A Collection of Research Processes for Genealogy and Proofs

VOLUME SEVEN, SECTION 37

File Dump, September 29, 1991: Shin's (Mapping) Hash Function Source Code (Changed Version)

by

Dr. Dong-Keun Shin

File Name: MAPGEN PASCAL

A Simulation Program (MAPGEN) for Shin's Mapping Hash Function which reads Generally Chosen Names(GCN) data set and produces output.

File Name: MAPNU PASCAL

A Simulation Program (MAPNU) for Shin's Mapping Hash Function which reads Randomly Chosen Numeric Strings (RNS) data set and produces output

File Name: MAPRAND PASCAL

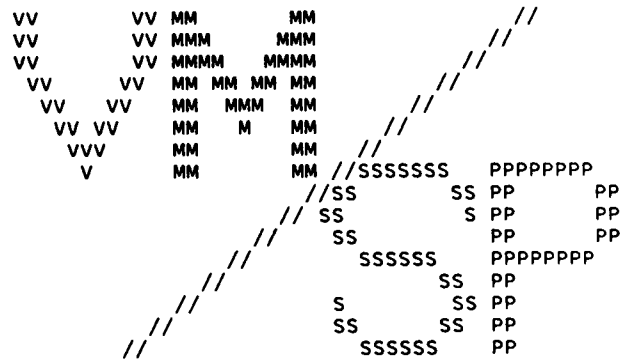
A Simulation Program (FAST_HW_HASHING) for Shin's Mapping Hash Function which reads Randomly Chosen Names (RCN) data set and produces output.

Note: 1. In these Simulation Programs, Shin's Original Mapping Hash Function was changed to test when all of the 16 ROM contests are identical.
2. Dong-Keun Shin printed out every file in his account and keep the hard copies of them before moving out from his computing environment, .

On June 13, 1996, Submitted to the Chair of
Department of Electrical Engineering and Computer Sciences
College of Engineering
University of California, Berkeley
Berkeley, CA 94720

RS EC	RSCSREC	USERID	ORIGIN
SHIN	GWUVM	DISTCODE	SYSTEM
MAPRAND	PASCAL	FILENAME	FILETYPE
09/29/91	21:36:55	FILE CREATION DATE	
9666	00003752	SPOOLID	COUNT
09/29/91	21:37:56	FILE PRINT DATE	
A	0211	CLASS	DEVICE
P1N	OFF	FORMS	DESTINATION

```
333333333  7   77777   9999999
333333333  77 777777  999999999
33    33   77   77    99     99
      33        77    99     99
     3333       77    999999999
    333333      77     99999999
        33      77           99
33      33      77    99     99
333333333       77    999999999
 3333333        77     9999999
```



TAG DATA: FILE (3158) ORIGIN GWUUNIV SHIN 9/29/91 21:36:12 E.D.T.

```
RRRRRRRR    SSSSSSS    CCCCCCC    SSSSSSS   RRRRRRRR   EEEEEEEEE   CCCCCCC
RRRRRRRRR  SSSSSSSSS  CCCCCCCCC  SSSSSSSSS  RRRRRRRRR  EEEEEEEEE  CCCCCCCCC
RR      RR SS      SS CC      CC SS      SS RR      RR EE         CC      CC
RR      RR SS         CC         SS         RR      RR EE         CC
RRRRRRRRR  SSSSSSSS   CC         SSSSSSSS   RRRRRRRRR  EEEEEE     CC
RRRRRRRR    SSSSSSSS  CC          SSSSSSSS  RRRRRRRR   EEEEEE     CC
RR  RR             SS CC                SS RR  RR     EE         CC
RR    RR    SS      SS CC      CC SS      SS RR   RR    EE         CC      CC
RR     RR   SSSSSSSSS  CCCCCCCCC  SSSSSSSSS RR    RR   EEEEEEEEE  CCCCCCCCC
RR      RR   SSSSSSS    CCCCCCC    SSSSSSS  RR     RR  EEEEEEEEE   CCCCCCC

 SSSSSSS   HH      HH   IIIIII    N       NN
SSSSSSSSS  HH      HH   IIIIII    NN      NN
SS     SS  HH      HH     II      NNN     NN
SS         HH      HH     II      NNNN    NN
SSSSSSSS   HHHHHHHHHH    II      NN NN   NN
 SSSSSSSS  HHHHHHHHHH    II      NN  NN  NN
       SS  HH      HH     II      NN   NNNN
SS     SS  HH      HH     II      NN    NNN
SSSSSSSSS  HH      HH   IIIIII    NN     NN
 SSSSSSS   HH      HH   IIIIII    NN      N
```

FILE: MAPGEN PASCAL A THE GEORGE WASHINGTON UNIVERSITY COMPUTER CENTER

```
program MAPGEN (input, output);


const
     MAX_NUM_KEYS = 1024;
     NUM_IDENT_CHAR = 16;
     MAX_BUCKET_ADDR_BITS = 8;
     MAX_BOOL_DIGIT = 13;
     NUM_ASCII_CHAR = 70;

     NUM_FIRST_EXOR = 8;
     NUM_SECOND_EXOR = 4;
     NUM_THIRD_EXOR = 2;

     BUCKET_SIZE = 255;

     DEBUG_FLAG = true;
     DEBUG_ONLY = true;


type
     KEY_ARRAY_TYPE = array (.1..NUM_IDENT_CHAR.) of char;

     PRIME_BOOL_TYPE = array (.1..MAX_BOOL_DIGIT.) of boolean;

     CHAR_RECORD_TYPE = record
                          ch : char;
                          ASCII_num : integer;
                          prime_num : integer;
                          bool_prime : PRIME_BOOL_TYPE;
                        end;

     ASCII_TABLE = array (.1..NUM_IDENT_CHAR, 1..NUM_ASCII_CHAR.) of
                   CHAR_RECORD_TYPE;

     BOOL_PRIME_KEY_TYPE = array (.1..NUM_IDENT_CHAR.) of
                           record
                                bool_prime : PRIME_BOOL_TYPE;
                           end;

     FIRST_EXOR_ARR_TYPE = array (.1..NUM_FIRST_EXOR.) of boolean;

     SECOND_EXOR_ARR_TYPE = array (.1..NUM_SECOND_EXOR.) of boolean;

     THIRD_EXOR_ARR_TYPE = array (.1..NUM_THIRD_EXOR.) of boolean;

     HASHED_KEY_REG_TYPE = array (.1..MAX_BUCKET_ADDR_BITS.) of boolean;

     HASH_ADDR_TYPE = array (.1..MAX_BOOL_DIGIT.) of boolean;

     COUNT_KEY_TYPE = array (.0..BUCKET_SIZE.) of integer;

     LINK = @KEY_RECORD;
     KEY_RECORD = record
                    Key_Arr : KEY_ARRAY_TYPE;
```

```
                    next : LINK;
                  end;

     BUCKET_POINTER_ARRAY = array (.0..BUCKET_SIZE.) of LINK;


var
     Key_Register : KEY_ARRAY_TYPE;

     ASCII_Arr : ASCII_TABLE;

     Key_No, Char_No, Bit_No : integer;

     Key_Char : char;

     index : integer;

     Bool_Key_Arr : BOOL_PRIME_KEY_TYPE;

     EXOR1_Arr : FIRST_EXOR_ARR_TYPE;
     EXOR2_Arr : SECOND_EXOR_ARR_TYPE;
     EXOR3_Arr : THIRD_EXOR_ARR_TYPE;

     Hashed_Addr : HASH_ADDR_TYPE;

     Bucket_Arr : BUCKET_POINTER_ARRAY;

     Count_Arr : COUNT_KEY_TYPE;

     ADDR_OFFSET : integer;



procedure Int_To_Bool_Convert (number:integer; var Bool_Arr:
                                                  PRIME_BOOL_TYPE);

var  i : integer;

begin
     for i := 1 to MAX_BOOL_DIGIT do
          Bool_Arr(.i.) := false;

     i := 1;
     while (number >= 2) and (i <= MAX_BOOL_DIGIT) do
       begin

          if (number mod 2) =1 then
               Bool_Arr(.i.) := true
          else
               Bool_Arr(.i.) := false;

          number := number div 2;
          i := i + 1;
```

```
        end;

      if (number = 1) and (i <= MAX_BOOL_DIGIT) then
           Bool_Arr(.i.) := true;

end;



procedure Initialization;

const
     char_divisor = 20;
     number_divisor = 10;

var  i, j, k: integer;
     number : integer;
     character : char;

begin

     readln (ADDR_OFFSET);
     for i := 1 to NUM_IDENT_CHAR do
       begin

          for j := 1 to NUM_ASCII_CHAR do
            begin

               with ASCII_Arr(.i,j.) do
                 begin
                    ch := '?';
                    prime_num := 7;
                    ASCII_num := 777;

                    for k := 1 to MAX_BOOL_DIGIT do
                         bool_prime(.k.) := false;
                 end;
            end;

       end;

     for j := 1 to NUM_ASCII_CHAR do
       begin
          read(character);

          for i := 1 to NUM_IDENT_CHAR do
            begin
               ASCII_Arr(.i,j.).ch := character;
            end;

          if j mod char_divisor = 0 then
               readln;
    end;
```

```
    readln;


    for j :=1 to NUM_ASCII_CHAR do
      begin
         read(number);
         if number > 64 then number := number - 64;

         for i := 1 to NUM_IDENT_CHAR do
           begin
              ASCII_ARR(.i,j.).ASCII_num := number;
           end;

         if j mod number_divisor = 0 then
              readln;

      end;

    if DEBUG_ONLY then

 for i := 1 to NUM_IDENT_CHAR do
   begin
   writeln;
   writeln;
   write('  THE CONTENTS OF THE RANDOM ACCESS MEMORY FOR CHARACTER ');
   writeln('# ',i:1,' OF A KEY');
   writeln;
   writeln('  <ASCII CHAR> <ORD NO> <PRIME NO> <BINARY PRIME NUMBER>');

    for j := 1 to NUM_ASCII_CHAR do
      begin

         with ASCII_Arr(.i, j.) do
           begin

              read(prime_num);

              if j mod number_divisor = 0 then
                   readln;

              Int_To_Bool_Convert(prime_num, bool_prime);

              if (ch <> '?') and DEBUG_FLAG then
                begin

                   write('      ',ch,' :        ');
                   write(ASCII_num:4,'      ');
                   write(prime_num:6,'    ');

                   for k := MAX_BOOL_DIGIT downto 1 do
                     begin
                        if bool_prime(.k.) then
                             write(1:2)
```

```
                        else
                             write(0:2);
                    end;

                    writeln;
                    writeln;
                end;  {if}
            end;  {with}
        end;  {for}

    readln;
    end;    {for}

    for i := 0 to BUCKET_SIZE do
      begin
         Bucket_Arr(.i.) := nil;
      end;

end;



procedure Init_While_Do_Loop;

var  i: integer;


  ;in

    Char_No := 1;

    for i := 1 to NUM_IDENT_CHAR do
      begin
         Key_Register(.i.) := ' ';
      end;
end;



function More_Chars_Left_For_Key : boolean;

begin

    if ((Char_No > NUM_IDENT_CHAR) or eoln) or eof then
      begin
         More_Chars_Left_For_Key := false;
         readln;
      end
    else
         More_Chars_Left_For_Key := true;
end;
```

```
procedure Read_A_Char;

begin
     read(Key_Char);
     Key_Register(.Char_No.) := Key_Char;
end;




procedure Look_Up_Char_In_Prime_Num_Table(var idx:integer);

var  found : boolean;
     j : integer;

begin
     j := 1;
     found := false;
     repeat
          if ASCII_Arr(.Char_No,j.).ch = Key_Char then
            begin
               found := true;
               idx := j;
            end
          else
            begin
               j := j + 1;
            end;
     until found or (j > NUM_ASCII_CHAR);

      if (j > NUM_ASCII_CHAR) and (not found) then
           idx := 64;

end;




procedure Save_Binary_Prime_Num(idx: integer);

var  i : integer;

begin

     with Bool_Key_Arr(.Char_No.) do
     begin
```

```
      for i := 1 to MAX_BOOL_DIGIT do
        begin
           bool_prime(.i.) := ASCII_Arr(.Char_No,idx.).bool_prime(.i.);
        end;

      end;  {with}

end;




procedure Increment_Char_Pointer;

begin
    Char_No := Char_No + 1;
end;




function EX_OR (Bit_X, Bit_Y: boolean): boolean;

begin

      if Bit_X and Bit_Y then EX_OR := false

      else if Bit_X and (not Bit_Y) then EX_OR := true

      else if (not Bit_X) and Bit_Y then EX_OR := true

      else if (not Bit_X) and (not Bit_Y) then EX_OR := false;

end;




procedure First_Level_Ex_Oring;

    i, j : integer;
```

```
begin

     i := 1;
     j := 1;

     repeat
          EXOR1_Arr(.j.) := EX_OR(Bool_Key_Arr(.i.).bool_prime(.Bit_No.)
               ,Bool_Key_Arr(.i+1.).bool_prime(.Bit_No.));
          i := i + 2;
          j := j + 1;
     until (j > NUM_FIRST_EXOR);

end;
```

```
procedure Second_Level_Ex_Oring;

var  i, j : integer;

begin

     i := 1;
     j := 1;

     repeat
          EXOR2_Arr(.j.) := EX_OR(EXOR1_Arr(.i.), EXOR1_Arr(.i+1.));
          i := i + 2;
          j := j + 1;
     until (j > NUM_SECOND_EXOR);

end;
```

```
procedure Third_Level_Ex_Oring;
var  i, j : integer;

begin

     i := 1;
     j := 1;

     repeat
```

```
          EXOR3_Arr(.j.) := EX_OR(EXOR2_Arr(.i.), EXOR2_Arr(.i+1.));

          i := i + 2;
          j := j + 1;
     until (j > NUM_THIRD_EXOR);

end;




procedure Last_Ex_Oring_And_Store_An_Addr_Bit;

begin
     Hashed_Addr(.Bit_No.) := EX_OR(EXOR3_Arr(.1.), EXOR3_Arr(.2.));
end;




procedure Bool_To_Int_Convert (Bool_Arr: HASH_ADDR_TYPE; var Hash_Addr:
                               integer);

var  sum : integer;
     i : integer;

begin

     sum := 0;

     for i := MAX_BUCKET_ADDR_BITS+ADDR_OFFSET downto 1+ADDR_OFFSET do
       begin

          if Bool_Arr(.i.) then
               sum := 2 * sum + 1
          else
               sum := 2 * sum;

       end;

     Hash_Addr :=  sum;
end;
```

```
procedure Store_Key_In_Addressed_Bucket;

var  Key_Pt : LINK;
     i : integer;
     address : integer;

begin

     Bool_To_Int_Convert(Hashed_Addr, address);

     new(Key_Pt);

     for i := 1 to NUM_IDENT_CHAR do
       begin
          Key_Pt@.Key_Arr(.i.) := Key_Register(.i.);
       end;

     if DEBUG_FLAG then
       begin
          write('   KEY ATTRIBUTE : ');
          for i := 1 to NUM_IDENT_CHAR do
            begin
               write(Key_Register(.i.));
            end;

          writeln('   BUCKET ADDRESS : ',address:3);
          writeln;
       end;

     Key_Pt@.next := Bucket_Arr(.address.);

     Bucket_Arr(.address.) := Key_Pt;

end;




procedure Print_Keys_In_Each_Bucket;

var  i, j : integer;
     pt : LINK;
     count : integer;
```

```
begin

     for i := 0 to BUCKET_SIZE do
       begin
          count := 0;
          pt := Bucket_Arr(.i.);
          writeln;
          writeln('----- Bucket Number : ', i : 3,'----');
          writeln;

          while pt <> nil do
            begin

               write('      ');

               for j := 1 to NUM_IDENT_CHAR do
                 begin
                    write(pt@.Key_Arr(.j.));
                 end;

               pt := pt@.next;
               writeln;
               count := count + 1;

            end;  {while}

          writeln;

          Count_Arr(.i.) := count;

       end;  {for}

end;




procedure Print_Statistics;

var  i : integer;

     max, min, sum, Empty_Bucket, Non_Empty_Bucket, total: integer;

     Keys_11_To_20, Keys_21_To_30, Keys_Over_31 : integer;

     One_Key_Bucket, Two_Keys_Bucket, Three_Keys_Bucket,
     Four_Keys_Bucket, Five_Keys_Bucket, Six_Keys_Bucket,
     Seven_Keys_Bucket, Eight_Keys_Bucket, Nine_Keys_Bucket,
```

```
     Ten_Keys_Bucket : integer;

     variance, Var_Sum : real;
     mean : integer;

begin

     min := 99999;
     max := 0;

     sum := 0;

     Empty_Bucket := 0;
     Non_Empty_Bucket := 0;
     One_Key_Bucket := 0;
     Two_Keys_Bucket := 0;
     Three_Keys_Bucket := 0;
     Four_Keys_Bucket := 0;
     Five_Keys_Bucket := 0;
     Six_Keys_Bucket := 0;
     Seven_keys_Bucket := 0;
     Eight_Keys_Bucket := 0;
     Nine_Keys_Bucket := 0;
     Ten_Keys_Bucket := 0;

     Keys_11_To_20 := 0;
     Keys_21_To_30 := 0;
     Keys_Over_31 := 0;

     variance := 0;
     Var_Sum := 0;
     mean := MAX_NUM_KEYS div (BUCKET_SIZE + 1);

     for i := 0 to BUCKET_SIZE do
       begin

          write(' Bucket Number : ',i:3);
          writeln(' contains ', Count_Arr(.i.):3, ' keys.');
          writeln;

          if Count_Arr(.i.) > max then
               max := Count_Arr(.i.);

          if Count_Arr(.i.) < min then
               min := Count_Arr(.i.);

          if Count_Arr(.i.) = 0 then
               Empty_Bucket := Empty_Bucket + 1
          else
               Non_Empty_Bucket := Non_Empty_Bucket + 1;

          sum := sum + Count_Arr(.i.);

          if (Count_Arr(.i.) >= 1) and (Count_Arr(.i.) <= 10) then
          begin
```

```
        case Count_Arr(.i.) of
             1 : One_Key_Bucket := One_Key_Bucket + 1;
             2 : Two_Keys_Bucket := Two_Keys_Bucket + 1;
             3 : Three_Keys_Bucket := Three_Keys_Bucket + 1;
             4 : Four_Keys_Bucket := Four_Keys_Bucket + 1;
             5 : Five_Keys_Bucket := Five_Keys_Bucket + 1;
             6 : Six_Keys_Bucket := Six_Keys_Bucket + 1;
             7 : Seven_Keys_Bucket := Seven_Keys_Bucket + 1;
             8 : Eight_Keys_Bucket := Eight_Keys_Bucket + 1;
             9 : Nine_Keys_Bucket := Nine_Keys_Bucket + 1;
            10 : Ten_Keys_Bucket := Ten_Keys_Bucket + 1;
        end;

        end;   {if}


        if (Count_Arr(.i.) >= 11) and (Count_Arr(.i.) <= 20) then
             Keys_11_To_20 := Keys_11_To_20 + 1;

        if (Count_Arr(.i.) >= 21) and (Count_Arr(.i.) <= 30) then
             Keys_21_To_30 := Keys_21_To_30 + 1;

        if (Count_Arr(.i.) >= 31) then
             Keys_Over_31 := Keys_Over_31 + 1;

        Var_Sum := Var_Sum + sqr(Count_Arr(.i.) - mean)

      end;  {for}

    variance := Var_Sum / (BUCKET_SIZE + 1);

    writeln;
    writeln;
    writeln;
    writeln;
    writeln;
    writeln;
    writeln;
    writeln;
    writeln;
    writeln;
    writeln;
    writeln;
    writeln;
    writeln;
    writeln;
    writeln;
    writeln;
    writeln;
    writeln;
    writeln('********************* STATISTICS **********************');
    writeln;
    write(' CHARACTERS TO PRIME NUMBERS MAPPING HASHING ---');
    write('BIT ',ADDR_OFFSET+1:1,'-');
```

```
    writeln(ADDR_OFFSET+MAX_BUCKET_ADDR_BIT:1);
    writeln;
    writeln(' ALL OF THE 16 RAM CONTENTS ARE IDENTICAL IN THIS CASE.');
    writeln;
    writeln;
    writeln(' TOTAL NUMBER OF KEYS :                          ', sum:4);
    writeln;
    total := Empty_Bucket + Non_Empty_Bucket;
    writeln(' TOTAL NUMBER OF BUCKETS :                       ', total:4);
    writeln;
    writeln(' MAXIMUM NUMBER OF KEYS IN A BUCKET :            ', max:4);
    writeln;
    writeln(' MINIMUM NUMBER OF KEYS IN A BUCKET :            ', min:4);
    writeln;
    writeln(' VARIANCE (MEAN SQUARE DEVIATION) :        ',
            variance:8:2);
    writeln;
    writeln(' STANDARD DEVIATION :                            ',
            sqrt(variance):8:2);
    writeln;
    writeln(' TOTAL NUMBER OF NON-EMPTY BUCKETS :                        ',
    Non_Empty_Bucket:3,'  ',Non_Empty_Bucket*100 / total:7:1,' %');
    writeln;
    writeln(' TOTAL NUMBER OF EMPTY BUCKETS :                            ',
    Empty_Bucket:3,'  ',Empty_Bucket*100 / total:7:1,' %');
    writeln;
    writeln(' NUMBER OF BUCKETS WHICH CONTAIN ONE KEY :                  ',
    One_Key_Bucket:3,'  ',One_Key_Bucket*100 / total:7:1,' %');
    writeln;
    writeln(' NUMBER OF BUCKETS WHICH CONTAIN TWO KEYS :                 ',
    Two_Keys_Bucket:3,'  ',Two_Keys_Bucket*100 / total:7:1,' %');
    writeln;
    writeln(' NUMBER OF BUCKETS WHICH CONTAIN THREE KEYS :               ',
    Three_Keys_Bucket:3,'  ',Three_Keys_Bucket*100 / total:7:1,' %');
    writeln;
    writeln(' NUMBER OF BUCKETS WHICH CONTAIN FOUR KEYS :                ',
    Four_Keys_Bucket:3,'  ',Four_Keys_Bucket*100 / total:7:1,' %');
    writeln;
    writeln(' NUMBER OF BUCKETS WHICH CONTAIN FIVE KEYS :                ',
    Five_Keys_Bucket:3,'  ',Five_Keys_Bucket*100 / total:7:1,' %');
    writeln;
    writeln(' NUMBER OF BUCKETS WHICH CONTAIN SIX KEYS :                 ',
    Six_Keys_Bucket:3,'  ',Six_Keys_Bucket*100 / total:7:1,' %');
    writeln;
    writeln(' NUMBER OF BUCKETS WHICH CONTAIN SEVEN KEYS :               ',
    Seven_Keys_Bucket:3,'  ',Seven_Keys_Bucket*100 / total:7:1,' %');
    writeln;
    writeln(' NUMBER OF BUCKETS WHICH CONTAIN EIGHT KEYS :               ',
    Eight_Keys_Bucket:3,'  ',Eight_Keys_Bucket*100 / total:7:1,' %');
    writeln;
    writeln(' NUMBER OF BUCKETS WHICH CONTAIN NINE KEYS :                ',
    Nine_Keys_Bucket:3,'  ',Nine_Keys_Bucket*100 / total:7:1,' %');
    writeln;
    writeln(' NUMBER OF BUCKETS WHICH CONTAIN TEN KEYS :                 ',
    Ten_Keys_Bucket:3,'  ',Ten_Keys_Bucket*100 / total:7:1,' %');
```

```
     writeln;
     writeln(' NUMBER OF BUCKETS WHICH CONTAIN KEYS FROM 11 TO 20 : ',
     Keys_11_To_20:3,'  ',Keys_11_To_20*100 / total:7:1,' %');
     writeln;
     writeln(' NUMBER OF BUCKETS WHICH CONTAIN KEYS FROM 21 TO 30 : ',
     Keys_21_To_30:3,'  ',Keys_21_To_30*100 / total:7:1,' %');
     writeln;
     writeln(' NUMBER OF BUCKETS WHICH CONTAIN KEYS OVER 31 :       ',
     Keys_Over_31:3,'  ',Keys_Over_31*100 / total:7:1,' %');

end;



{*************** MAIN PROGRAM STARTS HERE ******************}

begin

     Initialization;

     {Read Keys one by one converting them to hashed addresses}

     for Key_No := 1 to MAX_NUM_KEYS do

       begin

         Init_While_Do_Loop;

         {Read character by character for a key looking up the
          corresponding prime number and convert it to binary number}

         while More_Chars_Left_For_Key do

           begin
              Read_A_Char;
              Look_Up_Char_In_Prime_Num_Table(index);
              Save_Binary_Prime_Num(index);
              Increment_Char_Pointer;
           end;


         {Get the first digit of the binary prime numbers which
          converted from a character, and do the Exclusive-Or operation
          to get the first bit for the hashed address. Repeat this
          process up to the last digit.}

         for Bit_No :=1 to MAX_BOOL_DIGIT do

           begin
              First_Level_Ex_Oring;
              Second_Level_Ex_Oring;
              Third_Level_Ex_Oring;
              Last_Ex_Oring_And_Store_An_Addr_Bit;
           end;
```

```
          {Using the resulting hashed address, store the input key in
           the corresponding bucket.}

          Store_Key_In_Addressed_Bucket;

        end; {outer for}

      {Print out all the keys in every hashed buckets.}

      Print_Keys_In_Each_Bucket;

      {Print out all the necessary statistics for an analysis.}

      Print_Statistics;

end.
```

FILE: MAPNU PASCAL A THE GEORGE WASHINGTON UNIVERSITY COMPUTER CENTER

```
program MAPNU (input, output);

const

     MAX_NUM_KEYS = 1024;
     NUM_IDENT_CHAR = 16;
     MAX_BUCKET_ADDR_BITS = 8;
     MAX_BOOL_DIGIT = 13;
     NUM_ASCII_CHAR = 70;

     NUM_FIRST_EXOR = 8;
     NUM_SECOND_EXOR = 4;
     NUM_THIRD_EXOR = 2;

     BUCKET_SIZE = 255;
     DEBUG_FLAG = true;
     DEBUG_ONLY = true;


type
     KEY_ARRAY_TYPE = array (.1..NUM_IDENT_CHAR.) of char;

     PRIME_BOOL_TYPE = array (.1..MAX_BOOL_DIGIT.) of boolean;

     CHAR_RECORD_TYPE = record
                          ch : char;
                          ASCII_num : integer;
                          prime_num : integer;
                          bool_prime : PRIME_BOOL_TYPE;
                        end;

     ASCII_TABLE = array (.1..NUM_IDENT_CHAR, 1..NUM_ASCII_CHAR.) of
                   CHAR_RECORD_TYPE;

     BOOL_PRIME_KEY_TYPE = array (.1..NUM_IDENT_CHAR.) of
                           record
                                bool_prime : PRIME_BOOL_TYPE;
                           end;

     FIRST_EXOR_ARR_TYPE = array (.1..NUM_FIRST_EXOR.) of boolean;

     SECOND_EXOR_ARR_TYPE = array (.1..NUM_SECOND_EXOR.) of boolean;

     THIRD_EXOR_ARR_TYPE = array (.1..NUM_THIRD_EXOR.) of boolean;

     HASHED_KEY_REG_TYPE = array (.1..MAX_BUCKET_ADDR_BITS.) of boolean;

     HASH_ADDR_TYPE = array (.1..MAX_BOOL_DIGIT.) of boolean;

     COUNT_KEY_TYPE = array (.0..BUCKET_SIZE.) of integer;

     LINK = @KEY_RECORD;
     KEY_RECORD = record
                    Key_Arr : KEY_ARRAY_TYPE;
```

```
                    next : LINK;
                end;

    BUCKET_POINTER_ARRAY = array (.0..BUCKET_SIZE.) of LINK;


var
    Key_Register : KEY_ARRAY_TYPE;

    ASCII_Arr : ASCII_TABLE;

    Key_No, Char_No, Bit_No : integer;

    Key_Char : char;

    index : integer;

    Bool_Key_Arr : BOOL_PRIME_KEY_TYPE;

    EXOR1_Arr : FIRST_EXOR_ARR_TYPE;
    EXOR2_Arr : SECOND_EXOR_ARR_TYPE;
    EXOR3_Arr : THIRD_EXOR_ARR_TYPE;

    Hashed_Addr : HASH_ADDR_TYPE;

    Bucket_Arr : BUCKET_POINTER_ARRAY;

    Count_Arr : COUNT_KEY_TYPE;

    Next_Seed : integer;

    OFFSET : integer;


procedure Int_To_Bool_Convert (number:integer; var Bool_Arr:
                                                PRIME_BOOL_TYPE);

var  i : integer;

begin
    for i := 1 to MAX_BOOL_DIGIT do
        Bool_Arr(.i.) := false;

    i := 1;
    while (number >= 2) and (i <= MAX_BOOL_DIGIT) do
      begin

        if (number mod 2) =1 then
            Bool_Arr(.i.) := true
        else
            Bool_Arr(.i.) := false;

        number := number div 2;
        i := i + 1;
```

```
        end;

      if (number = 1) and (i <= MAX_BOOL_DIGIT) then
           Bool_Arr(.i.) := true;

end;



procedure Initialization;

const
     char_divisor = 20;
     number_divisor = 10;

var  i, j, k: integer;
     number : integer;
     character : char;

begin

     Next_Seed := 5;

     readln(OFFSET);

     for i := 1 to NUM_IDENT_CHAR do
       begin

          for j := 1 to NUM_ASCII_CHAR do
            begin

               with ASCII_Arr(.i,j.) do
                 begin
                    ch := '?';
                    prime_num := 7;
                    ASCII_num := 777;

                    for k := 1 to MAX_BOOL_DIGIT do
                         bool_prime(.k.) := false;
                 end;
            end;

       end;

     for j := 1 to NUM_ASCII_CHAR do
       begin
          read(character);

          for i := 1 to NUM_IDENT_CHAR do
            begin
               ASCII_Arr(.i,j.).ch := character;
            end;
```

```
          if j mod char_divisor = 0 then
               readln;
     end;

     readln;



     for j :=1 to NUM_ASCII_CHAR do
       begin
          read(number);
          if number > 64 then number := number - 64;

          for i := 1 to NUM_IDENT_CHAR do
            begin
               ASCII_ARR(.i,j.).ASCII_num := number;
            end;

          if j mod number_divisor = 0 then
               readln;

       end;



  for i := 1 to NUM_IDENT_CHAR do
    begin
    writeln;
    writeln;
    write('  THE CONTENTS OF THE RANDOM ACCESS MEMORY FOR CHARACTER ');
    writeln('# ',i:1,' OF A KEY');
    writeln;
    writeln('  <ASCII CHAR> <ORD NO> <PRIME NO> <BINARY PRIME NUMBER>');

     for j := 1 to NUM_ASCII_CHAR do
       begin

          with ASCII_Arr(.i, j.) do
            begin

               read(prime_num);

               if j mod number_divisor = 0 then
                    readln;

               Int_To_Bool_Convert(prime_num, bool_prime);

               if (ch <> '?') and DEBUG_FLAG then
                 begin

                    write('      ',ch,' :        ');
                    write(ASCII_num:4,'      ');
                    write(prime_num:6,'     ');

                    for k := MAX_BOOL_DIGIT downto 1 do
```

```
                        begin
                           if bool_prime(.k.) then
                                write(1:2)
                           else
                                write(0:2);
                        end;

                        writeln;
                        writeln;
                    end;  {if}
               end;  {with}
          end;  {for}

     readln;
     end;    {for}

     for i := 0 to BUCKET_SIZE do
       begin
          Bucket_Arr(.i.) := nil;
       end;

end;




procedure Generate_Random_Num_Keys;

const
     MAX_DIGIT = 4;
     NUM_RANDOM_GEN = 4;

     TEN_THOUSAND = 10000;
     TEN = 10;

type
     NUM_KEY_TYPE = array (.1..NUM_IDENT_CHAR.) of integer;

var
     number : integer;
     Num_Key_Reg : NUM_KEY_TYPE;
     Start_Addr : integer;
     divisor : integer;
     digit : integer;
     i, j : integer;



function random (var seed : integer) : integer;

 onst
```

```
     MULTIPLIER = 25173;
     INCREMENT = 13849;
     MODULUS = 65536;

begin
     random := (MULTIPLIER * seed + INCREMENT) mod MODULUS;
end;




procedure Store_Chars_In_Key_Reg;

var
     i : integer;

begin
     for i := 1 to NUM_IDENT_CHAR do
       begin
          case Num_Key_Reg(.i.) of
               0 : Key_Register(.i.) := '0';
               1 : Key_Register(.i.) := '1';
               2 : Key_Register(.i.) := '2';
               3 : Key_Register(.i.) := '3';
               4 : Key_Register(.i.) := '4';
               5 : Key_Register(.i.) := '5';
               6 : Key_Register(.i.) := '6';
               7 : Key_Register(.i.) := '7';
               8 : Key_Register(.i.) := '8';
               9 : Key_Register(.i.) := '9';
          end;
       end;
end;


begin  {Generate_Random_Num_Keys}

     for i := 1 to NUM_RANDOM_GEN do
       begin
          divisor := TEN_THOUSAND;
          number := random(Next_Seed);
          Next_Seed := number;
          number := number mod divisor;

          for j := MAX_DIGIT downto 1 do
            begin
               divisor := divisor div TEN;
               digit := number div divisor;
               Start_Addr := (i - 1) * MAX_DIGIT;
               Num_Key_Reg(.Start_Addr+j.) := digit;
               number := number - (digit * divisor);
```

```
            end;
        end;

     Store_Chars_In_Key_Reg;
end;









procedure Init_While_Do_Loop;

var  i: integer;


begin

     Char_No := 1;

     for i := 1 to NUM_IDENT_CHAR do
       begin
          Key_Register(.i.) := ' ';
       end;

     Generate_Random_Num_Keys;

end;



function More_Chars_Left_For_Key : boolean;

begin

     if (Char_No > NUM_IDENT_CHAR) then
       begin
          More_Chars_Left_For_Key := false;
       end
     else
          More_Chars_Left_For_Key := true;
```

```
procedure Read_A_Char;

begin
     Key_Char := Key_Register(.Char_No.);
end;




procedure Look_Up_Char_In_Prime_Num_Table(var idx:integer);

var  found : boolean;
     j : integer;

begin
     j := 1;
     found := false;
     repeat
          if ASCII_Arr(.Char_No,j.).ch = Key_Char then
            begin
               found := true;
               idx := j;
            end
          else
            begin
               j := j + 1;
            end;
     until found or (j > NUM_ASCII_CHAR);

      if (j > NUM_ASCII_CHAR) and (not found) then
           idx := 64;

end;




procedure Save_Binary_Prime_Num(idx: integer);

var  i : integer;

begin

     with Bool_Key_Arr(.Char_No.) do
     begin
```

```
      for i := 1 to MAX_BOOL_DIGIT do
        begin
           bool_prime(.i.) := ASCII_Arr(.Char_No,idx.).bool_prime(.i.);
        end;

      end;  {with}

end;




procedure Increment_Char_Pointer;

begin
      Char_No := Char_No + 1;
end;




function EX_OR (Bit_X, Bit_Y: boolean): boolean;

begin

      if Bit_X and Bit_Y then EX_OR := false

      else if Bit_X and (not Bit_Y) then EX_OR := true

      else if (not Bit_X) and Bit_Y then EX_OR := true

      else if (not Bit_X) and (not Bit_Y) then EX_OR := false;

end;




procedure First_Level_Ex_Oring;

"     i, j : integer;
```

```
begin

     i := 1;
     j := 1;

     repeat
          EXOR1_Arr(.j.) := EX_OR(Bool_Key_Arr(.i.).bool_prime(.Bit_No.)
              ,Bool_Key_Arr(.i+1.).bool_prime(.Bit_No.));
          i := i + 2;
          j := j + 1;
     until (j > NUM_FIRST_EXOR);

end;
```

```
procedure Second_Level_Ex_Oring;

var  i, j : integer;

begin

     i := 1;
     j := 1;

     repeat
          EXOR2_Arr(.j.) := EX_OR(EXOR1_Arr(.i.), EXOR1_Arr(.i+1.));
          i := i + 2;
          j := j + 1;
     until (j > NUM_SECOND_EXOR);

end;
```

```
procedure Third_Level_Ex_Oring;
var  i, j : integer;

begin

     i := 1;
     j := 1;

     repeat
```

```
          EXOR3_Arr(.j.) := EX_OR(EXOR2_Arr(.i.), EXOR2_Arr(.i+1.));

          i := i + 2;
          j := j + 1;
     until (j > NUM_THIRD_EXOR);

end;




procedure Last_Ex_Oring_And_Store_An_Addr_Bit;

begin
     Hashed_Addr(.Bit_No.) := EX_OR(EXOR3_Arr(.1.), EXOR3_Arr(.2.));
end;




procedure Bool_To_Int_Convert (Bool_Arr: HASH_ADDR_TYPE; var Hash_Addr:
                               integer);

var  sum : integer;
     i : integer;

begin

     sum := 0;

     for i := MAX_BUCKET_ADDR_BITS+OFFSET downto 1+OFFSET do
       begin

          if Bool_Arr(.i.) then
               sum := 2 * sum + 1
          else
               sum := 2 * sum;

       end;

     Hash_Addr :=  sum;
end;
```

```
procedure Store_Key_In_Addressed_Bucket;

var  Key_Pt : LINK;
     i : integer;
     address : integer;

begin

     Bool_To_Int_Convert(Hashed_Addr, address);

     new(Key_Pt);

     for i := 1 to NUM_IDENT_CHAR do
       begin
          Key_Pt@.Key_Arr(.i.) := Key_Register(.i.);
       end;

     if DEBUG_FLAG then
       begin
          write('   KEY ATTRIBUTE : ');
          for i := 1 to NUM_IDENT_CHAR do
            begin
               write(Key_Register(.i.));
            end;

          writeln('   BUCKET ADDRESS : ',address:3);
          writeln;
       end;

     Key_Pt@.next := Bucket_Arr(.address.);

     Bucket_Arr(.address.) := Key_Pt;

end;




procedure Print_Keys_In_Each_Bucket;

var  i, j : integer;
     pt : LINK;
     count : integer;
```

```
beqin

     for i := 0 to BUCKET_SIZE do
       begin
          count := 0;
          pt := Bucket_Arr(.i.);
          writeln;
          writeln('----- Bucket Number : ', i : 3,'----');
          writeln;

          while pt <> nil do
            begin

               write('      ');

               for j := 1 to NUM_IDENT_CHAR do
                 begin
                    write(pt@.Key_Arr(.j.));
                 end;

               pt := pt@.next;
               writeln;
               count := count + 1;

            end;  {while}

          writeln;

          Count_Arr(.i.) := count;

       end;  {for}

end;




procedure Print_Statistics;

var  i : integer;

     max, min, sum, Empty_Bucket, Non_Empty_Bucket, total: integer;

     Keys_11_To_20, Keys_21_To_30, Keys_Over_31 : integer;

     One_Key_Bucket, Two_Keys_Bucket, Three_Keys_Bucket,
     Four_Keys_Bucket, Five_Keys_Bucket, Six_Keys_Bucket,
     Seven_Keys_Bucket, Eight_Keys_Bucket, Nine_Keys_Bucket,
```

```
     Ten_Keys_Bucket : integer;

     variance, Var_Sum : real;
     mean : integer;

begin

     min := 99999;
     max := 0;

     sum := 0;

     Empty_Bucket := 0;
     Non_Empty_Bucket := 0;
     One_Key_Bucket := 0;
     Two_Keys_Bucket := 0;
     Three_Keys_Bucket := 0;
     Four_Keys_Bucket := 0;
     Five_Keys_Bucket := 0;
     Six_Keys_Bucket := 0;
     Seven_keys_Bucket := 0;
     Eight_Keys_Bucket := 0;
     Nine_Keys_Bucket := 0;
     Ten_Keys_Bucket := 0;

     Keys_11_To_20 := 0;
     Keys_21_To_30 := 0;
     Keys_Over_31 := 0;

     variance := 0;
     Var_Sum := 0;
     mean := MAX_NUM_KEYS div (BUCKET_SIZE + 1);

     for i := 0 to BUCKET_SIZE do
       begin

          write(' Bucket Number : ',i:3);
          writeln(' contains ', Count_Arr(.i.):3, ' keys.');
          writeln;

          if Count_Arr(.i.) > max then
               max := Count_Arr(.i.);

          if Count_Arr(.i.) < min then
               min := Count_Arr(.i.);

          if Count_Arr(.i.) = 0 then
               Empty_Bucket := Empty_Bucket + 1
          else
               Non_Empty_Bucket := Non_Empty_Bucket + 1;

          sum := sum + Count_Arr(.i.);

          if (Count_Arr(.i.) >= 1) and (Count_Arr(.i.) <= 10) then
          begin
```

```
        case Count_Arr(.i.) of
             1 : One_Key_Bucket := One_Key_Bucket + 1;
             2 : Two_Keys_Bucket := Two_Keys_Bucket + 1;
             3 : Three_Keys_Bucket := Three_Keys_Bucket + 1;
             4 : Four_Keys_Bucket := Four_Keys_Bucket + 1;
             5 : Five_Keys_Bucket := Five_Keys_Bucket + 1;
             6 : Six_Keys_Bucket := Six_Keys_Bucket + 1;
             7 : Seven_Keys_Bucket := Seven_Keys_Bucket + 1;
             8 : Eight_Keys_Bucket := Eight_Keys_Bucket + 1;
             9 : Nine_Keys_Bucket := Nine_Keys_Bucket + 1;
            10 : Ten_Keys_Bucket := Ten_Keys_Bucket + 1;
        end;

        end;   {if}


        if (Count_Arr(.i.) >= 11) and (Count_Arr(.i.) <= 20) then
             Keys_11_To_20 := Keys_11_To_20 + 1;

        if (Count_Arr(.i.) >= 21) and (Count_Arr(.i.) <= 30) then
             Keys_21_To_30 := Keys_21_To_30 + 1;

        if (Count_Arr(.i.) >= 31) then
             Keys_Over_31 := Keys_Over_31 + 1;

        Var_Sum := Var_Sum + sqr(Count_Arr(.i.) - mean)

      end;  {for}

    variance := Var_Sum / (BUCKET_SIZE + 1);

    writeln;
    writeln;
    writeln;
    writeln;
    writeln;
    writeln;
    writeln;
    writeln;
    writeln;
    writeln;
    writeln;
    writeln;
    writeln;
    writeln;
    writeln;
    writeln;
    writeln;
    writeln;
    writeln;
    writeln('********************.STATISTICS **********************');
    writeln;
    write(' CHARACTERS TO PRIME NUMBERS MAPPING HASH---BITS ');
    writeln(1+OFFSET:1,'-',MAX_BUCKET_ADDR_BITS+OFFSET:1);
```

```
    writeln;
    writeln(' ALL OF THE 16 RAM CONTENTS ARE IDENTICAL IN THIS CASE.');
    writeln;
    writeln;
    writeln(' TOTAL NUMBER OF KEYS :                          ', sum:4);
    writeln;
    total := Empty_Bucket + Non_Empty_Bucket;
    writeln(' TOTAL NUMBER OF BUCKETS :                       ', total:4);
    writeln;
    writeln(' MAXIMUM NUMBER OF KEYS IN A BUCKET :            ', max:4);
    writeln;
    writeln(' MINIMUM NUMBER OF KEYS IN A BUCKET :            ', min:4);
    writeln;
    writeln(' VARIANCE (MEAN SQUARE DEVIATION) :          ',
            variance:8:2);
    writeln;
    writeln(' STANDARD DEVIATION :                            ',
            sqrt(variance):8:2);
    writeln;
    writeln(' TOTAL NUMBER OF NON-EMPTY BUCKETS :                        ',
    Non_Empty_Bucket:3,'  ',Non_Empty_Bucket*100 / total:7:1,' %');
    writeln;
    writeln(' TOTAL NUMBER OF EMPTY BUCKETS :                            ',
    Empty_Bucket:3,'  ',Empty_Bucket*100 / total:7:1,' %');
    writeln;
    writeln(' NUMBER OF BUCKETS WHICH CONTAIN ONE KEY :                  ',
    One_Key_Bucket:3,'  ',One_Key_Bucket*100 / total:7:1,' %');
    writeln;
    writeln(' NUMBER OF BUCKETS WHICH CONTAIN TWO KEYS :                 ',
    Two_Keys_Bucket:3,'  ',Two_Keys_Bucket*100 / total:7:1,' %');
    writeln;
    writeln(' NUMBER OF BUCKETS WHICH CONTAIN THREE KEYS :               ',
    Three_Keys_Bucket:3,'  ',Three_Keys_Bucket*100 / total:7:1,' %');
    writeln;
    writeln(' NUMBER OF BUCKETS WHICH CONTAIN FOUR KEYS :                ',
    Four_Keys_Bucket:3,'  ',Four_Keys_Bucket*100 / total:7:1,' %');
    writeln;
    writeln(' NUMBER OF BUCKETS WHICH CONTAIN FIVE KEYS :                ',
    Five_Keys_Bucket:3,'  ',Five_Keys_Bucket*100 / total:7:1,' %');
    writeln;
    writeln(' NUMBER OF BUCKETS WHICH CONTAIN SIX KEYS :                 ',
    Six_Keys_Bucket:3,'  ',Six_Keys_Bucket*100 / total:7:1,' %');
    writeln;
    writeln(' NUMBER OF BUCKETS WHICH CONTAIN SEVEN KEYS :               ',
    Seven_Keys_Bucket:3,'  ',Seven_Keys_Bucket*100 / total:7:1,' %');
    writeln;
    writeln(' NUMBER OF BUCKETS WHICH CONTAIN EIGHT KEYS :               ',
    Eight_Keys_Bucket:3,'  ',Eight_Keys_Bucket*100 / total:7:1,' %');
    writeln;
    writeln(' NUMBER OF BUCKETS WHICH CONTAIN NINE KEYS :                ',
    Nine_Keys_Bucket:3,'  ',Nine_Keys_Bucket*100 / total:7:1,' %');
    writeln;
    writeln(' NUMBER OF BUCKETS WHICH CONTAIN TEN KEYS :                 ',
    Ten_Keys_Bucket:3,'  ',Ten_Keys_Bucket*100 / total:7:1,' %');
    writeln;
```

```
     writeln(' NUMBER OF BUCKETS WHICH CONTAIN KEYS FROM 11 TO 20 : ',
     Keys_11_To_20:3,'  ',Keys_11_To_20*100 / total:7:1,' %');
     writeln;
     writeln(' NUMBER OF BUCKETS WHICH CONTAIN KEYS FROM 21 TO 30 : ',
     Keys_21_To_30:3,'  ',Keys_21_To_30*100 / total:7:1,' %');
     writeln;
     writeln(' NUMBER OF BUCKETS WHICH CONTAIN KEYS OVER 31 :       ',
     Keys_Over_31:3,'  ',Keys_Over_31*100 / total:7:1,' %');

end;



{**************** MAIN PROGRAM STARTS HERE ******************}

begin

     Initialization;

     {Read Keys one by one converting them to hashed addresses}

     for Key_No := 1 to MAX_NUM_KEYS do

       begin

         Init_While_Do_Loop;

         {Read character by character for a key looking up the
          corresponding prime number and convert it to binary number}

         while More_Chars_Left_For_Key do

           begin
              Read_A_Char;
              Look_Up_Char_In_Prime_Num_Table(index);
              Save_Binary_Prime_Num(index);
              Increment_Char_Pointer;
           end;


         {Get the first digit of the binary prime numbers which
          converted from a character, and do the Exclusive-Or operation
          to get the first bit for the hashed address. Repeat this
          process up to the last digit.}

         for Bit_No :=1 to MAX_BOOL_DIGIT do

           begin
              First_Level_Ex_Oring;
              Second_Level_Ex_Oring;
              Third_Level_Ex_Oring;
              Last_Ex_Oring_And_Store_An_Addr_Bit;
           end;
```

```
          {Using the resulting hashed address, store the input key in
           the corresponding bucket.}

          Store_Key_In_Addressed_Bucket;

        end; {outer for}

      {Print out all the keys in every hashed buckets.}

      Print_Keys_In_Each_Bucket;

      {Print out all the necessary statistics for an analysis.}

      Print_Statistics;

end.
```

FILE: MAPRAND PASCAL A THE GEORGE WASHINGTON UNIVERSITY COMPUTER CENTER

```
//ime=1
// EXEC PASCG
   'SIN DD *
program FAST_HW_HASHING (input, output);


const
     MAX_NUM_KEYS = 1024;
     NUM_IDENT_CHAR = 16;
     MAX_BUCKET_ADDR_BITS = 8;
     MAX_BOOL_DIGIT = 13;
     NUM_ASCII_CHAR = 70;

     NUM_FIRST_EXOR = 8;
     NUM_SECOND_EXOR = 4;
     NUM_THIRD_EXOR = 2;

     BUCKET_SIZE = 255;

     DEBUG_FLAG = true;
     DEBUG_ONLY = true;


type
     KEY_ARRAY_TYPE = array (.1..NUM_IDENT_CHAR.) of char;

     PRIME_BOOL_TYPE = array (.1..MAX_BOOL_DIGIT.) of boolean;

     CHAR_RECORD_TYPE = record
                          ch : char;
                          ASCII_num : integer;
                          prime_num : integer;
                          bool_prime : PRIME_BOOL_TYPE;
                        end;

     ASCII_TABLE = array (.1..NUM_IDENT_CHAR, 1..NUM_ASCII_CHAR.) of
                   CHAR_RECORD_TYPE;

     BOOL_PRIME_KEY_TYPE = array (.1..NUM_IDENT_CHAR.) of
                           record
                                bool_prime : PRIME_BOOL_TYPE;
                           end;

     FIRST_EXOR_ARR_TYPE = array (.1..NUM_FIRST_EXOR.) of boolean;

     SECOND_EXOR_ARR_TYPE = array (.1..NUM_SECOND_EXOR.) of boolean;

     THIRD_EXOR_ARR_TYPE = array (.1..NUM_THIRD_EXOR.) of boolean;

     HASHED_KEY_REG_TYPE = array (.1..MAX_BUCKET_ADDR_BITS.) of boolean;

     HASH_ADDR_TYPE = array (.1..MAX_BOOL_DIGIT.) of boolean;

     COUNT_KEY_TYPE = array (.0..BUCKET_SIZE.) of integer;
```

```
     LINK = @KEY_RECORD;
     KEY_RECORD = record
                     Key_Arr : KEY_ARRAY_TYPE;
                     next : LINK;
                  end;

     BUCKET_POINTER_ARRAY = array (.0..BUCKET_SIZE.) of LINK;



var
     Key_Register : KEY_ARRAY_TYPE;

     ASCII_Arr : ASCII_TABLE;

     Key_No, Char_No, Bit_No : integer;

     Key_Char : char;

     index : integer;

     Bool_Key_Arr : BOOL_PRIME_KEY_TYPE;

     EXOR1_Arr : FIRST_EXOR_ARR_TYPE;
     EXOR2_Arr : SECOND_EXOR_ARR_TYPE;
     EXOR3_Arr : THIRD_EXOR_ARR_TYPE;

     Hashed_Addr : HASH_ADDR_TYPE;

     Bucket_Arr : BUCKET_POINTER_ARRAY;

     Count_Arr : COUNT_KEY_TYPE;




procedure Int_To_Bool_Convert (number:integer; var Bool_Arr:
                                                  PRIME_BOOL_TYPE);

var  i : integer;

begin
     for i := 1 to MAX_BOOL_DIGIT do
          Bool_Arr(.i.) := false;

     i := 1;
     while (number >= 2) and (i <= MAX_BOOL_DIGIT) do
       begin

          if (number mod 2) =1 then
               Bool_Arr(.i.) := true
          else
               Bool_Arr(.i.) := false;
```

```
          number := number div 2;
          i := i + 1;
        end;

     if (number = 1) and (i <= MAX_BOOL_DIGIT) then
          Bool_Arr(.i.) := true;

end;




procedure Initialization;

const
     char_divisor = 20;
     number_divisor = 10;

var  i, j, k: integer;
     number : integer;
     character : char;

begin

     for i := 1 to NUM_IDENT_CHAR do
       begin

          for j := 1 to NUM_ASCII_CHAR do
            begin

               with ASCII_Arr(.i,j.) do
                 begin
                    ch := '?';
                    prime_num := 7;
                    ASCII_num := 777;

                    for k := 1 to MAX_BOOL_DIGIT do
                         bool_prime(.k.) := false;
                 end;
            end;

       end;

     for j := 1 to NUM_ASCII_CHAR do
       begin
          read(character);

          for i := 1 to NUM_IDENT_CHAR do
            begin
               ASCII_Arr(.i,j.).ch := character;
            end;

          if j mod char_divisor = 0 then
               readln;
```

```
    end;

    readln;


    for j :=1 to NUM_ASCII_CHAR do
      begin
         read(number);
         if number > 64 then number := number - 64;

         for i := 1 to NUM_IDENT_CHAR do
           begin
              ASCII_ARR(.i,j.).ASCII_num := number;
           end;

         if j mod number_divisor = 0 then
              readln;

      end;

    if DEBUG_ONLY then

  for i := 1 to NUM_IDENT_CHAR do
    begin
    writeln;
    writeln;
    write('  THE CONTENTS OF THE RANDOM ACCESS MEMORY FOR CHARACTER ');
    writeln('# ',i:1,' OF A KEY');
    writeln;
    writeln('  <ASCII CHAR> <ORD NO> <PRIME NO> <BINARY PRIME NUMBER>');

     for j := 1 to NUM_ASCII_CHAR do
       begin

          with ASCII_Arr(.i, j.) do
            begin

               read(prime_num);

               if j mod number_divisor = 0 then
                    readln;

               Int_To_Bool_Convert(prime_num, bool_prime);

               if (ch <> '?') and DEBUG_FLAG then
                 begin

                    write('     ',ch,' :       ');
                    write(ASCII_num:4,'     ');
                    write(prime_num:6,'    ');

                    for k := MAX_BOOL_DIGIT downto 1 do
                      begin
                         if bool_prime(.k.) then
```

```
                              write(1:2)
                         else
                              write(0:2);
                        end;

                        writeln;
                        writeln;
                   end;  {if}
              end;  {with}
        end;  {for}

     readln;
     end;    {for}

     for i := 0 to BUCKET_SIZE do
       begin
          Bucket_Arr(.i.) := nil;
       end;

end;



procedure Init_While_Do_Loop;

var  i: integer;


begin

     Char_No := 1;

     for i := 1 to NUM_IDENT_CHAR do
       begin
          Key_Register(.i.) := ' ';
       end;
end;



function More_Chars_Left_For_Key : boolean;

begin

     if ((Char_No > NUM_IDENT_CHAR) or eoln) or eof then
       begin
          More_Chars_Left_For_Key := false;
          readln;
       end
     else
          More_Chars_Left_For_Key := true;
;
```

```
procedure Read_A_Char;

begin
     read(Key_Char);
     Key_Register(.Char_No.) := Key_Char;
end;




procedure Look_Up_Char_In_Prime_Num_Table(var idx:integer);

var  found : boolean;
     j : integer;

begin
     j := 1;
     found := false;
     repeat
          if ASCII_Arr(.Char_No,j.).ch = Key_Char then
            begin
               found := true;
               idx := j;
            end
          else
            begin
               j := j + 1;
            end;
     until found or (j > NUM_ASCII_CHAR);

      if (j > NUM_ASCII_CHAR) and (not found) then
           idx := 64;

end;




procedure Save_Binary_Prime_Num(idx: integer);

var  i : integer;

begin

     with Bool_Key_Arr(.Char_No.) do
```

```
      begin

      for i := 1 to MAX_BOOL_DIGIT do
        begin
           bool_prime(.i.) := ASCII_Arr(.Char_No,idx.).bool_prime(.i.);
        end;

      end;  {with}

end;




procedure Increment_Char_Pointer;

begin
      Char_No := Char_No + 1;
end;




function EX_OR (Bit_X, Bit_Y: boolean): boolean;

begin

      if Bit_X and Bit_Y then EX_OR := false

      else if Bit_X and (not Bit_Y) then EX_OR := true

      else if (not Bit_X) and Bit_Y then EX_OR := true

      else if (not Bit_X) and (not Bit_Y) then EX_OR := false;

end;




procedure First_Level_Ex_Oring;

  i, j : integer;
```

```
begin

     i := 1;
     j := 1;

     repeat
          EXOR1_Arr(.j.) := EX_OR(Bool_Key_Arr(.i.).bool_prime(.Bit_No.)
              ,Bool_Key_Arr(.i+1.).bool_prime(.Bit_No.));
          i := i + 2;
          j := j + 1;
     until (j > NUM_FIRST_EXOR);

end;
```

```
procedure Second_Level_Ex_Oring;

var  i, j : integer;

begin

     i := 1;
     j := 1;

     repeat
          EXOR2_Arr(.j.) := EX_OR(EXOR1_Arr(.i.), EXOR1_Arr(.i+1.));
          i := i + 2;
          j := j + 1;
     until (j > NUM_SECOND_EXOR);

end;
```

```
procedure Third_Level_Ex_Oring;
var  i, j : integer;

begin

     i := 1;
     j := 1;
```

```
     repeat
          EXOR3_Arr(.j.) := EX_OR(EXOR2_Arr(.i.), EXOR2_Arr(.i+1.));

          i := i + 2;
          j := j + 1;
     until (j > NUM_THIRD_EXOR);

end;




procedure Last_Ex_Oring_And_Store_An_Addr_Bit;

begin
     Hashed_Addr(.Bit_No.) := EX_OR(EXOR3_Arr(.1.), EXOR3_Arr(.2.));
end;




procedure Bool_To_Int_Convert (Bool_Arr: HASH_ADDR_TYPE; var Hash_Addr:
                               integer);

var  sum : integer;
     i : integer;

begin

     sum := 0;

     for i := MAX_BUCKET_ADDR_BITS+1 downto 1+1 do
       begin

          if Bool_Arr(.i.) then
               sum := 2 * sum + 1
          else
               sum := 2 * sum;

       end;

     Hash_Addr :=  sum;
end;
```

```
procedure Store_Key_In_Addressed_Bucket;

var  Key_Pt : LINK;
     i : integer;
     address : integer;

begin

     Bool_To_Int_Convert(Hashed_Addr, address);

     new(Key_Pt);

     for i := 1 to NUM_IDENT_CHAR do
       begin
          Key_Pt@.Key_Arr(.i.) := Key_Register(.i.);
       end;

     if DEBUG_FLAG then
       begin
          write('   KEY ATTRIBUTE : ');
          for i := 1 to NUM_IDENT_CHAR do
            begin
               write(Key_Register(.i.));
            end;

          writeln('   BUCKET ADDRESS : ',address:3);
          writeln;
       end;

     Key_Pt@.next := Bucket_Arr(.address.);

     Bucket_Arr(.address.) := Key_Pt;

end;




procedure Print_Keys_In_Each_Bucket;

var  i, j : integer;
     pt : LINK;
     count : integer;
```

```
    n

     for i := 0 to BUCKET_SIZE do
       begin
          count := 0;
          pt := Bucket_Arr(.i.);
          writeln;
          writeln('----- Bucket Number : ', i : 3,'----');
          writeln;

          while pt <> nil do
            begin

               write('      ');

               for j := 1 to NUM_IDENT_CHAR do
                 begin
                    write(pt@.Key_Arr(.j.));
                 end;

               pt := pt@.next;
               writeln;
               count := count + 1;

            end;  {while}

          writeln;

          Count_Arr(.i.) := count;

       end;  {for}

end;




procedure Print_Statistics;

var  i : integer;

     max, min, sum, Empty_Bucket, Non_Empty_Bucket, total: integer;

     Keys_11_To_20, Keys_21_To_30, Keys_Over_31 : integer;

     One_Key_Bucket, Two_Keys_Bucket, Three_Keys_Bucket,
     Four_Keys_Bucket, Five_Keys_Bucket, Six_Keys_Bucket,
```

```
     Seven_Keys_Bucket, Eight_Keys_Bucket, Nine_Keys_Bucket,
     Ten_Keys_Bucket : integer;

     variance, Var_Sum : real;
     mean : integer;

begin

     min := 99999;
     max := 0;

     sum := 0;

     Empty_Bucket := 0;
     Non_Empty_Bucket := 0;
     One_Key_Bucket := 0;
     Two_Keys_Bucket := 0;
     Three_Keys_Bucket := 0;
     Four_Keys_Bucket := 0;
     Five_Keys_Bucket := 0;
     Six_Keys_Bucket := 0;
     Seven_keys_Bucket := 0;
     Eight_Keys_Bucket := 0;
     Nine_Keys_Bucket := 0;
     Ten_Keys_Bucket := 0;

     Keys_11_To_20 := 0;
     Keys_21_To_30 := 0;
     Keys_Over_31 := 0;

     variance := 0;
     Var_Sum := 0;
     mean := MAX_NUM_KEYS div (BUCKET_SIZE + 1);

     for i := 0 to BUCKET_SIZE do
       begin

          write(' Bucket Number : ',i:3);
          writeln(' contains ', Count_Arr(.i.):3, ' keys.');
          writeln;

          if Count_Arr(.i.) > max then
               max := Count_Arr(.i.);

          if Count_Arr(.i.) < min then
               min := Count_Arr(.i.);

          if Count_Arr(.i.) = 0 then
               Empty_Bucket := Empty_Bucket + 1
          else
               Non_Empty_Bucket := Non_Empty_Bucket + 1;

          sum := sum + Count_Arr(.i.);

          if (Count_Arr(.i.) >= 1) and (Count_Arr(.i.) <= 10) then
```

```
        begin

        case Count_Arr(.i.) of
             1 : One_Key_Bucket := One_Key_Bucket + 1;
             2 : Two_Keys_Bucket := Two_Keys_Bucket + 1;
             3 : Three_Keys_Bucket := Three_Keys_Bucket + 1;
             4 : Four_Keys_Bucket := Four_Keys_Bucket + 1;
             5 : Five_Keys_Bucket := Five_Keys_Bucket + 1;
             6 : Six_Keys_Bucket := Six_Keys_Bucket + 1;
             7 : Seven_Keys_Bucket := Seven_Keys_Bucket + 1;
             8 : Eight_Keys_Bucket := Eight_Keys_Bucket + 1;
             9 : Nine_Keys_Bucket := Nine_Keys_Bucket + 1;
            10 : Ten_Keys_Bucket := Ten_Keys_Bucket + 1;
        end;

        end;   {if}


        if (Count_Arr(.i.) >= 11) and (Count_Arr(.i.) <= 20) then
             Keys_11_To_20 := Keys_11_To_20 + 1;

        if (Count_Arr(.i.) >= 21) and (Count_Arr(.i.) <= 30) then
             Keys_21_To_30 := Keys_21_To_30 + 1;

        if (Count_Arr(.i.) >= 31) then
             Keys_Over_31 := Keys_Over_31 + 1;

        Var_Sum := Var_Sum + sqr(Count_Arr(.i.) - mean)

    end;  {for}

    variance := Var_Sum / (BUCKET_SIZE + 1);

    writeln;
    writeln;
    writeln;
    writeln;
    writeln;
    writeln;
    writeln;
    writeln;
    writeln;
    writeln;
    writeln;
    writeln;
    writeln;
    writeln;
    writeln;
    writeln;
    writeln;
    writeln;
    writeln;
    writeln;
    writeln('********************* STATISTICS **********************');
    writeln;
    writeln(' CHARACTERS TO PRIME NUMBERS MAPPING HASHING (BIT 2-9)');
```

```
      writeln;
      writeln(' ALL OF THE 16 RAM CONTENTS ARE IDENTICAL IN THIS CASE.');
      writeln;
      writeln;
      writeln(' TOTAL NUMBER OF KEYS :                          ', sum:4);
      writeln;
      total := Empty_Bucket + Non_Empty_Bucket;
      writeln(' TOTAL NUMBER OF BUCKETS :                       ', total:4);
      writeln;
      writeln(' MAXIMUM NUMBER OF KEYS IN A BUCKET :            ', max:4);
      writeln;
      writeln(' MINIMUM NUMBER OF KEYS IN A BUCKET :            ', min:4);
      writeln;
      writeln(' VARIANCE (MEAN SQUARE DEVIATION) :          ',
              variance:8:2);
      writeln;
      writeln(' STANDARD DEVIATION :                        ',
              sqrt(variance):8:2);
      writeln;
      writeln(' TOTAL NUMBER OF NON-EMPTY BUCKETS :                    ',
     Non_Empty_Bucket:3,'  ',Non_Empty_Bucket*100 / total:7:1,' %');
      writeln;
      writeln(' TOTAL NUMBER OF EMPTY BUCKETS :                        ',
     Empty_Bucket:3,'  ',Empty_Bucket*100 / total:7:1,' %');
      writeln;
      writeln(' NUMBER OF BUCKETS WHICH CONTAIN ONE KEY :              ',
     One_Key_Bucket:3,'  ',One_Key_Bucket*100 / total:7:1,' %');
      writeln;
      writeln(' NUMBER OF BUCKETS WHICH CONTAIN TWO KEYS :             ',
     Two_Keys_Bucket:3,'  ',Two_Keys_Bucket*100 / total:7:1,' %');
      writeln;
      writeln(' NUMBER OF BUCKETS WHICH CONTAIN THREE KEYS :           ',
     Three_Keys_Bucket:3,'  ',Three_Keys_Bucket*100 / total:7:1,' %');
      writeln;
      writeln(' NUMBER OF BUCKETS WHICH CONTAIN FOUR KEYS :            ',
     Four_Keys_Bucket:3,'  ',Four_Keys_Bucket*100 / total:7:1,' %');
      writeln;
      writeln(' NUMBER OF BUCKETS WHICH CONTAIN FIVE KEYS :            ',
     Five_Keys_Bucket:3,'  ',Five_Keys_Bucket*100 / total:7:1,' %');
      writeln;
      writeln(' NUMBER OF BUCKETS WHICH CONTAIN SIX KEYS :             ',
     Six_Keys_Bucket:3,'  ',Six_Keys_Bucket*100 / total:7:1,' %');
      writeln;
      writeln(' NUMBER OF BUCKETS WHICH CONTAIN SEVEN KEYS :           ',
     Seven_Keys_Bucket:3,'  ',Seven_Keys_Bucket*100 / total:7:1,' %');
      writeln;
      writeln(' NUMBER OF BUCKETS WHICH CONTAIN EIGHT KEYS :           ',
     Eight_Keys_Bucket:3,'  ',Eight_Keys_Bucket*100 / total:7:1,' %');
      writeln;
      writeln(' NUMBER OF BUCKETS WHICH CONTAIN NINE KEYS :            ',
     Nine_Keys_Bucket:3,'  ',Nine_Keys_Bucket*100 / total:7:1,' %');
      writeln;
      writeln(' NUMBER OF BUCKETS WHICH CONTAIN TEN KEYS :             ',
     Ten_Keys_Bucket:3,'  ',Ten_Keys_Bucket*100 / total:7:1,' %');
      writeln;
```

```
     writeln(' NUMBER OF BUCKETS WHICH CONTAIN KEYS FROM 11 TO 20 : ',
     Keys_11_To_20:3,'  ',Keys_11_To_20*100 / total:7:1,' %');
     writeln;
     writeln(' NUMBER OF BUCKETS WHICH CONTAIN KEYS FROM 21 TO 30 : ',
     Keys_21_To_30:3,'  ',Keys_21_To_30*100 / total:7:1,' %');
     writeln;
     writeln(' NUMBER OF BUCKETS WHICH CONTAIN KEYS OVER 31 :       ',
     Keys_Over_31:3,'  ',Keys_Over_31*100 / total:7:1,' %');

end;



{**************** MAIN PROGRAM STARTS HERE ******************}

begin

     Initialization;

     {Read Keys one by one converting them to hashed addresses}

     for Key_No := 1 to MAX_NUM_KEYS do

       begin

         Init_While_Do_Loop;

         {Read character by character for a key looking up the
          corresponding prime number and convert it to binary number}

         while More_Chars_Left_For_Key do

           begin
              Read_A_Char;
              Look_Up_Char_In_Prime_Num_Table(index);
              Save_Binary_Prime_Num(index);
              Increment_Char_Pointer;
           end;


         {Get the first digit of the binary prime numbers which
          converted from a character, and do the Exclusive-Or operation
          to get the first bit for the hashed address. Repeat this
          process up to the last digit.}

         for Bit_No :=1 to MAX_BOOL_DIGIT do

           begin
              First_Level_Ex_Oring;
              Second_Level_Ex_Oring;
              Third_Level_Ex_Oring;
              Last_Ex_Oring_And_Store_An_Addr_Bit;
           end;
```

```
          {Using the resulting hashed address, store the input key in
           the corresponding bucket.}

          Store_Key_In_Addressed_Bucket;

        end; {outer for}

      {Print out all the keys in every hashed buckets.}

      Print_Keys_In_Each_Bucket;

      {Print out all the necessary statistics for an analysis.}

      Print_Statistics;

end.
//GO.SYSIN DD *
ABCDEFGHIJKLMNOPQRST
UVWXYZ01234 abcdefgh
ijklmnopqrstuvwxyz56
789#.,'-_$
65 66 67 68 69 70 71 72 73 74
75 76 77 78 79 80 81 82 83 84
85 86 87 88 89 90 48 49 50 51
52 32 97 98 99 100 101 102 103 104
105 106 107 108 109 110 111 112 113 114
115 116 117 118 119 120 121 122 53 54
55 56 57 35 46 44 39 45 95 36
   9 2063 7927 5087 3583 1307 7687 1523 3643  223
 103  523 7129 5669 3229 7789 8527 4969 2549 1721
3469 5189 5563 5981 4021 3187 3167 4409 6827 1109
 461 6323  769 4363 4801 1481 6367 7963 1747 2203
5081 3083 6547 3727 7069 2887 2221 8009 1987 2161
2683 4583 4127 7541 6361  967 5627 2309 4787 6581
8287  743 5347 3709 6763 1021 1949 5449 3041 3907
Danny Shin
Danny Shin
Anderson J
Anderson J A
Anderson J C
Anderson J D
Anderson J H
Anderson J I
Anderson J J
Anderson J K
Anderson J L
Anderson J M
Anderson J R
Anderson J T
Anderson J W
Hudson Charles S
Minjack Dolores
Sokkappa Balraj G
W  din Peter
 ernstrom John R Jr.
```

FILE: MAPRAND PASCAL A THE GEORGE WASHINGTON UNIVERSITY COMPUTER CENTER

Brooks Rose A
Diamonstein Kevin M
jins Howard H
Labarski Alfred
Merriman Allen R
Ravera Robert
Tate Ada Cecelia
Keebler Gregory
Fuller Harry
Campbell William P
Einhorn Annie Lee
Ludwig August W
Puckly Michael J
Stumbo Richard M
Moran Jearld V
Lewis Jack
Cooper Bruce Michael
Bush Chester k
Nolan Donald
Snider Giovanni
McMillan John C
Haynes Michael
Chrismon Alan C
Horton Adona I
Ruffner Ernest L
Thompson John F
Nelson Kristine K
elean Susan
nandez Ricardo
Holzman Charlotte
Meruvia Louisa
Peeples Deborah
Shoot Tony
Thornton Elizabeth
Yasuda Akiharu
McGrath Michel V
Irwin Harry E
Roberts Tracey L
Stull Bernard
Williamson David W
McBride Stewart
Fretz Tammy
Carlin Joyce Y
Glaser Ronald
Jameson Geoffrey B
Paolozzi Thomas
Rinkerman Gary
Stuart Alfred D
Lewis Kathleen
Gunn George D
Cummings Jerome J
Askins Paul W
Goodwin Everett C
gum Wade
el Christopher

FILE: MAPRAND PASCAL A THE GEORGE WASHINGTON UNIVERSITY COMPUTER CENTER

Rugbart Dennis
Irvin Wayne M
·one Jolene Penick
ch.1sman Amanda M
Beard Donaldson E
Loftus Susan
Mitchell Gilbert H
Sarahan Josephine
Tracy Harold
Wolff Raymond H
Northrop Carl W
Borders John M
Dowd Robert
Gomez Raphael
Rodgers Philip
Venable Albert
Hill Brenda
Culver Ralph
Bean Harold G
Cox Janice
Gonet Edw A
Hopkins Frank J
Lazo Andres
Mertens Frank E
Phipps Barbara
Snider Adrienne
McGee Gloria J
Holtzman Scott
ch Richard A
Beane Debra E
Gawsett Christy
Lazarus Allan R
Ragan Thomas Edward
Spencer Helen K
West Clarence L
Quinn Charles
LeBlanc Christopher M
Fletcher Herbert
Burnham Gary
Ali Mohammad
Dorsey Ann
Flester Toni
Guernsey David
Hatcher Amelia
Cook Benjamin
Cavazos Anthony A
Demory Calvin
Gonsky Robert
Herwitz Jonathan
Kanoza Douglas
Lazzari Fernando
Lose Graydon I
Mervis Bob
M sen Cliff
illiam Harold

FILE: MAPRAND PASCAL A THE GEORGE WASHINGTON UNIVERSITY COMPUTER CENTER

Rawson Bruce S
Saenz Adolph B
ler Chris
Glasse Mark
Dorsett Charmaine
Gawkowski Amy
Greber Alan
Malmberg Allan F
Puff paul
Nyborg Ellen J
Jobber Claude R
Chatelain Fred D
Allam Anthony
Masoller Edwardo
Naeher Edw W
Sadur A
Weschler Kenneth J
Musser Carl W
Larco Lorianne F
Dowdle Billy
Christensen A B
Winter Arthur
Anderton Frank
Knater Hershel
McLaughlin Brain M
Rawlins Andrew
Weisberger Doug
Voegel Eric P
na Christine
Parrado Maria
Hollomon Duncan
Finkenbinder Thomas C
Christen Roy E
Azar Abdul Hakim
Lawhorn Chas G
Mertins Gerhard G
Phongsavan Chamrath
Shurts Robert
Yolton L William
Mallgraf William
Hudnall Charles W
Gernando Eleanor
Brownlee Bruce
Infante Manuel
Merz Albert H
Potter Cynthia
Walthall Benj R
Woodall Gary
Noshirvani Hamid F
Loucas Nicholas S
Hiles Andrew
Dorosin Robert T
Balthrop Deborah
lord Brian
rwit Paul

FILE: MAPRAND PASCAL A THE GEORGE WASHINGTON UNIVERSITY COMPUTER CENTER

Luebeck John B
Nolton Becky
stein Albert A
opchak Chris
Linders Michael
Hale Thomas M
Brown Milton
Genovese Carroll F
Rawson Marshall
Vouvalis Nausika G
Kuan Perng
Marie Louis
Chrisinger Edw L
Ballou Catherine H
Kautz Arthur C
Roslyn Albert S
Willey Anne A
Stokes Donald
Manto Damon
Knepper Randolph L
Merski Richard
Taylor Jeff
Carnock Michael
Beams Gregory
Isadore Richard J
Merwald Gerda V
Reeder Bernard P
hilpot Arthur C
th Marian
Wessus Benyam
Tallia Randy
Quintano Joseph H
Freeman Charles
Johnson Winston E
Graham Lee
Babei Malak
Cooney Donald
Gernald Clarence R
Noel William
Shoon Fred
Wertman Douglas A
McKowen Emmett
Singletary Michelle
Wessinger Hugh Jennings
MacNamara Lawrence
Hartshorn Merrill F
Buscovich Melanie
Fortney Anthony
LeClair Gene
Snetman Paul
Verdery Joseph H
Ziern Wolfgang
Kilborn Edgar
nan Betina
schmeyer Dane F

FILE: MAPRAND PASCAL A THE GEORGE WASHINGTON UNIVERSITY COMPUTER CENTER

Aurtado Alfredo
Mertins Gerhard G
o Leslie C
Thomas Roger E
Fox Eugene
Kann Stephen
Lynn Donald H
Monaghan Dennis P
Patton Robin Dee
Lebowitz Allen
Hartley Leon F
Burton Steven L
Hayward Mary
Mattson Melanie
Pannell Dollie
Robinson Mark
Stultz Ronald
Keefe Kenneth L
Hall Patrick
Doherty Hugh M
Keck Homer C
Nash Alexander Melvin
Staples Victoria H
Wessels Barry W
Ross Charles Bryan
Noland Henry Peyton
Anderson Adrian B
erson Alan S
erson Alden P
Kropp Allen E
Kropp Allyson
Kropp Alvin D
Kropp Amie C
Krohl Anders
Krohl Andy
Slate Angela S
Slate Annamarie
Veith Arrietta E
veith Arthur J
Lance Barbara B
Lance Barton L
Lance Bernard
Lance Beverly
Lance Bobby D
Lance Bowman C
Roman Bradley
Roman Brent
Roman Brian
Romas Bruce
Roman Carl A
Roman Carlyn F
Roman Caroline G
Roman Charles
as Chris
Scott Clark

Scott Cynthia
Scott Delwin M
t Donald D
Scott Donald H
Scott Donald P
Stamm Douglas J
Stamm Duane C
McKee Dwayne S
McKee Edward L
McKee Elizabeth
McKee Ellen L
McKee Elliott T
Lucas Emma O
Lucas Emmett R
Lucas Eric
Lucas Edward L
Lucas F Burr
Lucas Floyd G
Lucas Frances H
Lucas Frank T
Lucas Franklin R
Lucas Fred A
Lucas Frederick
Inman G Sydwell
Inman George A
Inman Gerald O
Inman Gilbert
Inman Gina M
in Glen F
Inman Glenda
Inman Glenn
Inman Gordon E
Inman Grace
Inman Gregory P
Inman Harold
Irvin Harry G
Irvin Harry P
Fahey Helen P
Fahey Henry W
Fahey Herbert
Fahey Herman C
Fahey Jack
Fahey James
Coons Jane
Coons Joe
Coons John
Coons Joseph G
Coons Judith
Close Karen D
Close Karl
Close Katherine
Close Kathy
Close Kenneth B
o Kevin L
sso Kevin N

FILE: MAPRAND PASCAL A THE GEORGE WASHINGTON UNIVERSITY COMPUTER CENTER

Rosso Kim
Rosso Kristen
l Kurk
Stoll Lambert
Stoll Larry D
Stoll Lawrence M
Stoll Lee Jr
Stone Lee
Stone Leonard F
Stone LeRoy J
Stone Leslie K
Marsh Lewis E
Marsh Linda K
Marsh Lisa
Marsh Lloyd P Jr
Lutes Loren
Lutes Louis G
Kraus Lynn
Kapsa Marcia
Jaffe Margaret C
Jaffe Mark E
Jaffe Mary G
Rolfe Matthew T
Rolfe Michael
Rolfe Norris W
Rolfe Paul B
Rolfe Peder
Rolfe Peter K
Rolfe Philip B
Rolfe Phyllis A
Rolfe Ralph E
Rolfe Randall
Rolfe Raymond E
Rolfe Reba
Payne Rebecca
Payne Rexford
Payne Richard K
Payne Rob L
Payne Robert
Payne Robert Bruce
Payne Roger
Payne Roland W
Payne Ronda
Massa Rosemary
Massa Roy W
Massa Russell B
Massa Ruth E
Massa Sally
Massa Samuel
Letts Sandra
Letts Sandy
Hayes Sara F
Gross Scott D
Gross Seven A
Gross Shannon M

FILE: MAPRAND PASCAL A THE GEORGE WASHINGTON UNIVERSITY COMPUTER CENTER

Gross Shawn
Gross Sidney D
s Signe L
Gross Sonia M
Gross Stanley F
Gross Stephen J
Gross Steve
Ferry Steven
Ferry Tania
Ferry Teresa
Ferry Teri L
Ferry Terry
Ferry Thomas
Ferry Timothy
Razer Margaret C
Razer Tobyn J
Kapur Tom E
Kapur Truman
Kapur Vernice
Kapur Wade L
Kapur Wallace W
Kapur William R
Browning Ben H
Browning Benji Howard
Browning Bruce
Browning Charles E
Browning Chester H
Browning Denton R
ning Denver
Browning Doris J
Browning Earl S Jr
Browning Eugene
Browning George M
Browning James J
Browning Jan
Browning Jeanne
Browning Kelly
Kiernan Michael R
Kiernan Patricia
Kiernan Randy
Kiernan Robert E
Kiernan Tommy R
Little Warren
Little Wayne
Little William
Little William S
Cooper Adren
Cooper Aileen
Cooper Albert
Cooper Alfred D
Cooper Alla
Cooper Alvin
Cooper Arthur I
er Benjamin
oper Bradford

FILE: MAPRAND PASCAL A THE GEORGE WASHINGTON UNIVERSITY COMPUTER CENTER

Cooper Brooks
Cooper Carleton
Cooper Charles E
Cooper Christine
Murray Claude E
Murray Daniel
Murray Darrell
Murray David F
Murray Dennis C
Murray Donald F
Murray Eileen K
Murray Douglas G
Murray Gary R
Murray Geoffrey
Slater Glenn C
Slater Graham
Slater Gregory
Slater Harry
Slater James D
Slater Jane
Slater Jay
Slater Jeanne E
Tyler Jennifer
Tyler Jerry L
Tyler Jessie
Coburn Jimmie Steve
Coburn John
Coburn Joshua
Coburn Kevin
Coburn Kyna
Coburn Leland Q
Coburn Lonnie
Burris Louvenia
Burris Lynn
Burris Maldwyn J
Burris Margaret K
Burris Nancy
Burris Neal
Burris Nila
Burris Norman GG
Burris Pamela
Hewitt Patricia
Hewitt Paula
Hewitt Pete
Hewitt Ralph D
Hewitt Richard D
Hewitt Roger
Hewitt Samuel M
Hewitt Sue
Hewitt Ted O
Cooper Thomas E
Cooper Timothy
Cooper Tommye
Cooper William H
Cooper Williams

Dorsey Anne
Dorsey Barbara
sey Bonnie
Dorsey Bryan
Dorsey Daniel K
Dorsey Deborah
Dorsey Elizabeth M
Dorsey Elsie
Dorsey George L
Dorsey Harry Philip
Dorsey Jeff L
Dorsey John
Dorsey Katheleen M
Steele Kenneth R
Steele Kevin J
Steele Clinton
Steele R Corbin
Steele Robt E
Steele Rosetta
Steele Sandra
Steele Sharon
Steere Spencer
Steere Stanton L
Steere Thomas
Evans A Lee
Evans Alan
Evans Albert J
Evans David L
ns Dennis
Evans Donald
Evans Dorothy
Evans Doug
Evans Dwight
Evans Eddie
Evans Eliot D
Evans Elizabeth D
Evans Frank
Evans Franklin N
Evans Gordon
Evans Gaynelle
Veach Geoffrey
Veach George E
Veach Gerald E
veach Gidwill
Veach Glenn
Wine Glorious
Winer Gregory C
Winer Haydn K
Winer Herman
Kasse Jack P
Kasse James D
Kasse James K
Gibbs Jeanette L
bs Jeffrey
bs Joanna

FILE: MAPRAND PASCAL A THE GEORGE WASHINGTON UNIVERSITY COMPUTER CENTER

Gibbs John
Gibbs Judith A
[illegible]bs Julia
Gibbs Julius F
Gibbs Karen L
Gibbs Keith E
Gibbs Kenny
Gibbs Kirk
Cocke Lucy M
Cocke Marianne
Cocke Mitchell
Cocke Norene R
Cocke Peter Y
Cocke Phyllis
Beall Preston
Beall Richard L
Beall Russell
Beall Ruth
Beall Sarah
Beall Sidney L
Beall Stuart J
Beall Thomas C
Beall Timothy R
Beall Todd
Beall Vernon L
Beall Walter S
Beall Willie Jr
[illegible]ord Andrew
[illegible]d Anna Mae
Ford Barry
Ford Brendan
Ford Bruce
Ford Calvin R
Ford Caroll
Ford Clyde
Ford Connie
Ford Declan
Ford Dwight D
Ford Gene
Ford Henry P
Slay Herbert N
Slay Howard
Rose Jacob
Rose Jerome C
Rose Jess T
Rose Judy
Rose Kent
Rose Lester A
Knop Marguerite A
Knop Mary W
Knop Peyton H
Knox Robert Jr
Knox Terry
[illegible]x Tonya
[illegible]ox Wilfred C

FILE: MAPRAND PASCAL A THE GEORGE WASHINGTON UNIVERSITY COMPUTER CENTER

Gilmore Donald L
Gilmore Edward Sr
[illegible]more George E
Gilmore Hugh R
Gilmore Jessie S
Gilmore Margo
Gilmore Mike
Gilmore Ned D
Gilmore Orris F
Gilmore Ronald F
Gilmore Scott
Herbert Anita J
Herbert Arthur Carrier
Herbert Charrick L
Herbert Clayton W
Herbert Douglas C
Herbert Isabel
Herbert John
Herbert Jos A
Herbert Jule Jr
Herbert Lennell
Holland Linda
Holland Melvin
Holland Pauline
Holland Sandra
Holland Tammy
Holland Teresa
[illegible]lland Victory
[illegible]rner Walter
Fahrner Wayne
Fahrner William Maj
Irwin Chris
Irwin Dean L
Irwin Frederick Dale
Irwin Jim
Irwin Louis E
Irwin Tracey
Irwin Ricky
Irwin Stewart H
Johnson Byron
Johnson Carrie
Johnson Clifford P
Johnson Darrell
Johnson Dixie L
Johnson Edgar M
Johnson Emery
Johnson Greg
Johnson Howard
Collins Kimbrely A
Collins Lynette
Collins Maggie
Collins Marie W
Collins Marilyn S
[illegible]lins Marion
[illegible]llins Martin

Collins Maurice Carl
Collins Miles
lins Mitch
Brunner Moira
Brunner Monica
Brunner Monroe H
Brunner Patty
Brunner Raymond
Brunner Reuben
Brunner Sam
Bittner Stacey E
Bittner Ted
Bittner Theodore C
Gelinas Todd
Gelinas Velma
Gelinas Vernon B
Holden Vincent E
Holden Wallace
Holden Warner
Holden Wesley M
Keefe Braian
Keefe Debra
Keefe Edward M
Keehner J Martin
Keel B
Keeler Charles
Knight Franklin
ight Margot H
wles Jennifer
Knowles Katherine
Knudsen Sara M
Knudsen Stephen A
Lee Kunchull
Lee Marion A
Lee Marjorie
Lee Romaine
Lee Rory T Y
Leech Cynthia
Leeman Daniel
Lewis John E
Lewis Joseph D
Lewis Julian R
Lowell Pati
Lowenthal Emil
Lowery Bill
Macey E Christopher
Macey Wayne
MacGovern Robt N
MacGregor B J
MacGuire B J
Mach Christy
MacIntosh Donald
Mackeen S T
key Francis J
alker Edwin R

Walker Furman L
Walker Gerald T
ker Gordon
ıe.ry Alex
Terry Burdie
Terry Noel
Sweeney Thomas R
Swanson Dric V
Swann Charles
Stouse Eugene
Stoneman Tobert
Steinwinder L S
Smith Mary
Smith Alan
Smith Chip
Smith Caroline
Smith Chester H
Smith Christine
Smith Dana
Smith Darrell W
Smith Jefferson S
Smith Kevin R
Smith Mark E
Smith Merrill L
Smith Navarro
Smith Theodore H
Smith Wm Lewis
Seaman Barb E
man Eileen
Seaman James
Schneider Ivy
Schneider Keith H
Schlosser Alln D
Sargent Coutney R
Sargent Eliwood W
Sadler Ludy
Sadler Tierney
Sadler Mildred E
Rosner Fred J
Roseman James N
Roden Matthew
Robinson Perry
Robeson Palmer
Reed Herbert
Reed Jos M
Reed Philip
Reed Richard
Reed Steven W
Raymond Dorothy
Raymond Harvie
Raymond Leon
Raymond Vincent
Quinn Donald P
liam Alice L
ıce Corabel A

FILE: MAPRAND PASCAL A THE GEORGE WASHINGTON UNIVERSITY COMPUTER CENTER

Price Maxwell L
Price Morris E
ıce Helen T
ɹnz Arthur J
Power Lynne H
Powars Forrest
Potter Randal
Pollard Michael J
Pitt Pierre
Pittard C M
Parkinson Clarence F
Parkinson Leonard
Parkier Bradford J
Parkier Cecil E
Orleans David
Orleans Olgerts G
ONeill William
ONeill Kevin
Nichols Clayton I
Nichols Daryl E
Nelson Fred
Nelson Gerald
Nelson Harold D
Nelson Herb
Myer Jonathan
Nelson Edwin N
Nelson Frederick C
ɹelson Fanny
phy Margaret
Murphy Muriel
Murphy Netta
Morris Anthony
Morris Barry
Morris Debbie
Morris Donna
Morris Jeter
Morris Mollie
Morris Ronald W
Meekins Chris
Meekins Francis J
McLaughlin Jerry
McLaughlin Joseph D
McLaughlin Theresa F
Lewis Margaret C
Lewis Matthew Jr
Lewis Preston
Lewis Shelly
Lewis Webster T
Lewis Wellington
Layton Kenneth A
Layton Thomas P
Landis Diane
Landis Gregory
dis Janna
ause Araine

Krause Grant
Krause Jeffrey
ıse Walter P
ıramer Edward A
Kramer Ida Mae
King David M
King Donna
King Edmund J M
King Frederick M
King James
Kemper Anthony J
Kemper Beryl
Kemper Sandra
Kaye Arthur
Kaye Merelyn S
Kaye Zachary A
Kallen Arthur D
Kallio Alan J
Keane Daniel
Keane Mary J
Keane Sharon
Keane Thomas J
Jensen Oral J
Jensen Peter
Jensen Stanley
Jensen Thomas C
Jensen Windy
Abella Gabriel
amson Jonathan
Abramson Lynn
Abushady S
Ally Leon
Aloma Arturo
Alonso Adriana
Alonso Elena T
Armstrong Russ
Arnold Chas J
Arnold Peter L
Arnold Philip N
Bakley George
Balcom Roger C
Ball Claude
Ball Coletha
Ball Grank H
Ball Jean E
Ball Louis
Barton Johnny
Barton Rosalind C
Baxter Robt
Baxter Raymond S
Baxter Timothy M
Beach Eugene Y
Beach Norman P
ch Tommy
ch Alyce M

FILE: MAPRAND PASCAL A THE GEORGE WASHINGTON UNIVERSITY COMPUTER CENTER

Bitch Marian
Bird Joseph
[illegible] Richard
Birdsong Frank A
Briggs Albert
Briggs Duncan
Briggs Gary G
Briggs Marcus
Briggs Winston D
Burke Julia S
Burke Kathlenn
Burke Miriam V
Call Charles
Callaghan Brian
Callaghan Dorothy
Callaghan Robert
Carlson Forbe
Carlson Elaine M
Carlson Frances
Carlson Grady
Carlson Howard N
Carlson Lawrence R
Carlson Wendell
Carlstrom Brain
Chamberlain Alexander Scott
Chamberlain Patricia
Champion Ernest
[illegible]ndler Don M
[illegible]ndler Lance W
Coleman Albert
Coleman Dillon
coleman Elizabeth F
Crawford Andrews B Maj
Crawford Cathrine
Crawford Claudia C
Crawford Douglas P
Crawford Pauline
Crawford Steven
Crawford Thelma M
Davis Gordon S
Davis Jeremy
Davis Joan
Davis June M
Davis Lee
Davis Lloyd
Dunbar Annie
Dunbar Bradley
Dunbar Frederick
Duncan Clyde O
Duncan Duane
Duncan Jas J
Ecker David J
Ecker Jeffrey
[illegible]er Gary G
[illegible]erguson Blant Chaplin

FILE: MAPRAND PASCAL A THE GEORGE WASHINGTON UNIVERSITY COMPUTER CENTER

Ferguson Bruce A
Ferguson Cathy
juson Dale
erguson Jessica
Ferguson Katherine
Ferguson Mark G
Foreman Amy
Foreman Carter
Foreman George
Foreman Seth
Foreman Tim J
Gates Allan W
Gates Culver V
Gates Jesse
Gates Kirby
Gates Lawrence M
Gates Max
Glass Jennie L
Glass Roger P
Glass Glassco Ben
Goss Brian
Goss Hope
Goss Melissa
Gould Geo a
Gould Jewell
Gould Kenneth A
Hamilton Dale
milton Drew B
ilton Frederick
Hamilton Gregory
Hamilton Howard B
Hamilton Lindsay
Hamilton Milton H
Harrison Michael T
Harrison Raymond W
Harrison Monika E
Harrison Sharon
Harrison Warren J
Hess Jimmy D
Hess Orace H
Hess Sydney S
Hoffman Bernard
Hoffman Charles W
Hoffman Conrad R
Jennings Maggie
Jennings Nelson
Jennings Roseanna
Jennings Vivan M
Jennings Virginia G
Jensen Bernie
Jensen Chas A
Jensen Finn A
Jensen Hilary
sen Jamie
ensen Kirsten

FILE: MAPRAND PASCAL A THE GEORGE WASHINGTON UNIVERSITY COMPUTER CENTER

Jensen Richard D
Kanlan Abraham O
lan Bud
Kaplowitz Brian
Karami Muhamoud
Kemp Anne Marie
Kendall Eugene H
Kendrick Anna Magaret
Kendrick Don
Kendrick Mary Parker
Khanna Sudhir
Kidd Doreen R
Kid Gwen
Kidd Harry L Jr

```
333333333  777777777   9999999                EEEEEEEEE  N       NN  DDDDDDDD
333333333  777777777  999999999               EEEEEEEEE  NN      NN  DDDDDDDDD
      33   77    77   99     99               EE         NNN     NN  DD      DD
     33         77    99     99               EE         NNNN    NN  DD      DD
    3333        77    999999999   -------     EEEEEE     NN NN   NN  DD      DD
   333333       77     99999999   -------     EEEEEE     NN  NN  NN  DD      DD
      33        77           99               EE         NN   NNNN  DD      DD
33    33        77    99     99               EE         NN    NNN  DD      DD
333333333       77    999999999               EEEEEEEEE  NN      NN  DDDDDDDDD
 3333333        77     9999999                EEEEEEEEE  NN       N  DDDDDDDD
```

PRINT COMPLETED AT 21:38:21 FOR USER: RSCSREC DIST: SHIN

```
EEEEEEEEE  N       NN  DDDDDDDD                333333333  777777777   9999999
EEEEEEEEE  NN      NN  DDDDDDDDD               333333333  777777777  999999999
EE         NNN     NN  DD      DD              33    33   77    77   99     99
EE         NNNN    NN  DD      DD                   33        77   99     99
EEEEEE     NN NN   NN  DD      DD   -------        3333       77    999999999
EEEEEE     NN  NN  NN  DD      DD   -------      333333      77     99999999
EE         NN   NNNN  DD      DD                  33        77           99
EE         NN    NNN  DD      DD               33    33   77     99     99
EEEEEEEEE  NN      NN  DDDDDDDDD               333333333   77     999999999
EEEEEEEEE  NN       N  DDDDDDDD                 3333333     77      9999999
```

XEROX EPS XEROX EPS XEROX EPS XEROX EPS XEROX EPS XEROX EPS XEROX
XEROX EPS XEROX EPS XEROX EPS XEROX EPS XEROX EPS XEROX EPS XEROX
XEROX EPS XEROX EPS XEROX EPS XEROX EPS XEROX EPS XEROX EPS XEROX
XEROX EPS XEROX EPS XEROX EPS XEROX EPS XEROX EPS XEROX EPS XEROX

DATE: 29 SEP 91 AT 21:40:20

DEPARTMENT: DFAULT:JDL*

JOB ID: 2409 REPORT NO. 82

FILE ID:

INPUT PROCESSING TIME: 00:00:57

OUTPUT PROCESSING TIME: 00:00:53

REPORT COMPLETION CODE: 4

PAGES TO BIN: 71

PAGES TO TRAY: 0

PAPER PATH HOLES: 0

LINES PRINTED: 3806

ONLINE IDLE (SEC): 34

BLOCKS READ: 0

BLOCKS SKIPPED: 0

RECORDS READ: 3855

DJDE RECORDS READ: 1

MAXIMUM COPY COUNT: 1

OVERPRINTS: 0

COLLATE: YES

SF/MF: MULTI

SIMPLEX/DUPLEX: BOTH

JDE,JDL USED: DFLT,DFAULT

ACCTINFO:

INITIAL FONT LIST: LO112C

INITIAL FORM LIST: -NONE

INITIAL CME LIST: -NONE

XEROX EPS XEROX EPS XEROX EPS XEROX EPS XEROX EPS XEROX EPS XEROX
XEROX EPS XEROX EPS XEROX EPS XEROX EPS XEROX EPS XEROX EPS XEROX
XEROX EPS XEROX EPS XEROX EPS XEROX EPS XEROX EPS XEROX EPS XEROX
XEROX EPS XEROX EPS XEROX EPS XEROX EPS XEROX EPS XEROX EPS XEROX