORT $L2640
3001 sbb eax, eax
3002 sbb eax, -1
3003 $L2640:
3004 test eax, eax
3005 jge SHORT $L2145
3006 ; Line 1451
3007 mov cl, BYTE PTR [edx+ebx+12]
3008 test cl, cl
3009 lea eax, DWORD PTR [edx+ebx+12]
3010 je  SHORT $L2147
3011 ; Line 1459
3012 mov ecx, ebx
3013 lea edi, DWORD PTR _wrd$[esp+892340]
3014 mov esi, eax
3015 xor ebp, ebp
3016 repe cmpsb
3017 je  SHORT $L2695
3018 sbb ebp, ebp
3019 sbb ebp, -1
3020 $L2695:
3021 test ebp, ebp
3022 jle SHORT $L2148
3023 ; Line 1461
3024 mov edx, DWORD PTR [edx+4]
3025 ; Line 1463
3026 jmp SHORT $L2151
3027 $L2148:
3028 mov esi, eax
3029 mov ecx, ebx
3030 lea edi, DWORD PTR _wrd$[esp+892340]
3031 xor eax, eax
3032 repe cmpsb
```



```
3033 je  SHORT $L2642
3034 sbb eax, eax
3035 sbb eax, -1
3036 $L2642:
3037 test eax, eax
3038 jge SHORT $L2150
3039 ; Line 1466
3040 mov edx, DWORD PTR [edx+8]
3041 ; Line 1468
3042 jmp SHORT $L2151
3043 $L2150:
3044 mov DWORD PTR _FoundInLinkedList$[esp+892340], 1
3045 $L2151:
3046 ; Line 1469
3047 mov ecx, DWORD PTR _WORDcountAttemptsToPut$[esp+892340]
3048 mov eax, DWORD PTR _WORDcountAttemptsToPut$[esp+892344]
3049 add ecx, 1
3050 adc eax, 0
3051 mov DWORD PTR _WORDcountAttemptsToPut$[esp+892340], ecx
3052 mov DWORD PTR _WORDcountAttemptsToPut$[esp+892344], eax
3053 ; Line 1473
3054 jmp SHORT $L2153
3055 $L2147:
3056 ; Line 1475
3057 mov edx, DWORD PTR [edx+4]
3058 ; Line 1478
3059 jmp SHORT $L2153
3060 $L2145:
3061 mov DWORD PTR _FoundInLinkedList$[esp+892340], 1
3062 $L2153:
3063 ; Line 1479
3064 mov esi, DWORD PTR _WORDcountAttemptsToPut$[esp+892340]
3065 mov ecx, DWORD PTR _WORDcountAttemptsToPut$[esp+892344]
3066 add esi, 1
3067 adc ecx, 0
3068 test edx, edx
3069 mov DWORD PTR _WORDcountAttemptsToPut$[esp+892340], esi
3070 mov DWORD PTR _WORDcountAttemptsToPut$[esp+892344], ecx
3071 jne $L2141
3072 $L2142:
3073 ; Line 1482
3074 mov edx, DWORD PTR _WORDcountAttemptsToPut$[esp+892340]
3075 mov ecx, DWORD PTR _WORDcountAttemptsToPut$[esp+892344]
3076 or  eax, -1
3077 add edx, eax
3078 adc ecx, eax
3079 mov DWORD PTR _WORDcountAttemptsToPut$[esp+892344], ecx
3080 mov DWORD PTR _WORDcountAttemptsToPut$[esp+892340], edx
3081 $L2139:
3082 */
3083
3084 if (FoundInLinkedList == 0)
3085 {
3086 // 2] if Search failed Trasirascht(pushing in stack PseudoLinkedPointer(visited LEAFs)) Search [
3087 //    'TracingSearch' is the same as 'Search' except that adds the trail in my simulated stack,
3088 //    the goal is not to waste time in 'Search' by dealing with no needed trail in case of not 'Insert'.
3089 //    Simulated stack contains pairs of 'Address of ParentLEAF' + 'Offset of ParentPointer in ParentLEAF i.e. 0 for LP, 4 for MP, 8 for RP'.
3090 //    'Offset ...' saves unnecessary comparisons of NEWword which after splitting goes up.
3091                     memcpy( &PseudoLinkedPointer, BufStart+Slot, 4 );
3092                StackPtr = 0;
3093                     while (PseudoLinkedPointer != 0)
3094                     {
3095 // ***** 'P W P' section [
3096 // LW: existence check if ( *(char *)(PseudoLinkedPointer+4+4+4) != 0 )
3097 // RW: existence check if ( *(char *)(PseudoLinkedPointer+4+4+4+wrdlen) != 0 )
3098                        // here ALWAYS LW exists: no need for existence check - line below
3099                        // if ( *(char *)(PseudoLinkedPointer+4+4+4) != 0 )
3100                        if (memcmp(PseudoLinkedPointer+4+4+4,wrd,wrdlen) > 0) // go LP
3101                           { memcpy( &PseudoLinkedPointerNEW, PseudoLinkedPointer + 0, 4 ); //LP
3102                if (StackPtr > 65536*3-1-1) { printf( "\nLeprechaun: Failure! 'B-tree order 3' simulated stack overflow!\n" ); return( 13 );}
3103                BSTstack[StackPtr] = PseudoLinkedPointer; ++StackPtr; //pt to visited leaf
3104                BSTstack[StackPtr] = 0; ++StackPtr; //LPoffset=0;MPoffset=4;RPoffset=8;
3105                                    PseudoLinkedPointer = PseudoLinkedPointerNEW;
3106                           }
3107                        else if (memcmp(PseudoLinkedPointer+4+4+4,wrd,wrdlen) < 0) // go RP or MP
3108                           { // RW existence check - line below:
3109                             if ( *(char *)(PseudoLinkedPointer+4+4+4+wrdlen) != 0 ) // RW exists
3110                                { // Here all 'P W P' section is repeated; the way of handling case when dynamic number of words in leaf
3111 // ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
3112 // ***** 'P W P' section 2 [
3113 // LW: existence check if ( *(char *)(PseudoLinkedPointer+4+4+4) != 0 )
3114 // RW: existence check if ( *(char *)(PseudoLinkedPointer+4+4+4+wrdlen) != 0 )
3115                        // here ALWAYS RW exists: no need for existence check - line below
3116                        // if ( *(char *)(PseudoLinkedPointer+4+4+4+wrdlen) != 0 )
3117                        if (memcmp(PseudoLinkedPointer+4+4+4+wrdlen,wrd,wrdlen) > 0) // go MP
3118                           { memcpy( &PseudoLinkedPointerNEW, PseudoLinkedPointer + 4, 4 ); //MP
3119                if (StackPtr > 65536*3-1-1) { printf( "\nLeprechaun: Failure! 'B-tree order 3' simulated stack overflow!\n" ); return( 13 );}
3120                BSTstack[StackPtr] = PseudoLinkedPointer; ++StackPtr; //pt to visited leaf
```

```
3121                BSTstack[StackPtr] = 4; ++StackPtr; //LPoffset=0;MPoffset=4;RPoffset=8;
3122                           PseudoLinkedPointer = PseudoLinkedPointerNEW;
3123                         }
3124                      else if (memcmp(PseudoLinkedPointer+4+4+4+wrdlen,wrd,wrdlen) < 0) // go RP
3125                         { // No ?W after RW - go RP
3126                           memcpy( &PseudoLinkedPointerNEW, PseudoLinkedPointer + 4 + 4, 4 ); //RP
3127                if (StackPtr > 65536*3-1-1) { printf( "\nLeprechaun: Failure! 'B-tree order 3' simulated stack overflow!\n" ); return( 13 );}
3128                BSTstack[StackPtr] = PseudoLinkedPointer; ++StackPtr; //pt to visited leaf
3129                BSTstack[StackPtr] = 8; ++StackPtr; //LPoffset=0;MPoffset=4;RPoffset=8;
3130                           PseudoLinkedPointer = PseudoLinkedPointerNEW;
3131                         }
3132                      else FoundInLinkedList = 1; // wrd is RW
3133 // ***** 'P W P' section 2 ]
3134 // ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
3135                            }
3136                         else // RW empty - go MP
3137                            { memcpy( &PseudoLinkedPointerNEW, PseudoLinkedPointer + 4, 4 ); //MP
3138                if (StackPtr > 65536*3-1-1) { printf( "\nLeprechaun: Failure! 'B-tree order 3' simulated stack overflow!\n" ); return( 13 );}
3139                BSTstack[StackPtr] = PseudoLinkedPointer; ++StackPtr; //pt to visited leaf
3140                BSTstack[StackPtr] = 4; ++StackPtr; //LPoffset=0;MPoffset=4;RPoffset=8;
3141                             PseudoLinkedPointer = PseudoLinkedPointerNEW;
3142                           }
3143                      }
3144                   else FoundInLinkedList = 1; // wrd is LW
3145 // ***** 'P W P' section ]
3146                   } // while
3147 // 2] if Search failed Trasirascht(pushing in stack PseudoLinkedPointer(visited LEAFs)) Search ]
3148
3149 // 3] Insert Iterative [
3150 //    There are total 4 situations:
3151 //    Case #1: Outer NODE(including ROOT) [  ][  ][  ][LW][  ]
3152 //    Case #2: Outer NODE(including ROOT) [  ][  ][  ][LW][RW]  Split Occurs   -----
3153 //    Case #3: ROOT                       [LP][MP][  ][LW][  ]                  |  'wrdUP' (wrdlen bytes)
3154 //    Case #4: Inner NODE(including ROOT) [LP][MP][RP][LW][RW]  Split Occurs   --- |   &
3155 //                                                                              | |  'PseudoLinkedPointerNEW' (ptr to NEW LEAF)
3156 //    There are total 2 situations for PARENT LEAF: <------------------------------  ARE GOING UP
3157 //    Case #3: [LP][MP][  ][LW][  ]
3158 //    Case #4: [LP][MP][RP][LW][RW]  Split Occurs
3159
3160 // ~  First deal alonely with the OUTER NODE(LEAF) where Search stopped i.e Case #1 & Case #2:
3161       POffsetInLEAF = BSTstack[--StackPtr];
3162       PseudoLinkedPointer = BSTstack[--StackPtr];
3163 // NOTE: ONE LEAF IS FULL ONLY WHEN LAST CELL FOR KEY(here RW) EXISTS!
3164 // RW: existence check if ( *(char *)(PseudoLinkedPointer+4+4+4+wrdlen) != 0 )
3165 if ( *(char *)(PseudoLinkedPointer+4+4+4+wrdlen) != 0 ) // If LEAF is full: Case #2
3166    { SplitOccured = 1; WORDcountDistinct++; bufNumberOfWords[LetterOffset]++;
3167           // ALlocate NEW LEAF:
3168           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)
3169                      {
3170                      memcpy( &PseudoLinkedPointerNEW, &bufend[LetterOffset], 4 );
3171                      bufend[LetterOffset] = bufend[LetterOffset] + 4 + 4 + 4; // + 4 due to above commenting
3172                      bufend[LetterOffset] = bufend[LetterOffset] + 2*wrdlen;
3173                      if (MAXusedBuffer[wrdlen] < (unsigned long)(bufend[LetterOffset] - BufStart)) {MAXusedBuffer[wrdlen] = (unsigned
    long)(bufend[LetterOffset] - BufStart);}
3174                      }
3175           else
3176           { printf( "\nLeprechaun: Failure! Increment 'Memory for each Letter' parameter(third one)!\n" );
3177 fprintf( fp_outLOG, "Input File with a list of TEXTual Files: %s\n", argv[1] );
3178 fprintf( fp_outLOG, "Size of all TEXTual Files: %s\n", _ui64toaKAZEcomma(FilesLEN, llTOaDigits, 10) );
3179 fprintf( fp_outLOG, "Word count: %s of them %s distinct\n", _ui64toaKAZEcomma(WORDcount, llTOaDigits, 10), _ui64toaKAZEcomma((unsigned long
    long)WORDcountDistinct, llTOaDigits2, 10) );
3180 fprintf( fp_outLOG, "Number Of Files: %lu\n", NumberOfFiles );
3181 fprintf( fp_outLOG, "Number of Lines: %lu\n", NumberOfLines );
3182 fprintf( fp_outLOG, "Allocated memory in MB: %lu\n", (unsigned long)(memory_size>>20)+1 );
3183 fprintf( fp_outLOG, "Total Attempts to Find/Put WORDs into B-trees order 3: %s\n", _ui64toaKAZEcomma(WORDcountAttemptsToPut, llTOaDigits, 10)
    );
3184 for( k = 1; k < 32; k++ )
3185 { 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
3186 }
3187 fprintf( fp_outLOG, "Used value for third parameter in KB: %lu\n", (unsigned long)Thunderwith );
3188              fprintf( fp_outLOG, "Leprechaun: Failure! Increment 'Memory for each Letter' parameter(third one)!\n\n");
3189              return( 1 );
3190           }
3191       if (POffsetInLEAF == 0) // wrd < LW
3192          {
3193                      memcpy( wrdUP, PseudoLinkedPointer+4+4+4, wrdlen ); // LW up
3194                      memcpy( PseudoLinkedPointer+4+4+4, wrd, wrdlen );   // wrd go to OLD LEAF
3195                      memcpy( PseudoLinkedPointerNEW+4+4+4, PseudoLinkedPointer+4+4+4+wrdlen, wrdlen ); // RW go to NEW LEAF
3196                      *(char *)(PseudoLinkedPointer+4+4+4+wrdlen) = 0;    // RW mark unused in OLD LEAF
3197          }
3198       if (POffsetInLEAF == 4) // LW < wrd < RW
3199          {
3200                      memcpy( wrdUP, wrd, wrdlen );                       // wrd up
3201                      memcpy( PseudoLinkedPointerNEW+4+4+4, PseudoLinkedPointer+4+4+4+wrdlen, wrdlen ); // RW go to NEW LEAF
```



```
3202                      *(char *)(PseudoLinkedPointer+4+4+4+wrdlen) = 0;    // RW mark unused in OLD LEAF
3203          }
3204       if (POffsetInLEAF == 8) // wrd > RW
3205          {
3206                      memcpy( wrdUP, PseudoLinkedPointer+4+4+4+wrdlen, wrdlen ); // RW up
3207                      *(char *)(PseudoLinkedPointer+4+4+4+wrdlen) = 0;    // RW mark unused in OLD LEAF
3208                      memcpy( PseudoLinkedPointerNEW+4+4+4, wrd, wrdlen );   // wrd go to NEW LEAF
3209          }
3210    }
3211 else // If LEAF is not full: Case #1
3212    { SplitOccured = 0; WORDcountDistinct++; bufNumberOfWords[LetterOffset]++;
3213       if (POffsetInLEAF == 0) // wrd < [LW][] so [LW][] -> [][LW] -> [wrd][LW]
3214          {
3215                      memcpy( PseudoLinkedPointer+4+4+4+wrdlen, PseudoLinkedPointer+4+4+4, wrdlen );
3216                      memcpy( PseudoLinkedPointer+4+4+4, wrd, wrdlen );
3217          }
3218       if (POffsetInLEAF == 4) // wrd > [LW][] so [LW][] -> [LW][wrd]
3219          {
3220                      memcpy( PseudoLinkedPointer+4+4+4+wrdlen, wrd, wrdlen );
3221          }
3222    }
3223
3224 if (SplitOccured != 0)
3225 {
3226 // ~  Second deal with the INNER NODE(S) i.e Case #3 & Case #4:
3227       while (StackPtr != 0 || SplitOccured != 0)
3228             {
3229        // 'PseudoLinkedPointerNEW' is new LEAF to be inserted
3230        // 'wrdUP' is NEW word to be inserted
3231        if (StackPtr != 0)
3232        {
3233           POffsetInLEAF = BSTstack[--StackPtr];
3234           PseudoLinkedPointer = BSTstack[--StackPtr];
3235 if ( *(char *)(PseudoLinkedPointer+4+4+4+wrdlen) != 0 ) // If LEAF is full: Case #4
3236    { SplitOccured = 1;
3237                      memcpy( wrdUPold, wrdUP, wrdlen ); // LW up
3238                      PseudoLinkedPointerNEWold = PseudoLinkedPointerNEW;
3239           // ALlocate NEW LEAF:
3240           if( (unsigned long)(bufend[LetterOffset] - BufStart) + 2*wrdlen + 4 + 4 + 4 < GRMBLFoolAgain[(int)wrdlen] ) // +4 more for BST
    instead of LL; + more(see LEAF)
3241                      {
3242                      memcpy( &PseudoLinkedPointerNEW, &bufend[LetterOffset], 4 );
3243                      bufend[LetterOffset] = bufend[LetterOffset] + 4 + 4 + 4; // + 4 due to above commenting
3244                      bufend[LetterOffset] = bufend[LetterOffset] + 2*wrdlen;
3245                      if (MAXusedBuffer[wrdlen] < (unsigned long)(bufend[LetterOffset] - BufStart)) {MAXusedBuffer[wrdlen] = (unsigned
    long)(bufend[LetterOffset] - BufStart);}
3246                      }
3247           else
3248           { printf( "\nLeprechaun: Failure! Increment 'Memory for each Letter' parameter(third one)!\n" );
3249 fprintf( fp_outLOG, "Input File with a list of TEXTual Files: %s\n", argv[1] );
3250 fprintf( fp_outLOG, "Size of all TEXTual Files: %s\n", _ui64toaKAZEcomma(FilesLEN, llTOaDigits, 10) );
3251 fprintf( fp_outLOG, "Word count: %s of them %s distinct\n", _ui64toaKAZEcomma(WORDcount, llTOaDigits, 10), _ui64toaKAZEcomma((unsigned long
    long)WORDcountDistinct, llTOaDigits2, 10) );
3252 fprintf( fp_outLOG, "Number Of Files: %lu\n", NumberOfFiles );
3253 fprintf( fp_outLOG, "Number Of Lines: %lu\n", NumberOfLines );
3254 fprintf( fp_outLOG, "Allocated memory in MB: %lu\n", (unsigned long)(memory_size>>20)+1 );
3255 fprintf( fp_outLOG, "Total Attempts to Find/Put WORDs into B-trees order 3: %s\n", _ui64toaKAZEcomma(WORDcountAttemptsToPut, llTOaDigits, 10)
    );
3256 for( k = 1; k < 32; k++ )
3257 { 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
3258 }
3259 fprintf( fp_outLOG, "Used value for third parameter in KB: %lu\n", (unsigned long)Thunderwith );
3260              fprintf( fp_outLOG, "Leprechaun: Failure! Increment 'Memory for each Letter' parameter(third one)!\n\n");
3261              return( 1 );
3262           }
3263       if (POffsetInLEAF == 0) // wrdUPold < LW
3264          {
3265                      memcpy( wrdUP, PseudoLinkedPointer+4+4+4, wrdlen ); // LW up
3266                      memcpy( PseudoLinkedPointer+4+4+4, wrdUPold, wrdlen );   // wrdUPold go to OLD LEAF
3267                      memcpy( PseudoLinkedPointerNEW+4+4+4, PseudoLinkedPointer+4+4+4+wrdlen, wrdlen ); // RW go to NEW LEAF
3268                      *(char *)(PseudoLinkedPointer+4+4+4+wrdlen) = 0;    // RW mark unused in OLD LEAF
3269                      // [LP](PseudoLinkedPointerNEWold)[MP][RP](wrdUPold)[LW][RW]       ------
3270                      //                          pair [LW] PseudoLinkedPointerNEW goes up          |
3271                      // PseudoLinkedPointer:                          PseudoLinkedPointerNEW: |
3272                      // [LP](PseudoLinkedPointerNEWold)[](wrdUPold)  [MP][RP][][RW]      <----
3273                      // no need to put zero in RP because logic is based on words existence:
3274                      memcpy( PseudoLinkedPointerNEW+0, PseudoLinkedPointer+4, 4 );
3275                      memcpy( PseudoLinkedPointerNEW+4, PseudoLinkedPointer+8, 4 );
3276                      memcpy( PseudoLinkedPointer+4, &PseudoLinkedPointerNEWold, 4 );
3277          }
3278       if (POffsetInLEAF == 4) // LW < wrdUPold < RW
3279          {
3280                      memcpy( wrdUP, wrdUPold, wrdlen );                       // wrdUPold up
3281                      memcpy( PseudoLinkedPointerNEW+4+4+4, PseudoLinkedPointer+4+4+4+wrdlen, wrdlen ); // RW go to NEW LEAF
3282                      *(char *)(PseudoLinkedPointer+4+4+4+wrdlen) = 0;    // RW mark unused in OLD LEAF
```

```
                        // [LP][MP](PseudoLinkedPointerNEWold)[RP][LW](wrdUPold)[RW]       ------
                        //                 pair [wrdUPold] PseudoLinkedPointerNEW goes up          |
                        // PseudoLinkedPointer:  PseudoLinkedPointerNEW:                           |
                        // [LP][MP][][LW]        (PseudoLinkedPointerNEWold)[RP][][RW]      <----
                        // no need to put zero in RP because logic is based on words existence:
                        memcpy( PseudoLinkedPointerNEW+0, &PseudoLinkedPointerNEWold, 4 );
                        memcpy( PseudoLinkedPointerNEW+4, PseudoLinkedPointer+8, 4 );
                }
              if (POffsetInLEAF == 8) // wrdUPold > RW
                {
                        memcpy( wrdUP, PseudoLinkedPointer+4+4+4+wrdlen, wrdlen ); // RW up
                        *(char *)(PseudoLinkedPointer+4+4+4+wrdlen) = 0;    // RW mark unused in OLD LEAF
                        memcpy( PseudoLinkedPointerNEW+4+4+4, wrdUPold, wrdlen );   // wrdUPold go to NEW LEAF
                        // [LP][MP][RP](PseudoLinkedPointerNEWold)[LW][RW](wrdUPold)       ------
                        //                 pair [RW] PseudoLinkedPointerNEW goes up                |
                        // PseudoLinkedPointer:  PseudoLinkedPointerNEW:                           |
                        // [LP][MP][][LW]        [RP](PseudoLinkedPointerNEWold)[](wrdUPold) <---
                        // no need to put zero in RP because logic is based on words existence:
                        memcpy( PseudoLinkedPointerNEW+0, PseudoLinkedPointer+8, 4 );
                        memcpy( PseudoLinkedPointerNEW+4, &PseudoLinkedPointerNEWold, 4 );
                }
        }
else // If LEAF is not full: Case #3
    { SplitOccured = 0;
      if (POffsetInLEAF == 0) // wrdUP < [LW][] so [LW][] -> [][LW] -> [wrdUP][LW]
        {
                        memcpy( PseudoLinkedPointer+4+4+4+wrdlen, PseudoLinkedPointer+4+4+4, wrdlen );
                        memcpy( PseudoLinkedPointer+4+4+4, wrdUP, wrdlen );
                                // [LP][MP][] -> [LP][][MP] -> [LP][np][MP]
                        memcpy( PseudoLinkedPointer+8, PseudoLinkedPointer+4, 4 );
                        memcpy( PseudoLinkedPointer+4, &PseudoLinkedPointerNEW, 4 );
        }
      if (POffsetInLEAF == 4) // wrdUP > [LW][] so [LW][] -> [LW][wrdUP]
        {
                        memcpy( PseudoLinkedPointer+4+4+4+wrdlen, wrdUP, wrdlen );
                                // [LP][MP][] -> [LP][MP][np]
                        memcpy( PseudoLinkedPointer+8, &PseudoLinkedPointerNEW, 4 );
        }
          break;
    }
       }
       else // Empty stack means ROOT and more over ROOT is already splitted(Case #4 is off)
       {
     // If LEAF is not full: Case #3
     // THIS IS WHERE A NEW(SECOND) LEAF 'PseudoLinkedPointerROOT' must be allocated:
            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( 1 <= wrdlen && wrdlen <= 31 )
              wrdlen = 0;
                        // This fragment is MIRRORed: #1 copy ]
                }
                //else if( workbyte >= 'A' &&  workbyte <= 'Z' )
                else if( workbyte <= 'z' )
    {
                        if( wrdlen < 31 )
                        { wrd[ wrdlen ] = workbyte + 32 ; }
                        wrdlen++;
    }
                else if( workbyte >= 'a' &&  workbyte <= 'z' )
    {
                        if( wrdlen < 31 )
                        { 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
printf( "%s; Word count: %s of them %s distinct; Done: %lu/64\n", Auberge[Melnitchka++], _ui64toaKAZEcomma(WORDcount, llTOaDigits, 10), _ui64toaKAZEcomma((unsigned long long)WORDcountDistinct, llTOaDigits2, 10), 64 );

        LINE10len = 0;
LINE10[ LINE10len ] = 0;
fclose( fp_inLINE );
                }
                }
        } // 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( "Words per second performance: %sW/s\n", _ui64toaKAZEcomma(WORDcount/((int) t3-t1), llTOaDigits, 10) );  // Rev. 12+

        // 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++;
//  }
```

```
3447
3448   for( j = 0; j < NumberOfSLOTs; j++ )
3449   {
3450                       Slot = j<<2;
3451                       memcpy( &PseudoLinkedPointer, BufStart+Slot, 4 );
3452                       if (PseudoLinkedPointer != 0)
3453                       {
3454 NumberOfTrees++;
3455 if (BSTorBtree != 1)
3456 {
3457 // ========================================================== BST traverse [
3458     // DONE JOB:
3459     // Must be written BST traverse ! with simulated stack i.e. non-recursive.
3460     // ...
3461                                       // /*
3462                                       // Given a binary search tree, print out
3463                                       // its data elements in increasing
3464                                       // sorted order.
3465                                       // */
3466                                       // void printTree(struct node* node) {
3467                                       // if (node == NULL) return;
3468                                       // printTree(node->left);
3469                                       // printf("%d ", node->data);
3470                                       // printTree(node->right);
3471                                       // }
3472
3473     // FUTURE JOB:
3474     // I need functions:
3475     //                  BST_LeafNumber() // greater the better
3476     //                  BST_NodeNumber() // 'BSTcurrent' below
3477     //                  BST_Peak()       // i.e. levels, root has height = 1
3478     //                  BST_PeakIB()     // IBBST(Ideal Balanced BST) has 1 + lgNodeNumber height
3479     // I need 'Ideal Balancing BST FRAGMENT' with simulated stack:
3480     // I need 'Ideal Balancing BST FRAGMENT' to be executed when Peak() >= PeakIB()<<1:
3481
3482 // ------------------------------------------------------------------------ [
3483   BSTcurrentNode = 0; BSTcurrentPeak = 0; BSTcurrentLeaf = 0;
3484                       BSTcurrentPeakMAX = 0; // Height of current BST
3485   StackPtr = 0;
3486   while ( 2==2 ) {
3487         while (PseudoLinkedPointer != 0)
3488               {
3489                 if (StackPtr > 65536*3-1-3) { printf( "\nLeprechaun: Failure! BST simulated stack overflow, too high BST!\n" ); return( 13 );}
3490                 memcpy( &PseudoLinkedPointerNEWleft, PseudoLinkedPointer, 4 );
3491                 PseudoLinkedPointer = PseudoLinkedPointer + 4;
3492                 memcpy( &PseudoLinkedPointerNEWright, PseudoLinkedPointer, 4 );
3493                 BSTstack[StackPtr] = PseudoLinkedPointer + 4; ++StackPtr; //ptr to wrd
3494                 BSTstack[StackPtr] = PseudoLinkedPointerNEWright; ++StackPtr;
3495                 BSTstack[StackPtr] = PseudoLinkedPointerNEWleft; ++StackPtr; //needed for stats not for recursion
3496         // BST stats [
3497                   if (PseudoLinkedPointerNEWleft == 0 && PseudoLinkedPointerNEWright == 0) {BSTcurrentLeaf++; BSTsTotalLEAFs++;}
3498                   BSTcurrentPeak++;
3499                   if (BSTcurrentPeakMAX < BSTcurrentPeak) BSTcurrentPeakMAX = BSTcurrentPeak;
3500                 BSTstack[StackPtr] = BSTcurrentPeak; ++StackPtr; //needed for stats not for recursion
3501         // BST stats ]
3502                 PseudoLinkedPointer = PseudoLinkedPointerNEWright; // choose right instead of 'PseudoLinkedPointerNEWleft' because of stats
    print
3503               }
3504         if (StackPtr == 0) break;
3505         BSTcurrentPeak = BSTstack[--StackPtr]; // level of the node(1 is root) needed only for stats(print)
3506         PseudoLinkedPointerNEWleft = BSTstack[--StackPtr]; // left pointer needed only for stats(print)
3507         PseudoLinkedPointerNEWright = BSTstack[--StackPtr]; // right pointer
3508         memcpy( wrd, BSTstack[--StackPtr], i%31+1 );
3509         fwrite(wrd, i%31+1, 1, fp_out); TotalWLchars = TotalWLchars + i%31+1;
3510         fwrite(CRdLFa, 2, 1, fp_out);
3511         BSTcurrentNode++;
3512         PseudoLinkedPointer = PseudoLinkedPointerNEWleft;
3513   }
3514 // ------------------------------------------------------------------------ ]
3515         // BST stats [
3516         if (BSTwithMAXnode < BSTcurrentNode) {
3517            BSTwithMAXnode = BSTcurrentNode;
3518            BSTwithMAXnodePEAK = BSTcurrentPeakMAX;
3519            BSTwithMAXnodeLEAF = BSTcurrentLeaf;
3520            BSTcurrentNodeMAXqUANTITY = 0;
3521         }
3522         if (BSTwithMAXnode == BSTcurrentNode) BSTcurrentNodeMAXqUANTITY++;
3523         if (BSTwithMAXpeak < BSTcurrentPeakMAX) {
3524            BSTwithMAXpeak = BSTcurrentPeakMAX;
3525            BSTwithMAXpeakNODE = BSTcurrentNode;
3526            BSTwithMAXpeakLEAF = BSTcurrentLeaf;
3527            BSTcurrentPeakMAXqUANTITY = 0;       iBSTwithMAXpeak=i; jBSTwithMAXpeak=j;
3528         }
3529         if (BSTwithMAXpeak == BSTcurrentPeakMAX) BSTcurrentPeakMAXqUANTITY++;
3530         if (BSTwithMAXleaf < BSTcurrentLeaf) {
3531            BSTwithMAXleaf = BSTcurrentLeaf;
3532            BSTwithMAXleafNODE = BSTcurrentNode;
3533            BSTwithMAXleafPEAK = BSTcurrentPeakMAX;
```



```
3534            BSTcurrentLeafMAXqUANTITY = 0;
3535         }
3536         if (BSTwithMAXleaf == BSTcurrentLeaf) BSTcurrentLeafMAXqUANTITY++;
3537         // BST stats ]
3538 // ========================================================== BST traverse ]
3539 } else
3540 {
3541 // $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$ B-tree order 3 traverse [
3542     // DONE JOB:
3543     // Must be written B-tree traverse ! with simulated stack i.e. non-recursive.
3544     // ...
3545   StackPtr = 0;
3546   while ( 2==2 ) {
3547         while (PseudoLinkedPointer != 0)
3548               {
3549                 if (StackPtr > 65536*3-1) { printf( "\nLeprechaun: Failure! B-tree simulated stack overflow, too high B-tree!\n" ); return( 13
    );}
3550                 BSTstack[StackPtr] = PseudoLinkedPointer; ++StackPtr; //ptr to Rwrd
3551 if ( *(char *)(PseudoLinkedPointer + 4 + 4 + 4 + i%31+1) == 0 ) {memcpy( PseudoLinkedPointer + 4 + 4, &BufStart[NumberOfSLOTs*4], 4 );}
3552                 memcpy( &PseudoLinkedPointerNEWleft, PseudoLinkedPointer, 4 );
3553                 memcpy( &PseudoLinkedPointerNEWmiddle, PseudoLinkedPointer + 4, 4 );
3554                 memcpy( &PseudoLinkedPointerNEWright, PseudoLinkedPointer + 4 + 4, 4 );
3555 // Give first from right to left non-zero PTR
3556                 if (PseudoLinkedPointerNEWright !=0 )
3557                 { memcpy( PseudoLinkedPointer + 4 + 4, &BufStart[NumberOfSLOTs*4], 4 );
3558                   PseudoLinkedPointer = PseudoLinkedPointerNEWright;
3559                 }
3560                 else if (PseudoLinkedPointerNEWmiddle !=0 )
3561                 { memcpy( PseudoLinkedPointer + 4, &BufStart[NumberOfSLOTs*4], 4 );
3562                   PseudoLinkedPointer = PseudoLinkedPointerNEWmiddle;
3563                 }
3564                 else if (PseudoLinkedPointerNEWleft !=0 )
3565                 { memcpy( PseudoLinkedPointer, &BufStart[NumberOfSLOTs*4], 4 );
3566                   PseudoLinkedPointer = PseudoLinkedPointerNEWleft;
3567                 }
3568                 else
3569                 {
3570                   PseudoLinkedPointer = 0;
3571                 }
3572               }
3573         if (StackPtr == 0) break;
3574         PseudoLinkedPointer = BSTstack[--StackPtr];
3575                 memcpy( &PseudoLinkedPointerNEWleft, PseudoLinkedPointer, 4 );
3576                 memcpy( &PseudoLinkedPointerNEWmiddle, PseudoLinkedPointer + 4, 4 );
3577                 memcpy( &PseudoLinkedPointerNEWright, PseudoLinkedPointer + 4 + 4, 4 );
3578         if (PseudoLinkedPointerNEWleft+PseudoLinkedPointerNEWmiddle+PseudoLinkedPointerNEWright == 0) // One LEAF is PRINTED when LP=0 MP=0
    RP=0
3579            {
3580         memcpy( wrd, PseudoLinkedPointer + 4 + 4 + 4, i%31+1 );
3581         fwrite(wrd, i%31+1, 1, fp_out); TotalWLchars = TotalWLchars + i%31+1;
3582         fwrite(CRdLFa, 2, 1, fp_out);
3583         if ( *(char *)(PseudoLinkedPointer + 4 + 4 + 4 + i%31+1) != 0 )
3584              { memcpy( wrd, PseudoLinkedPointer + 4 + 4 + 4 + i%31+1, i%31+1 );
3585                fwrite(wrd, i%31+1, 1, fp_out); TotalWLchars = TotalWLchars + i%31+1;
3586                fwrite(CRdLFa, 2, 1, fp_out);
3587              }
3588         PseudoLinkedPointer = 0;
3589            }
3590   }
3591 // $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$ B-tree order 3 traverse ]
3592 }
3593                       }
3594   } // j
3595
3596 } // i
3597
3598 if (BSTorBtree != 1)
3599 {
3600 // ~~~~~ Longest path ~~~~~ [
3601 i=iBSTwithMAXpeak; j=jBSTwithMAXpeak;
3602   BufStart = pointerflush + (i / 31) * WHOLEletter_BufferSize + OffsetsInBuffer[i % 31];
3603                       Slot = j<<2;
3604                       memcpy( &PseudoLinkedPointer, BufStart+Slot, 4 );
3605                       if (PseudoLinkedPointer != 0)
3606                       {
3607 // ------------------------------------------------------------------------ [
3608   BSTcurrentNode = 0; BSTcurrentPeak = 0; BSTcurrentLeaf = 0;
3609                       BSTcurrentPeakMAX = 0; // Height of current BST
3610   StackPtr = 0;
3611         // BST print [
3612                   fprintf( fp_outLOG, "A(not always THE) Binary-Search-Tree with the longest path(height, PEAK, number of levels):\n" );
3613         // BST print ]
3614   while ( 2==2 ) {
3615         while (PseudoLinkedPointer != 0)
3616               {
3617                 if (StackPtr > 65536*3-1-3) { printf( "\nLeprechaun: Failure! BST simulated stack overflow, too high BST!\n" ); return( 13 );}
3618                 memcpy( &PseudoLinkedPointerNEWleft, PseudoLinkedPointer, 4 );
3619                 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 ]
}

(void) time(&t2);
if (t2 <= t1) {t2 = t1; t2++;}
printf("Time for making unsorted wordlist: %d second(s)\n", (int) t2-t1);
fprintf( fp_outLOG, "Bytes per second performance: %sB/s\n", _ui64toaKAZEcomma(FilesLEN/((int) t3-t1), llTOaDigits, 10) );  // Rev. 12+
fprintf( fp_outLOG, "Words per second performance: %sW/s\n", _ui64toaKAZEcomma(WORDcount/((int) t3-t1), llTOaDigits, 10) );  // Rev. 12+
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 );
NumberOfHashCollisions = WORDcountDistinct - NumberOfTrees;
fprintf( fp_outLOG, "Number Of Trees(GREATER THE BETTER): %lu\n", NumberOfTrees );
fprintf( fp_outLOG, "Forest population(Hash Function Quality regarding Collisions i.e. Hash Table Utilization): %lu%s\n", (NumberOfTrees*100)/(26*31*8192), "%\0" );
fprintf( fp_outLOG, "Number Of Hash Collisions(Distinct WORDs - Number Of Trees): %lu\n", NumberOfHashCollisions );

if (BSTorBtree != 1)
{
fprintf( fp_outLOG, "Maximum Attempts to Find/Put a WORD into a Binary-Search-Tree: '%s'\n", _ui64toaKAZEcomma(BSTwithMAXpeak, llTOaDigits, 10) );
fprintf( fp_outLOG, "Total Attempts to Find/Put WORDs into Binary-Search-Trees: %s\n", _ui64toaKAZEcomma(WORDcountAttemptsToPut, llTOaDigits, 10) );
fprintf( fp_outLOG, "Total Number of LEAFs in Binary-Search-Trees(GREATER THE BETTER): %s\n", _ui64toaKAZEcomma(BSTsTotalLEAFs, llTOaDigits, 10) );
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));
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));
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));
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));
```



```
} else
{
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
  if ( MAXusedBufferABS < (31 * ((MAXusedBuffer[k]>>10)+1)) / GRMBLhill[(int)k] ) {MAXusedBufferABS = 1+(31 * ((MAXusedBuffer[k]>>10)+1)) / GRMBLhill[(int)k];}
  Utiliza1 = Utiliza1 + (MAXusedBuffer[k]>>10)+1;
  Utiliza2 = Utiliza2 + (((GRMBLhill[(int)k] * LetterBuffer)/31)>>10)+1;
}
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
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
fprintf( fp_outLOG, "Used value for third parameter in KB: %lu\n", (unsigned long)Thunderwith );
fprintf( fp_outLOG, "Use next time as third parameter: %lu-\n", MAXusedBufferABS ); // 26 are all 26-DESIRED=24
fprintf( fp_outLOG, "Time for making unsorted wordlist: %d second(s)\n", (int) t2-t1);

// EXIT EXIT EXIT EXIT EXIT EXIT EXIT EXIT EXIT EXIT EXIT EXIT EXIT EXIT
printf( "Deallocated memory in MB: %lu\n", (memory_size>>20)+1 );
free(pointerflushUNALIGN);
fclose(fp_out);
fclose(fp_outLOG);
// SORT SORT SORT SORT SORT SORT SORT SORT SORT SORT SORT SORT SORT SORT
   //printf("Uploading unsorted wordlist ...\n");
   if ((nlines = readlines(argv[2], &backup)) >= 0)
   {  //printf("Number of words(lines) uploaded: %lu\n", nlines);
      //printf("Note1: Press 'Ctrl+C' to abort sorting, unsorted wordlist(second parameter)\n");
      //printf("       will remain intact(unless flushing is in progress) because of\n");
      //printf("       pointers-to-data are being sorted not the data itself.\n");
      //printf("Note2: In near future 'InsertionX26Sort' will be replaced with 'QuickX26Sort':\n");
      //printf("       which is much faster than 'QuickSort' applied for data-at-once!\n");

      // !!!??? I AM DISAPPOINTED x26 is just an illusion !!!???
      // X26 x26 x26 x26 x26 x26 x26 x26 x26 x26 x26 x26 x26 x26 x26 x26 x26
// argc is 4|5|6 due to eventual missing BufferSize
if( argc == 4 ) // not 5 due to eventual missing BufferSize
    k = 3;
if( argc == 5 || argc == 6 )
    k = 4;
if (*argv[k] != 'A' && *argv[k] != 'a' && *argv[k] != 'B' && *argv[k] != 'b' && *argv[k] != 'C' && *argv[k] != 'c' && *argv[k] != 'D' && *argv[k] != 'd')
{ printf("Sorting(with 'MultiKeyQuickSortX26Sort' by J. Bentley and R. Sedgewick) ...\n");
      /* ???!!! What an unexpected behavior! I have been hit: for SOED5.HTM(259,835 distinct words): InsertionX26Sort gives 46s, InsertionSort gives 28s */
      for( k = 0; k < 26; k++ )
      { printf( "Sort pass %s/26 ...\r", _ui64toaKAZEzerocomma(k+1, llTOaDigits, 10)+(26-2));
        HEADOffetFromStartBUKVA = TAILOffetFromStartBUKVA;
        while( (TAILOffetFromStartBUKVA < nlines) && (*backup[TAILOffetFromStartBUKVA] - 'a' == k) )
        { TAILOffetFromStartBUKVA++;
        }
        if (HEADOffetFromStartBUKVA != TAILOffetFromStartBUKVA)
        { mkqsort_main(backup + HEADOffetFromStartBUKVA, TAILOffetFromStartBUKVA - HEADOffetFromStartBUKVA + 0); // backup[0..nlines-1]
        }
      }
}
else
{
if (*argv[k] == 'A' || *argv[k] == 'a')
   { printf("Sorting(with 'InsertionSort') ...");
     InsertSortKAZE(backup, nlines, 0); // backup[0..nlines-1]
   }
if (*argv[k] == 'B' || *argv[k] == 'b')
   { printf("Sorting(with 'InsertionX26Sort') ...\n");
      /* ???!!! What an unexpected behavior! I have been hit: for SOED5.HTM(259,835 distinct words): InsertionX26Sort gives 46s, InsertionSort gives 28s */
      for( k = 0; k < 26; k++ )
      { printf( "Sort pass %s/26 ...\r", _ui64toaKAZEzerocomma(k+1, llTOaDigits, 10)+(26-2));
        HEADOffetFromStartBUKVA = TAILOffetFromStartBUKVA;
        while( (TAILOffetFromStartBUKVA < nlines) && (*backup[TAILOffetFromStartBUKVA] - 'a' == k) )
        { TAILOffetFromStartBUKVA++;
        }
        if (HEADOffetFromStartBUKVA != TAILOffetFromStartBUKVA)
        { InsertSortKAZE(backup + HEADOffetFromStartBUKVA, TAILOffetFromStartBUKVA - HEADOffetFromStartBUKVA + 0, 0); // backup[0..nlines-1]
        }
      }
   }
if (*argv[k] == 'C' || *argv[k] == 'c')
   { printf("Sorting(with 'MultiKeyQuickSortSort' by J. Bentley and R. Sedgewick) ...");
     mkqsort_main(backup, nlines); // backup[0..nlines-1]
   }
if (*argv[k] == 'D' || *argv[k] == 'd')
```

```
3770    { printf("Sorting(with 'MultiKeyQuickSortX26Sort' by J. Bentley and R. Sedgewick) ...\n");
3771      /* ???!!! What an unexpected behavior! I have been hit: for SOED5.HTM(259,835 distinct words): InsertionX26Sort gives 46s, InsertionSort
    gives 28s */
3772      for( k = 0; k < 26; k++ )
3773      { printf( "Sort pass %s/26 ...\r", _ui64toaKAZEzerocomma(k+1, llTOaDigits, 10)+(26-2));
3774        HEADOffetFromStartBUKVA = TAILOffetFromStartBUKVA;
3775        while( (TAILOffetFromStartBUKVA < nlines) && (*backup[TAILOffetFromStartBUKVA] - 'a' == k) )
3776        { TAILOffetFromStartBUKVA++;
3777        }
3778        if (HEADOffetFromStartBUKVA != TAILOffetFromStartBUKVA)
3779        { mkqsort_main(backup + HEADOffetFromStartBUKVA, TAILOffetFromStartBUKVA - HEADOffetFromStartBUKVA + 0); // backup[0..nlines-1]
3780        }
3781      }
3782   }
3783 }
3784      // X26 X26 X26 X26 X26 X26 X26 X26 X26 X26 X26 X26 X26 X26 X26 X26 X26 X26 X26 X26 X26
3785
3786      printf("\nFlushing sorted words ...\n");
3787      if( ( fp_out = fopen( argv[2], "wb+" ) ) == NULL )
3788      { printf( "Leprechaun: Can't create file %s \n", argv[2] ); return( 1 ); }
3789      for( j = 0; j < nlines; j++ )
3790      { //Slot = KuxHash3plus(backup[j]);
3791        //fprintf(fp_out, "Hashcode: %s - ", _ui64toaKAZEzerocomma(Slot, llTOaDigits, 10)+(26-5));
3792        fprintf(fp_out, "%s", backup[j]);
3793        fwrite(CRdLFa, 2, 1, fp_out );
3794      }
3795      (void) time(&t3);
3796      if (t3 <= t2) {t3 = t2; t3++;}
3797      printf("Time for sorting unsorted wordlist: %d second(s)\n", (int) t3-t2);
3798
3799 /*
3800 // Hash benchmarking ------------------------------------------------ [
3801
3802 // 5[
3803 clocks1 = clock();
3804     for (Bozan=0; Bozan < (1<<4); Bozan++) // 16 times, at end >>4
3805 {
3806      for( j = 0; j < nlines; j++ )
3807      { //Slot = KuxHash3plus(backup[j]);
3808        //fprintf(fp_out, "Hashcode: %s - ", _ui64toaKAZEzerocomma(Slot, llTOaDigits, 10)+(26-5));
3809              Slot = FNV1A_Hash_4_OCTETS(backup[j], (strlen(backup[j])>>2)); //13+++
3810      }
3811 }
3812 clocks2 = clock();
3813 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))  );
3814 // 5]
3815
3816 // 1[
3817 clocks1 = clock();
3818     for (Bozan=0; Bozan < (1<<4); Bozan++) // 16 times, at end >>4
3819 {
3820      for( j = 0; j < nlines; j++ )
3821      { //Slot = KuxHash3plus(backup[j]);
3822        //fprintf(fp_out, "Hashcode: %s - ", _ui64toaKAZEzerocomma(Slot, llTOaDigits, 10)+(26-5));
3823 // To make it EVEN !!!
3824 //wrdlen = strlen(backup[j]);
3825 //if (strlen(backup[j]) != 0)
3826              Slot = FNV1A_Hash(backup[j]); //13+++
3827      }
3828 }
3829 clocks2 = clock();
3830 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))  );
3831 // 1]
3832
3833 // 2[
3834 clocks1 = clock();
3835     for (Bozan=0; Bozan < (1<<4); Bozan++) // 16 times, at end >>4
3836 {
3837      for( j = 0; j < nlines; j++ )
3838      { //Slot = KuxHash3plus(backup[j]);
3839        //fprintf(fp_out, "Hashcode: %s - ", _ui64toaKAZEzerocomma(Slot, llTOaDigits, 10)+(26-5));
3840              Slot = FNV1A_Hash_4_OCTETS_31(backup[j], (strlen(backup[j])>>2)); //13+++
3841      }
3842 }
3843 clocks2 = clock();
3844 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))  );
3845 // 2]
3846
3847 // 4[
3848 clocks1 = clock();
3849     for (Bozan=0; Bozan < (1<<4); Bozan++) // 16 times, at end >>4
3850 {
3851      for( j = 0; j < nlines; j++ )
3852      { //Slot = KuxHash3plus(backup[j]);
3853        //fprintf(fp_out, "Hashcode: %s - ", _ui64toaKAZEzerocomma(Slot, llTOaDigits, 10)+(26-5));
```



```
3854 // To make it EVEN !!!
3855 //wrdlen = strlen(backup[j]);
3856 //if (strlen(backup[j]) != 0)
3857              Slot = KuxHash3plus(backup[j]); //13++
3858      }
3859 }
3860 clocks2 = clock();
3861 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))  );
3862 // 4]
3863
3864 // 6[
3865 clocks1 = clock();
3866     for (Bozan=0; Bozan < (1<<4); Bozan++) // 16 times, at end >>4
3867 {
3868      for( j = 0; j < nlines; j++ )
3869      { //Slot = KuxHash3plus(backup[j]);
3870        //fprintf(fp_out, "Hashcode: %s - ", _ui64toaKAZEzerocomma(Slot, llTOaDigits, 10)+(26-5));
3871 wrdlen = strlen(backup[j]);
3872 if (wrdlen<=19) // 4x4+3=19 i.e. last contains 7 clashes
3873              Slot = FNV1A_Hash_Granularity(backup[j], wrdlen>>2, 2); //13+++++
3874 else           // 2x8+4=20 i.e. first contains 6 clashes
3875              Slot = FNV1A_Hash_Granularity(backup[j], wrdlen>>3, 3); //13+++++
3876      } // Conclusion: two functions > 64 bytes lead to horrible slowness, so unite them in one: fit in the cache line.
3877 }
3878 clocks2 = clock();
3879 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))  );
3880 // 6]
3881
3882 // Hash benchmarking ------------------------------------------------ ]
3883 */
3884
3885 if( ( fp_outLOG = fopen( "Leprechaun.LOG", "a+" ) ) == NULL )
3886 { printf( "Leprechaun: Can't open file Leprechaun.LOG.\n" ); return( 1 ); }
3887 fprintf( fp_outLOG, "Time for sorting unsorted wordlist: %d second(s)\n\n", (int) t3-t2);
3888 printf( "Leprechaun: Done.\n" );
3889      return 0;
3890   }
3891   else
3892   {  printf("Leprechaun: Input file too large, wordlist remains unsorted!\n");
3893      return 1;
3894   }
3895 // SORT SORT SORT SORT SORT SORT SORT SORT SORT SORT SORT SORT SORT SORT SORT SORT
3896 }
3897
3898 /*
3899 TO BE DONE: Ideal Balancing BST [
3900
3901 link rotR(link h)
3902   { link x = h->l; h->l = x->r; x->r = h;
3903     return x; }
3904
3905 link rotL(link h)
3906   { link x = h->r; h->r = x->l; x->l = h;
3907     return x; }
3908
3909 link partR(link h, int k)
3910   { int t = h->l->N;
3911     if (t > k )
3912       { h->l = partR(h->l, k); h = rotR(h); }
3913     if (t < k )
3914       { h->r = partR(h->r, k-t-1); h = rotL(h); }
3915     return h;
3916   }
3917
3918 link balanceR(link h)
3919   {
3920     if (h->N < 2) return h;
3921     h = partR(h, h->N/2);
3922     h->l = balanceR(h->l);
3923     h->r = balanceR(h->r);
3924     return h;
3925   }
3926
3927 TO BE DONE: Ideal Balancing BST ]
3928 */
3929
3930 /*
3931 #include <stdlib.h>
3932 #include "Item.h"
3933 typedef struct STnode* link;
3934 struct STnode { Item item; link l, r; int N };
3935 static link head, z;
3936 link NEW(Item item, link l, link r, int N)
3937   { link x = malloc(sizeof *x);
3938     x->item = item; x->l = l; x->r = r; x->N = N;
3939     return x;
```

```
  }
void STinit()
  { head = (z = NEW(NULLitem, 0, 0, 0)); }
int STcount() { return head->N; }
Item searchR(link h, Key v)
  { Key t = key(h->item);
    if (h == z) return NULLitem;
    if eq(v, t) return h->item;
    if less(v, t) return searchR(h->l, v);
            else return searchR(h->r, v);
  }
Item STsearch(Key v)
  { return searchR(head, v); }
link insertR(link h, Item item)
  { Key v = key(item), t = key(h->item);
    if (h == z) return NEW(item, z, z, 1);
    if less(v, t)
         h->l = insertR(h->l, item);
    else h->r = insertR(h->r, item);
    (h->N)++; return h;
  }
void STinsert(Item item)
  { head = insertR(head, item); }
*/

/*
int count(link h)
  {
    if (h == NULL) return 0;
    return count(h->l) + count(h->r) + 1;
  }

int height(link h)
  { int u, v;
    if (h == NULL) return -1;
    u = height(h->l); v = height(h->r);
    if (u > v) return u+1; else return v+1;
  }

void printnode(char c, int h)
  { int i;
    for (i = 0; i < h; i++) printf("  ");
    printf("%c\n", c);
  }

void show(link x, int h)
  {
    if (x == NULL) { printnode("*", h); return; }
    show(x->r, h+1);
    printnode(x->item, h);
    show(x->l, h+1);
  }
*/
```