A Collection of Research Processes for Genealogy and Proofs

VOLUME SIX, SECTION 34

Total Source Codes and Compilation, and Execution Listing for a Simulation Program of Shin's Join Algorithm and Shin's Hash Function

by

Dr. Dong-Keun Shin

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

```
time=1
//   EC PASCG
//S.SIN DD *
program SIMULATION (input, output);

const
     MAX_NUM_KEYS = 800;
     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;

     SIM_DEBUG  = false;
     S_DEBUGED  = false;

     MAX_STACK_SIZE = 5;
     NUL = 999;
     EMPTY = -1;
     DISCRIMINATOR = 'C';



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;

     STACK_PRIME_BOOL_TYPE = array (.1..MAX_STACK_SIZE.) of
                             record
                                  Bool_Key_Arr : BOOL_PRIME_KEY_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;

    CODER_TYPE = record
                    Hashed_Addr : HASH_ADDR_TYPE;
                 end;

    HASH_CODER_STACK = array (.1..MAX_STACK_SIZE.) of CODER_TYPE;

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

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

    BIT_ARR_TYPE = array (.0..BUCKET_SIZE.) of boolean;

    BIT_ARR_RECORD = record
                       Bit_Arr : BIT_ARR_TYPE;
                       Next_Bit : integer;
                       Source_Bucket,
                       Target_Bucket : BUCKET_POINTER_ARRAY;
                     end;

    BIT_ARR_TABLE = array (.1..MAX_STACK_SIZE.) of BIT_ARR_RECORD;

    ADDR_TYPE_ARRAY = array (.1..MAX_STACK_SIZE.) of integer;

    ASCII_RECORD = record
                     ASCII_Arr : ASCII_TABLE;
                   end;

    TYPE_ASCII_STACK = array (.1..MAX_STACK_SIZE.) of ASCII_RECORD;



var

    ASCII_Arr : ASCII_TABLE;

    Key_Char : char;

    Bool_Stack : STACK_PRIME_BOOL_TYPE;

    EXOR1_Arr : FIRST_EXOR_ARR_TYPE;
```

```
     EXOR2_Arr : SECOND_EXOR_ARR_TYPE;
     EXOR3_Arr : THIRD_EXOR_ARR_TYPE;

     Code_Stack : HASH_CODER_STACK;

     Stk_Pt : integer;
     Hd_Source_Pt, Hd_Target_Pt : LINK;
     stack : BIT_ARR_TABLE;
     finish : boolean;
     Addr_Num : integer;
     Hash_Value: integer;

     Source_Count, Target_Count, Result_Count, Hash_Count : integer;

     ASCII_Stack : TYPE_ASCII_STACK;




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, n : integer;
     number : integer;
     character : char;

begin

for n := 1 to MAX_STACK_SIZE do
  begin

     with ASCII_Stack(.n.) do
       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 := NUL;
                         ASCII_num := NUL;

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

            end;

     end;
  end;

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

          for i := 1 to NUM_IDENT_CHAR do
            begin
               for k := 1 to MAX_STACK_SIZE do
                 begin
                    with ASCII_Stack(.k.) do
                    ASCII_Arr(.i,j.).ch := character;
                 end;
            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
              for k := 1 to MAX_STACK_SIZE do
                begin
                   with ASCII_Stack(.k.) do
                   ASCII_Arr(.i,j.).ASCII_num := number;
                end;
           end;

         if j mod number_divisor = 0 then
              readln;

      end;


  for i := 1 to NUM_IDENT_CHAR do
    begin

     for j := 1 to NUM_ASCII_CHAR do
       begin

         with ASCII_Stack(.1.).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);

           end;  {with}
       end;  {for}

     readln;
     end;    {for}

     for n := 2 to MAX_STACK_SIZE do
       begin
          with ASCII_STACK(.n.) do
            begin
               for j := 1 to NUM_ASCII_CHAR do
                 begin
                    with ASCII_Arr(.1,j.) do
                      begin
                         read(prime_num);
                         if j mod number_divisor = 0 then
                              readln;
```

```
                    Int_To_Bool_Convert(prime_num, bool_prime);
                end;
            end;

          readln;

          for i := 1 to NUM_IDENT_CHAR do
            begin
               for j := 1 to NUM_ASCII_CHAR do
                 begin
                    ASCII_Arr(.i,j.).prime_num :=
                             ASCII_Arr(.1,j.).prime_num;

                    for k := 1 to MAX_BOOL_DIGIT do
                         ASCII_Arr(.i,j.).bool_prime(.k.) :=
                                  ASCII_Arr(.1,j.).bool_prime(.k.);
                end;
            end;
        end;  {with}
    end;     {for}


  Stk_Pt := 1;
  finish := false;
  Addr_Num := NUL;

  for i := 1 to MAX_STACK_SIZE do
    begin
       with stack(.i.) do
         begin
            for j := 0 to BUCKET_SIZE do
              begin
                 Bit_Arr(.j.) := false;
                 Source_Bucket(.j.) := nil;
                 Target_Bucket(.j.) := nil;
              end;
            Next_Bit := NUL;
         end;  {with}
    end;   {for}

  Hd_Source_Pt := nil;
  Hd_Target_Pt := nil;

  Hash_Count := 0;
  Source_Count := 0;
  Target_Count := 0;
  Result_Count := 0;

end;
```

```
procedure Form_Source_And_Target_Relations;

var
     Key_Pt : LINK;
     Char_No_First, Char_No : integer;
     Source_Target_Flag, Target_Flag, Last_Name_Flag : boolean;
     Key_No : integer;

     {For Debuggin Purpose}
     k1, k2 : integer;
     pt1, pt2 : LINK;




procedure Init_While_Do_Loop;

var  i: integer;


 egin

     Char_No := 1;

     new(Key_Pt);

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

     Source_Target_Flag := false;
     Target_Flag := false;
     Last_Name_Flag := true;
     Char_No_First := 1;
end;



function More_Chars_Left_For_Key : boolean;

begin

     if (((Char_No > NUM_IDENT_CHAR) or (Char_No_First > 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);

     if Last_Name_Flag then
       begin
          Key_Pt@.Key_Arr(.Char_No.) := Key_Char;
          Char_No := Char_No + 1;
       end
     else
       begin
          Key_Pt@.First_Name_Arr(.Char_No_First.) := Key_Char;
          Char_No_First := Char_No_First + 1;
       end;


   if (Key_Char = ' ') and (not Source_Target_Flag) then
       begin
          read(Key_Char);
          if ord(Key_Char) > ord(DISCRIMINATOR) then
            begin
               Target_Flag := true;
            end;
          Key_Pt@.First_Name_Arr(.Char_No_First.) := Key_Char;
          Char_No_First := Char_No_First + 1;
          Source_Target_Flag := true;
          Last_Name_Flag := false;
       end;
end;




pr  edure Attach_Key_To_Source_Or_Target_Relations;
```

```
begin
     if Target_Flag then
       begin
          Key_Pt@.next := Hd_Target_Pt;
          Hd_Target_Pt := Key_Pt;
          Target_Count := Target_Count + 1;
       end
     else
       begin
          Key_Pt@.next := Hd_Source_Pt;
          Hd_Source_Pt := Key_Pt;
          Source_Count := Source_Count + 1;
       end;
end;




{----------- Form_Source_And_Target_Relations --------------------}

begin
     for Key_No := 1 to MAX_NUM_KEYS do
       begin
          Init_While_Do_Loop;
          While More_Chars_Left_For_Key do
            begin
               Read_A_Char;
            end;
          Attach_Key_To_Source_Or_Target_Relation;
       end;

  writeln;
  write(' If the first character of a first name is between "A" ');
  writeln('and "', DISCRIMINATOR,'",');
  writeln(' the name will be included in the source relation.');
  write(' Otherwise, the name will be included');
  writeln(' in the target relation.');
  writeln;

if SIM_DEBUG then
  begin
  writeln;
  write (' ------------SOURCE------------------------------');
  writeln ('TARGET-----------------');
  writeln;
  pt1 := Hd_Source_Pt;
  pt2 := Hd_Target_Pt;
  k1 := 1;
  while ((pt1 <> nil) or (pt2 <> nil)) and (k1 <= MAX_NUM_KEYS) do
    begin
       write(' ');
       if pt1 <> nil then
         begin
            for k2 := 1 to NUM_IDENT_CHAR do
```

```
                write(ptl@.Key_Arr(.k2.));
            for k2 := 1 to NUM_IDENT_CHAR do
                write(ptl@.First_Name_Arr(.k2.));
            if ptl@.next <> nil then
                ptl := ptl@.next
            else
                ptl := nil;
          end
        else
          begin
            for k2 := 1 to NUM_IDENT_CHAR*2 do
                write(' ');
          end;

        write('          ');
        if pt2 <> nil then
          begin
            for k2 := 1 to NUM_IDENT_CHAR do
                write(pt2@.Key_Arr(.k2.));
            for k2 := 1 to NUM_IDENT_CHAR do
                write(pt2@.First_Name_Arr(.k2.));
            if pt2@.next <> nil then
                pt2 := pt2@.next
            else
                pt2 := nil;
          end;

        writeln;
        k1 := K1 + 1;

    end;
end;
end;
```

```
procedure Hash_Source_And_Target_Relations;

var
     point : LINK;
     address : integer;
     Key_Char : char;
     index : integer;
```

```
      i, j : integer;
      Source_Flag : boolean;
      Addr_Array : Addr_Type_Array;
      ok : boolean;
      ptl : LINK;



procedure Hash_Relation (pt : LINK; Source_Relation : boolean);

var
      Char_No, Bit_No : integer;
      Key_Pt : LINK;
      Next_Pt : LINK;
      i : integer;
      Coder_No : integer;



procedure Look_Up_Char_In_Prime_Num_Table(var idx:integer);

var   found : boolean;
      j : integer;

begin
      j := 1;
      found := false;
      repeat
       if ASCII_Stack(.Stk_Pt.).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, j : integer;

begin
```

```
for i := Stk_Pt to MAX_STACK_SIZE do
    begin
     with Bool_Stack(.i.).Bool_Key_Arr(.Char_No.) do
     begin

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

     end;  {with}
  end;

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(Code_No : integer);

var  i, j : integer;

begin

     i := 1;
```

```
     j := 1;

     repeat
        with Bool_Stack(.Code_No.) do
          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(Code_No : integer);

begin
  with Code_Stack(.Code_No.) do
     Hashed_Addr(.Bit_No.) := EX_OR(EXOR3_Arr(.1.), EXOR3_Arr(.2.));
end;
```

```
procedure Bool_To_Int_Convert (var addrs : Addr_Type_Array);

var  sum : integer;
     i, j : integer;
     offset : integer;

begin
     for i := 1 to MAX_STACK_SIZE do
       begin
          addrs(.i.) := NUL;
       end;

     for i := Stk'_Pt to MAX_STACK_SIZE do
       begin
          sum := 0;
          for j := MAX_BUCKET_ADDR_BITS+Stk_Pt downto 1+Stk_Pt do
            begin
               with Code_Stack(.i.) do
                 begin
                    if Hashed_Addr(.j.) then
                         sum := 2 * sum + 1
                    else
                         sum := 2 * sum;
                 end;
            end;
          addrs(.i.) := sum;
```

```
      end;
end;
```

```
{-------------------- Hash_Relation --------------------------------}

begin
     while pt <> nil do
       begin
          for Char_No := 1 to NUM_IDENT_CHAR do
            begin
               Key_Char := pt@.Key_Arr(.Char_No.);

            {Read character by character for a key looking up the
            corresponding prime number and convert it to binary number}
               Look_Up_Char_In_Prime_Num_Table(index);

               Save_Binary_Prime_Num(index);

            end;


          for Coder_No := Stk_Pt to MAX_STACK_SIZE do
            begin

          {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(Coder_No);
                   Second_Level_Ex_Oring;
                   Third_Level_Ex_Oring;
                   Last_Ex_Oring_And_Store_An_Addr_Bit(Coder_No);
                end;
            end;

          Bool_To_Int_Convert(Addr_Array);

          Hash_Count := Hash_Count + 1;
```

```
 f SIM_DEBUG then
  begin
     write(' ');
     for i:= 1 to NUM_IDENT_CHAR do
          write(pt@.Key_Arr(.i.));


     for i := 1 to MAX_STACK_SIZE do
          write(Addr_Array(.i.):7);
     writeln;
  end;

         Key_Pt := pt;
         Next_Pt := pt@.next;
         if Source_Relation then
           begin
              with stack(.Stk_Pt.) do
                 begin
                    Key_Pt@.next := Source_Bucket
                                          (.Addr_Array(.Stk_Pt.).);
                    Source_Bucket(.Addr_Array(.Stk_Pt.).)
                            := Key_Pt;
                 end;
                    for i := Stk_Pt to MAX_STACK_SIZE do
                         stack(.i.).Bit_Arr(.Addr_Array(.i.).) := true;
           end
          else
           begin
              ok := true;
              i := Stk_Pt;
              while ok and (i <= MAX_STACK_SIZE) do
                begin
                   with stack(.i.) do
                     begin
                      if not Bit_Arr(.Addr_Array(.i.).) then
                            ok := false;
                     end;
                   i := i + 1;
                end;

              if ok then
                begin
                   with stack(.Stk_Pt.) do
                     begin
                        Key_Pt@.next :=
                                 Target_Bucket(.Addr_Array(.Stk_Pt.).);
                        Target_Bucket(.Addr_Array(.Stk_Pt.).)
                                      := Key_Pt;
                     end;
                end;
           end;
         pt := Next_Pt;
    end;
er.
```

```
procedure Eliminate_Needless_Source_Relations;

var
     i : integer;

begin
     for i := 0 to BUCKET_SIZE do
       begin
          with stack(.Stk_Pt.) do
            begin
               if Target_Bucket(.i.) = nil then
                 begin
                    Source_Bucket(.i.) := nil;
                    Bit_Arr(.i.) := false;
                 end;
            end;
       end;
end;




{------- Hash_Source_And_Target_Relations start from here --------}

begin   {Hash_Source_And_Target_Relations}
     Source_Flag := true;
     point := Hd_Source_Pt;
     Hash_Relation (point, Source_Flag);


if SIM_DEBUG then
  begin
     for j := Stk_Pt to MAX_STACK_SIZE do
       begin
          writeln;
          write(' ------------');
          writeln(' BIT_ARRAY at Stack Level : ',j:1,' ------------');
          write(' ');
          for i := 0 to BUCKET_SIZE do
            begin

               if stack(.j.).Bit_Arr(.i.) then
                    write('1')
               else
                    write('0');
```

```
                if (i mod 50) = 49 then
                  begin;
                     writeln;
                  end;
                if (i mod 10) = 9 then
                     write(' ');
             end;
          writeln;
       end;
       writeln;

  end;


     Source_Flag := false;
     point := Hd_Target_Pt;
     Hash_Relation (point, Source_Flag);

     Eliminate_Needless_Source_Relations;



if SIM_DEBUG then
  begin
  writeln;
  writeln(' Hash_Source_And_Target called at Stack Level : ',Stk_Pt :1);
  with stack(.Stk_Pt.) do
       begin
          for i := 0 to BUCKET_SIZE do
            begin

               ptl := Source_Bucket(.i.);
               if ptl <> nil then
                 begin
                    writeln;
                    writeln(' --- Source Bucket No. ',i:1,' ---');
                 end;
               while ptl <> nil do
                 begin
                    write(' ');
                    for j := 1 to NUM_IDENT_CHAR do
                         write(ptl@.Key_Arr(.j.));

                    for j := 1 to NUM_IDENT_CHAR do
                         write(ptl@.First_Name_Arr(.j.));
                    writeln;
                    ptl := ptl@.next;
                 end;

               ptl := Target_Bucket(.i.);
               if ptl <> nil then
                 begin
                    writeln;
                    writeln(' --- Target Bucket No. ',i:3,' ---');
                 end;
```

```
              while ptl <> nil do
                begin
                   write(' ');
                   for j := 1 to NUM_IDENT_CHAR do
                        write(ptl@.Key_Arr(.j.));

                   for j := 1 to NUM_IDENT_CHAR do
                        write(ptl@.First_Name_Arr(.j.));
                   writeln;
                   ptl := ptl@.next;
                end;
           end;  {for}
      end;  {with}
  end;


end;









procedure Clear_Current_Upper_Part_Of_Stack;

var
     i, j : integer;
begin
     for i := Stk_Pt to MAX_STACK_SIZE do
       begin
          with stack(.i.) do
            begin
               for j := 0 to BUCKET_SIZE do
                 begin
                    Bit_Arr(.j.) := false;
                    Source_Bucket_Arr(.j.) := nil;
                    Target_Bucket_Arr(.j.) := nil;
                 end;
               Next_Bit := NUL;
            end;
       end;
end;









function Only_One_Bit_Set_After_Hash(var addr: integer) : boolean;

var
```

```
     flag : boolean;
     done : boolean;
     i, j : integer;

begin
     i := Stk_Pt;
     addr := NUL;
     done := false;
     while (not done) and (i <= MAX_STACK_SIZE) do
       begin
          flag := false;
          with stack(.i.) do
            begin
               j := 0;
               while (not done) and (j <= BUCKET_SIZE) do
                 begin
                    if Bit_Arr(.j.) and (not flag) then
                      begin
                         flag := true;
                         if i = Stk_Pt then
                           begin
                              addr := j;
                              Addr_Num := j;
                           end;
                      end
                    else
                         if Bit_Arr(.j.) and flag then
                           begin
                              done := true;
                              if i = Stk_Pt then
                                   Next_Bit := j;
                           end;
                    j := j + 1;
               end;
            end;

          i := i + 1;
       end;
     if done then
       begin
          Only_One_Bit_Set_After_Hash := false;
if SIM_DEBUG then
   begin
     writeln;
     write(' Only_One_Bit_Set? is false with an address : ', addr:1);
     writeln(' and the next address : ',stack(.Stk_Pt.).Next_Bit:1);
   end;
       end
     else
       begin
          Only_One_Bit_Set_After_Hash := true;
          if addr = NUL then
               addr := EMPTY;
i   IM_DEBUG then
   begin
```

```
          writeln;
          Pt_Source := Pt_Source@.next;
        end;
    end
  else
    begin
     if SIM_DEBUG then
          writeln(' There are no keys to be merged in this level.');
    end;
end;




procedure Save_Next_Addr_Bit;

var
     i : integer;
     found : boolean;
     num : integer;

begin
     num := NUL;
     found := false;
     i := Addr_Num + 1;
     while not found and (i <= BUCKET_SIZE) do
       begin
          if stack(.Stk_Pt.).Bit_Arr(.i.) then
            begin
               stack(.Stk_Pt.).Next_Bit := i;
               found := true;
            end;
          i := i + 1;
       end;

     if (not found) and (i > BUCKET_SIZE) then
          stack(.Stk_Pt.).Next_Bit := NUL;
end;




function No_More_Next_Addr: boolean;

begin
     if stack(.Stk_Pt.).Next_Bit = NUL then
       begin
          Addr_Num := NUL;
          No_More_Next_Addr := true;
```

```
      writeln;
      write(' Only_One_Bit_Set? is true with an address : ', addr:1);
      writeln(' and the next address : ',stack(.Stk_Pt.).Next_Bit:1);
    end;
      end;
end;




procedure Merge_Relations_And_Print_Out (addr: integer);

var
     Pt_Source, Pt_Target, Pt_T : LINK;
     i : integer;

begin
  if addr <> EMPTY then
   begin
     if SIM_DEBUG then
       begin
          write(' Merge Relations At Stack Level ',Stk_Pt : 1);
          writeln(' with  Bucket Number ',addr : 3);
       end;
     writeln;
     with stack(.Stk_Pt.) do
       begin
          Pt_Source := Source_Bucket(.addr.);
          Pt_Target := Target_Bucket(.addr.);
       end;
     while Pt_Source <> nil do
       begin
          Pt_T := Pt_Target;
          while Pt_T <> nil do
            begin
               write(' ');
               for i := 1 to NUM_IDENT_CHAR do
                    write(Pt_Source@.Key_Arr(.i.));

               for i := 1 to NUM_IDENT_CHAR do
                    write(Pt_Source@.First_Name_Arr(.i.));

               for i := 1 to NUM_IDENT_CHAR do
                    write(Pt_T@.Key_Arr(.i.));

               for i := 1 to NUM_IDENT_CHAR do
                    write(Pt_T@.First_Name_Arr(.i.));

               Pt_T := Pt_T@.next;
               Result_Count := Result_Count + 1;
               writeln;
            end;
          writeln;
```

```
      end
    else
      begin
         No_More_Next_Addr := false;
         Addr_Num := stack(.Stk_Pt.).Next_Bit;
      end;
end;
```

```
procedure Assign_Source_And_Target;

begin
  if SIM_DEBUG then
  begin
     writeln;
     write(' Assign_Source_And_T ==> Address Number : ',Addr_Num : 3);
     writeln('  Stack Number : ',Stk_Pt :1);
  end;
     Hd_Source_Pt := stack(.Stk_Pt.).Source_Bucket(.Addr_Num.);
     Hd_Target_Pt := stack(.Stk_Pt.).Target_Bucket(.Addr_Num.);
nd;
```

```
procedure pop;

begin
     Stk_Pt := Stk_Pt - 1;
     if Stk_Pt < 1 then
          writeln(' Stack Underflow !');

if SIM_DEBUG then
begin
writeln;
writeln(' Popped, Stack Pointer is ',Stk_Pt:1,' from ',Stk_Pt+1:1);
end;
end;
```

```
procedure push;

begin
     Stk_Pt := Stk_Pt + 1;
     if Stk_Pt > MAX_STACK_SIZE then
          writeln(' Stack Overflow !');
if SIM_DEBUG then
begin
writeln;
writeln(' Pushed, Stack Pointer is ',Stk_Pt:1,' from ',Stk_Pt-1:1);
end;
end;
```

```
function Bottom_Of_Stack : boolean;

begin
     if Stk_Pt = 1 then
          Bottom_Of_Stack := true
     else
          Bottom_Of_Stack := false;
end;
```

```
procedure Print_Statistics;

begin
```

```
     writeln;
     writeln;
     writeln;
     write(' THE TOTAL NUMBER OF KEYS IN THE SOURCE RELATION : ');
     writeln(Source_Count:4);
     writeln;
     write(' THE TOTAL NUMBER OF KEYS IN THE TARGET RELATION : ');
     writeln(Target_Count:4);
     writeln;
     write(' THE TOTAL NUMBER OF KEYS IN THE RESULT RELATION : ');
     writeln(Result_Count:4);
     writeln;
     writeln;
     writeln;
     write(' THE TOTAL NUMBER OF HASH CODER USED IN THE JOIN : ');
     writeln(Hash_Count:4);
     writeln;
end;




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

begin

Initialization;

Form_Source_And_Target_Relations;

repeat
   Clear_Current_Upper_Part_Of_Stack;

   Hash_Source_And_Target_Relations;

   if Only_One_Bit_Set_After_Hash(Hash_Value) then
   begin
      Merge_Relations_And_Print_Out(Hash_Value);
      if No_More_Next_Addr then
      begin
         if Bottom_Of_Stack then
            finish := true
         else
         begin
            pop;
            if No_More_Next_Addr then
            begin
               if Bottom_Of_Stack then
                  finish := true
               else
```

```
            begin
               pop;
               if No_More_Next_Addr then
               begin
                  if Bottom_Of_Stack then
                     finish := true
                  else
                  begin
                     pop;
                     if No_More_Next_Addr then
                     begin
                        if Bottom_Of_Stack then
                           finish := true
                        else
                        begin
                           pop;
                           if No_More_Next_Addr then
                           begin
                              if Bottom_Of_Stack then
                                 finish := true
                              else
                              begin
                                 Assign_Source_And_Target;
                                 Save_Next_Addr_Bit;
                                 push;
                              end;
                           end;
                        end;
                     end
                     else
                     begin
                        Assign_Source_And_Target;
                        Save_Next_Addr_Bit;
                        push;
                     end;
                  end;
               end
               else
               begin
                  Assign_Source_And_Target;
                  Save_Next_Addr_Bit;
                  push;
               end;
            end;
         end
         else
         begin
            Assign_Source_And_Target;
            Save_Next_Addr_Bit;
            push;
         end;
      end;
   end
   else
   begin
```

```
         Assign_Source_And_Target;
         Save_Next_Addr_Bit;
         push;
      end;
   end
   else
   begin
      Assign_Source_And_Target;
      Save_Next_Addr_Bit;
      push;
   end
until finish;

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
 15 116 117 118 119 120 121 122 53 54
55 56 57 35 46 44 39 45 95 36
27.. 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

2063 7927 5087 3583 1307 7687 1523 3643  223 7481
 523 7129 5669 3229 7789 8527 4969 2549 1721 3929
5189 5563 5981 4021 3187 3167 4409 6827 1109 4241
6323  769 4363 4801 1481 6367 7963 1747 2203 1061
3083 6547 3727 7069 2887 2221 8009 1987 2161 3301
4583 4127 7541 6361  967 5627 2309 4787 6581  641
 743 5347 3709 6763 1021 1949 5449 3041 3907 6689

7927 5087 3583 1307 7687 1523 3643  223 7481 5483
7129 5669 3229 7789 8527 4969 2549 1721 3929 1607
5563 5981 4021 3187 3167 4409 6827 1109 4241 1123
 769 4363 4801 1481 6367 7963 1747 2203 1061 3823
6547 3727 7069 2887 2221 8009 1987 2161 3301 8167
4127 7541 6361  967 5627 2309 4787 6581  641  443
5347 3709 6763 1021 1949 5449 3041 3907 6689 6067

5087 3583 1307 7687 1523 3643  223 7481 5483  401
3709   3229 7789 8527 4969 2549 1721 3929 1607 6121
5981 4021 3187 3167 4409 6827 1109 4241 1123 4129
```

```
4363 4801 1481 6367 7963 1747 2203 1061 3823 6949
372? 7069 2887 2221 8009 1987 2161 3301 8167 3449
75   6361  967 5627 2309 4787 6581  641  443 7349
3709 6763 1021 1949 5449 3041 3907 6689 6067 5521

3583 1307 7687 1523 3643  223 7481 5483  401 8447
3229 7789 8527 4969 2549 1721 3929 1607 6121  587
4021 3187 3167 4409 6827 1109 4241 1123 4129 2143
4801 1481 6367 7963 1747 2203 1061 3823 6949 6947
7069 2887 2221 8009 1987 2161 3301 8167 3449 3343
6361  967 5627 2309 4787 6581  641  443 7349 5683
6763 1021 1949 5449 3041 3907 6689 6067 5521 4003

1307 7687 1523 3643  223 7481 5483  401 8447 7949
7789 8527 4969 2549 1721 3929 1607 6121  587 1249
3187 3167 4409 6827 1109 4241 1123 4129 2143  821
1481 6367 7963 1747 2203 1061 3823 6949 6947 7589
2887 2221 8009 1987 2161 3301 8167 3449 3343 6269
 967 5627 2309 4787 6581  641  443 7349 5683 4421
1021 1949 5449 3041 3907 6689 6067 5521 4003 3769

7687 1523 3643  223 7481 5483  401 8447 7949  727
8527 4969 2549 1721 3929 1607 6121  587 1249 8236
3167 4409 6827 1109 4241 1123 4129 2143  821 7247
6367 7963 1747 2203 1061 3823 6949 6947 7589 2467
2221 8009 1987 2161 3301 8167 3449 3343 6269 1087
5627 2309 4787 6581  641  443 7349 5683 4421 3467
1?   5449 3041 3907 6689 6067 5521 4003 3769 5167

1523 3643  223 7481 5483  401 8447 7949  727 1601
4969 2549 1721 3929 1607 6121  587 1249 8236  541
4409 6827 1109 4241 1123 4129 2143  821 7247 2909
7963 1747 2203 1061 3823 6949 6947 7589 2467 4649
8009 1987 2161 3301 8167 3449 3343 6269 1087 5741
2309 4787 6581  641  443 7349 5683 4421 3467 3121
5449 3041 3907 6689 6067 5521 4003 3769 5167 6481

4969 2549 1721 3929 1607 6121  587 1249 8236  541
4409 6827 1109 4241 1123 4129 2143  821 7247 2909
7963 1747 2203 1061 3823 6949 6947 7589 2467 4649
8009 1987 2161 3301 8167 3449 3343 6269 1087 5741
2309 4787 6581  641  443 7349 5683 4421 3467 3121
5449 3041 3907 6689 6067 5521 4003 3769 5167 6481
1523 3643  223 7481 5483  401 8447 7949  727 1601

4409 6827 1109 4241 1123 4129 2143  821 7247 2909
7963 1747 2203 1061 3823 6949 6947 7589 2467 4649
8009 1987 2161 3301 8167 3449 3343 6269 1087 5741
2309 4787 6581  641  443 7349 5683 4421 3467 3121
5449 3041 3907 6689 6067 5521 4003 3769 5167 6481
1523 3643  223 7481 5483  401 8447 7949  727 1601
4969 2549 1721 3929 1607 6121  587 1249 8236  541

7?   1747 2203 1061 3823 6949 6947 7589 2467 4649
8009 1987 2161 3301 8167 3449 3343 6269 1087 5741
```

2309 4787 6581 641 443 7349 5683 4421 3467 3121
5449 3041 3907 6689 6067 5521 4003 3769 5167 6481
1523 3643 223 7481 5483 401 8447 7949 727 1601
4969 2549 1721 3929 1607 6121 587 1249 8236 541
4409 6827 1109 4241 1123 4129 2143 821 7247 2909

8009 1987 2161 3301 8167 3449 3343 6269 1087 5741
2309 4787 6581 641 443 7349 5683 4421 3467 3121
5449 3041 3907 6689 6067 5521 4003 3769 5167 6481
1523 3643 223 7481 5483 401 8447 7949 727 1601
4969 2549 1721 3929 1607 6121 587 1249 8236 541
4409 6827 1109 4241 1123 4129 2143 821 7247 2909
7963 1747 2203 1061 3823 6949 6947 7589 2467 4649

2309 4787 6581 641 443 7349 5683 4421 3467 3121
5449 3041 3907 6689 6067 5521 4003 3769 5167 6481
1523 3643 223 7481 5483 401 8447 7949 727 1601
4969 2549 1721 3929 1607 6121 587 1249 8236 541
4409 6827 1109 4241 1123 4129 2143 821 7247 2909
7963 1747 2203 1061 3823 6949 6947 7589 2467 4649
8009 1987 2161 3301 8167 3449 3343 6269 1087 5741

5449 3041 3907 6689 6067 5521 4003 3769 5167 6481
1523 3643 223 7481 5483 401 8447 7949 727 1601
4969 2549 1721 3929 1607 6121 587 1249 8236 541
4409 6827 1109 4241 1123 4129 2143 821 7247 2909
7963 1747 2203 1061 3823 6949 6947 7589 2467 4649
8009 1987 2161 3301 8167 3449 3343 6269 1087 5741
2309 4787 6581 641 443 7349 5683 4421 3467 3121

3041 3907 6689 6067 5521 4003 3769 5167 6481 5281
3643 223 7481 5483 401 8447 7949 727 1601 7741
2549 1721 3929 1607 6121 587 1249 8236 541 1201
6827 1109 4241 1123 4129 2143 821 7247 2909 2141
1747 2203 1061 3823 6949 6947 7589 2467 4649 5821
1987 2161 3301 8167 3449 3343 6269 1087 5741 6421
4787 6581 641 443 7349 5683 4421 3467 3121 2789

3907 6689 6067 5521 4003 3769 5167 6481 5281 1423
223 7481 5483 401 8447 7949 727 1601 7741 7603
1721 3929 1607 6121 587 1249 8236 541 1201 5843
1109 4241 1123 4129 2143 821 7247 2909 2141 4967
2203 1061 3823 6949 6947 7589 2467 4649 5821 1823
2161 3301 8167 3449 3343 6269 1087 5741 6421 6947
6581 641 443 7349 5683 4421 3467 3121 2789 2687

1283 1381 2767 3061 4507 4861 6067 6569 7907 8369
1447 1609 3023 3221 4663 5101 6323 6781 8243 8609
1607 1801 3307 3461 4967 5441 6703 7069 8447 8761
1787 2081 3527 3761 5167 5669 6907 7349 8647 9001
2027 2281 3767 4001 5443 5849 7127 7561 8863 9209
2243 2549 3943 4261 5647 6101 7487 7789 9067 9461
2467 2789 4243 4549 5867 6361 7643 8081 9323 9721

1303 1409 2803 3089 4523 4889 6143 6581 7927 8389

1483 1621 3067 3229 4703 5189 6343 6829 8263 8629
1617 1861 3323 3469 4987 5449 6763 7109 8467 8821
1[illegible] 2089 3547 3769 5227 5689 6947 7369 8663 9029
2063 2309 3803 4021 5483 5861 7187 7589 8867 9221
2267 2609 3947 4289 5683 6121 7507 7829 9103 9521
2503 2801 4283 4561 5903 6389 7687 8089 9343 9749

941 2207 2521 3923 4241 5623 6089 7307 7741 9043
1009 2243 2549 3943 4261 5647 6101 7487 7789 9067
1223 1361 2707 3049 4483 4801 6047 6529 7883 8329
1427 1601 2963 3209 4643 5081 6287 6761 8167 8581
1583 1789 3203 3449 4963 5381 6607 7001 8443 8741
1783 2069 3467 3709 5147 5641 6883 7321 8627 8969
2003 2269 3727 3989 5407 5821 7103 7549 8807 9181

9283 9689 2447 2749 4127 4481 5843 6329 7607 8069
9323 9721 2543 2861 4327 4621 5923 6421 7703 8101
9343 9749 2647 2909 4363 4649 5927 6449 7723 8161
9403 9769 2663 2969 4423 4721 5987 6469 7727 8209
9643 9781 2683 3001 4447 4729 6007 6481 7823 8221
9467 9829 2467 2789 4243 4549 5867 6361 7643 8081
9547 9901 2503 2801 4283 4561 5903 6389 7687 8089

Shin Danny
Shin Danny
Anderson J
Anderson J A
A[illegible]rson J C
A[illegible]erson 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
Verdin Peter
Fernstrom John R Jr.
Brooks Rose A
Diamonstein Kevin M
Hudgins Howard H
Labarski Alfred
Merriman Allen R
Ravera Robert
Tate Ada Cecelia
Keebler Gregory
[illegible]uller Harry
Campbell William P
E[illegible]orn Annie Lee
Ludwig August W

FILE: SIMULATE PASCAL A THE GEORGE WASHINGTON UNIVERSITY

uckly Michael J
Stumbo Richard M
Mo. . 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
Keelean Susan
Fernandez Ricardo
Holzman Charlotte
Meruvia Louisa
Peeples Deborah
Shoot Tony
Thornton Elizabeth
Yasuda Akiharu
McGrath Michel V
Irwin Harry E
Roberts Tracey L
tull Bernard
.i iamson David W
Mc. ide 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
Mangum Wade
Peel Christopher
Rugbart Dennis
Irvin Wayne M
Falcone Jolene Penick
Chrisman Amanda M
Beard Donaldson E
Loftus Susan
Mitchell Gilbert H
Sarahan Josephine
Tracy Harold
Wolff Raymond H
c rop Carl W
orders John M

FILE: SIMULATE PASCAL A THE GEORGE WASHINGTON UNIVERSITY

Dowd Robert
Gomez Raphael
Rogers 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
Busch 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
Mussen Cliff
William Harold
Rawson Bruce S
Saenz Adolph B
Miller Chris
Grasse Mark
Dorsett Charmaine
Gawkowski Amy
Greber Alan
Malmberg Allan F
Puff paul
Nyborg Ellen J
Jcer Claude R
Chatelain Fred D

FILE: SIMULATE PASCAL A THE GEORGE WASHINGTON UNIVERSITY

Allam Anthony
Masaller Edwardo
Na er 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
Yoegel Eric P
Reyna 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
graf William
Hall 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
Gaylord Brian
Hurwit Paul
Luebeck John B
Nolton Becky
Zumstein Albert A
Popchak Chris
Linders Michael
Hale Thomas M
Brown Milton
Genovese Carroll F
Rawson Marshall
Voivalis Nausika G
K Perng
Marie Louis

FILE: SIMULATE PASCAL A THE GEORGE WASHINGTON UNIVERSITY

Chrisinger Edw L
Ballou Catherine H
Ka 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
Philpot Arthur C
Smith Marian
Wessus Benyam
Tallia Randy
Quintano Joseph H
Freeman Charles
Johnson Winston E
Graham Lee
Babei Malak
Cooney Donald
Gernald Clarence R
Noel William
Sh n Fred
We man 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
Coonan Betina
Buschmeyer Dane F
Hurtado Alfredo
Mertins Gerhard G
Shomo Leslie C
Thomas Roger E
Fox Eugene
Kann Stephen
Lynn Donald H
Monaghan Dennis P
Patton Robin Dee
Lebowitz Allen
Ha ey Leon F
Burton Steven L

Hayward Mary
Mattson Melanie
P. ell 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
Anderson Alan S
Anderson 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
L ce Barbara B
L ce 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
Romas Chris
Scott Clark
Scott Cynthia
Scott Delwin M
Scott Donald D
Scott Donald H
Scott Donald P
Stamm Douglas J
Stamm Duane C
McKee Dwayne S
McKee Edward L
McKee Elizabeth
M e Ellen L
McKee Elliott T

Lucas Emma O
Lucas Emmett R
Lu 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
Inman 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
Rosso Kevin L
Rosso Kevin N
Rosso Kim
Rosso Kristen
Stoll Kurk
Stoll Lambert
Stoll Larry D
Stoll Lawrence M
Stoll Lee Jr
Stone Lee
Stone Leonard F
Stone LeRoy J
St Leslie K
Marsh Lewis E

FILE: SIMULATE PASCAL A THE GEORGE WASHINGTON UNIVERSITY

larsh Linda K
Mar-h Lisa
Ma. . 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
ayne Robert
. e Robert Bruce
Pa,.e 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
Gross Shawn
Gross Sidney D
Gross Signe L
Gross Sonia M
Gross Stanley F
Gross Stephen J
Gross Steve
Ferry Steven
Ferry Tania
Ferry Teresa
e / Teri L
.erry Terry

erry Thomas
Ferry Timothy
Ra: 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
Browning 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
Ki nan Randy
Ki .ian 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
Cooper Benjamin
Cooper Bradford
Cooper Brooks
Cooper Carleton
Cooper Charles E
Cooper Christine
Murray Claude E
Murray Daniel
Murray Darrell
Murray David F
lurray Dennis C
Murray Donald F
Mu y Eileen K
Murray Douglas G

Murray Gary R
Murray Geoffrey
Sl er 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
B ris Norman GG
B is 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
Dorsey Bonnie
Dorsey Bryan
Dorsey Daniel K
Dorsey Deboragh
Dorsey Elizabeth M
Dorsey Elsie
Dorsey George L
Dorsey Harry Philip
D ey Jeff L
Dorsey John

Jorsey Katheleen M
Steele Kenneth R
St e 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
Evans 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
Gibbs Jeffrey
Gibbs Joanna
Gibbs John
Gibbs Judith A
Gibbs Julia
Gibbs Julius F
Gibbs Karen L
Gibbs Keith E
Gibbs Kenny
Gibbs Kirk
Cocke Lucy M
Cocke Marianne
Cocke Mitchell
Cocke Norene R

FILE: SIMULATE PASCAL A THE GEORGE WASHINGTON UNIVERSITY

ocke Peter Y
Cocke Phyllis
Be. 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
Ford Andrew
Ford Anna Mae
Ford Barry
Ford Brendan
Ford Bruce
Ford Calvin R
Ford Caroll
Ford Clyde
Ford Connie
Ford Declan
Ford Dwight D
ord Gene
o d Henry P
Sl 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
Knox Tonya
Knox Wilfred C
Gilmore Donald L
Gilmore Edward Sr
Gilmore George E
Gilmore Hugh R
Gilmore Jessie S
Gilmore Margo
Gilmore Mike
Gilmore Ned D
Gilmore Orris F
Gilmore Ronald F
i re Scott
erbert Anita J

Herbert Arthur Carrier
Herbert Charrick L
He ert 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
Holland Victory
Fahrner 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
J. ison 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
Collins Marion
Collins Martin
Collins Maurice Carl
Collins Miles
Collins Mitch
Brunner Moira
Brunner Monica
Brunner Monroe H
Brunner Patty
Brunner Raymond
Brunner Reuben
Brunner Sam
B ner Stacey E
Bittner Ted

FILE: SIMULATE PASCAL A THE GEORGE WASHINGTON UNIVERSITY

Gittner Theodore C
Gelinas Todd
Ge ias 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
Knight Margot H
Knowles 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
L s 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
Mackey Francis J
Walker Edwin R
Walker Furman L
Walker Gerald T
Walker Gordon
Terry Alex
Terry Burdie
Terry Noel
Sweeney Thomas R
Swanson Dric V
Swann Charles
Stouse Eugene
S eman Tobert
Steinwinder L S

FILE: SIMULATE PASCAL A THE GEORGE WASHINGTON UNIVERSITY

Smith Mary
Smith Alan
Sm ı 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
Seaman 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
R n 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
Pulliam Alice L
Price Corabel A
Price Maxwell L
Price Morris E
Prince Helen T
prinz Arthur J
Power Lynne H
Powars Forrest
Potter Randal
Pollard Michael J
Pitt Pierre
Pittard C M
P inson Clarence F
Parkinson Leonard

arkier Bradford J
Parkier Cecil E
Or ns 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
Nelson Fanny
Murphy Margaret
Murphy Muriel
Murphy Netta
Morris Anthony
Morris Barry
Morris Debbie
Morris Donna
Morris Jeter
Morris Mollie
Morris Ronald W
eekins Chris
eekins Francis J
Mc ughlin 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
Landis Janna
Krause Araine
Krause Grant
Krause Jeffrey
Krause Walter P
Kramer Edward A
Kramer Ida Mae
King David M
King Donna
King Edmund J M
ing Frederick M
King James
e r Anthony J
emper Beryl

FILE: SIMULATE PASCAL A THE GEORGE WASHINGTON UNIVERSITY

Kemper Sandra
Kaye Arthur
Ka 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
Abramson 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
Ba'ley George
Bi om 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
Beach Tommy
Birch Alyce M
Bitch Marian
Bird Joseph
Bird Richard
Birdsong Frank A
Briggs Albert
Briggs Duncan
Briggs Gary G
Briggs Marcus
Briggs Winston D
Burke Julia S
Bu : Kathlenn
Burke Miriam V

FILE: SIMULATE PASCAL A THE GEORGE WASHINGTON UNIVERSITY

Call Charles
Ca' ghan Brian
Ca..aghan 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
Chandler Don M
Chandler 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
rawford Thelma M
D: s 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
Ecker Gary G
Ferguson Blant Chaplin
Ferguson Bruce A
Ferguson Cathy
Ferguson Dale
Ferguson Jessica
Ferguson Katherine
Ferguson Mark G
Foreman Amy
Foreman Carter
Foreman George
Fo man Seth
o. man Tim J
Gates Allan W

Gates Culver V
Ga 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
Hamilton Drew B
Hamilton 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
He 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
Jensen Jamie
Jensen Kirsten
Jensen Richard D
Kaplan Abraham O
Kapalan Bud
Kaplowitz Brian
Karami Muhamoud
Kemp Anne Marie
Kendall Eugene H
Kendrick Anna Magaret
Kendrick Don
Ke -ick Mary Parker
Kha..na Sudhir
Kidd Doreen R

FILE: SIMULATE PASCAL A THE GEORGE WASHINGTON UNIVERSITY

Kid Gwen
Kid Harry L Jr
Carver M-A
Hendershot Ralph P
Sejnoha Robert P
Jin Young
Montano M
Hall Robert A
Fain R L
Lyon Eric E
Murphy Francis J
Luskey Clarence E
Wentzell Daniel W
Noble Laura
Taylor Michael
McCullough Douglas
Payton Floy E
Aubouin Marie
Giglio Michael F
Hampe Carl
Mansaray Fatmatta K
Crosby Keith W
Castro Benjamin
Argerbright Michael L
Tarlton B
Thompson A
eckinger E P
William Timothy
DuVall Clifton
Cohen Marquerite Mrs
Albright Penrose Lucas R Adm
Shockley L
Kihm Christopher P
Evans Thomas E III
Pollisino L
Kishiyama Michael M
Mussmon Richard W
Pirsch P E
Burhop William J
Borg Matthew F
Bliss R G
Escher A
O'rea Eric
Arnold B J
Harding S
Waring Sara
Bonilla Juan
Webster Paul
Gonzalez S R
Allen David P
Ganey Thomas P Lt Col
Redmond Wendy
Ball P R
ou ang
Lukash James R

Lohr R B
Sp[illegible]el Barry M
Na..[illegible] Roger F
Dunn Richard
Ramsay Dennis M
Gellis Ian
Woodson Lewis S III
Buffalow Donald
Williams Lloyd
Falcon Vidal
Fetty DAvid
Brown Jay C
Carriker A Wendell
Sasaya Hidemitsu
Baker Vance B
Hyde Thomas J
Hatch D K
Darrow James
Baker George R Jr
Leger Cam
Black Susan H
Boyce Devon
Forrestel D
Brandt Catherine
Swiadek Stanley F
Fitzpatrick R M
Beattie B H
B [illegible]n Mark H
O'[illegible]oole Kevin
Thompson John H
Stanford K A
Ribichaud Robert F
Middents Dirk
Capizzi Jerry J
Kably Abdulla
Kulha Dwayne
Belfield Brenda
Pepper F Jay
Stevens Peggy A Mrs
Aschbrenner R
Pallotta Arthur J
Hashmi Jawaid .
Gute Mindy
Gary T Jack Jr
Hahn Thomas J
Eagan Cheryl
Jenkins L
Lukasavich William P
Dipaula Michael J
Murphy John F
Symons Chris
Coon H M
P[illegible]ell Harry S CMSgt
C[illegible]orini Edward E Col
Peterson Todd D

FILE: SIMULATE PASCAL A THE GEORGE WASHINGTON UNIVERSITY

/aitas V S
Ha 'son K
Lyc..s J A
Clinkscale Robert M Col
Pratt S E
Cook Phillip Bruce
Williams Raymond
Saludo Lourdes
Samelko Gene
Dryburgh John C
Provenzano A J
Gist P
Marini B
Welborn J A
Hershey Loren W
Sears Robert S
Roddy Daniel J
Weslar Wm H Dr
Schmitt Richard C
Korte Dennis
Leonard M C
Carroll Michael G
Ancell S
Thomas Frances B
Kim Yeun
Mutschelknaus Brad
enneberger Cindy
S els Charles J
Nesmith B
Steel Linda K
Kishok Gene D
Conroy Mary E
Maitra Amit K
Ivey Pat
Byers D
Gonzalez K H
Caulfield Suzanne M
Drohan M
Hink Harry D
Rowland Darrell W
Monroe Michael A
Jeffries H H
Morris James A
Emory Alan S
Johnson Allen O
Holt Charles
Hillock Glen
Sexton J G
Fuenzalida Hernan
Schaffer Matthew
Linden Gregory
Patsey Justin F
C' k R E Mrs
(r..ek Scott
Brooker Richard I

FILE: SIMULATE PASCAL A THE GEORGE WASHINGTON UNIVERSITY

Howley Peter J
Der Max Jr
Moh..mad Susan
Jones S L
Richardson David L
Von Richthoffen Zoe
McManus Karen
Castro Elena
Pitts Russell
Aaron David J
Barazotto P
Alhagh Noor M
Markwood Susan Banville
Healy Richard W
Stillwill Larry
Carter John M
Shumar Wilmer A
Faught C H
Csulak L
Solack Geo A
Elgin Larry
Campbell H J Capt
Hutchinson James F
Porter Carl W Capt
Sutherland Shaun
Nichols Waldo F Mrs
Crider Mary
Ma : Forest J
Wennig Don L
Scuderi Thomas G
Herbert Walter C Jr
LaFave J
Zayatz Thomas
Ryan Tim
Clark S A
Shumaker Carl
Ryan C
Arnold John H
Tessman Herman D
Burke R S
Gannon Robin
Escudero Oscar R
Dietz David A
Delashmitt James H
Stallings Timothy
Jackson Glen R
Bauckman S W
Crossman Thos D
Walker Edward P
Rawoens Achiel
Lee Charlotte A Mrs
Hassan Kamel
Gehle Ben
Mo. tello Arthur P
Zook M

FILE: SIMULATE PASCAL A THE GEORGE WASHINGTON UNIVERSITY

Gross Robin
Ad s Hebron E
Sh alter Glen
Childress Eddie T
Vorasane Nane
Ferguson James H
Engeman I R
Stearns Hugh B
Deardorff Robt K
Myers Irvin P
Chandler Paul R H
Epley James A
Larson L
Stonelake Joanne A
Wall Lillian
Riden T L
Holloway James
Williams Freeman W Col
Gosh Linda
Williams I
Rabung R M
DeFend Alan
Foster J
Seaman Robert W
Malvaso Patrick
Bucher Kenneth J
ldrich Marlene
H Kevin S
Denton Stephen J
Jeffries David
Watts Jonathan S
Castellano Richard
Fortman Hazel
Cutler Lois
Schwartz Barry
Osman Walie
Merritt James B 5th
O'Brien Geraldine
Libbey Charles F
Tempchin Stanley Dr
Donellan J E
Grouby Britt
Goins D D
Vaught Alexandria
Lantonio D J
Anderson Anders
Norgard K
Connell Frank M Jr
Adams John E
Pazar Charles E
Derrington Darrell
Fedrigault Yves
Ma n T A
lc ow K
Wilkins Shirley

FILE: SIMULATE PASCAL A THE GEORGE WASHINGTON UNIVERSITY

Burke Carl J
Do[illegible]erty James J Jr
Gu[illegible]erson Kenny A
Brewington Ann P
Smyser Willis M Col
Harsher Clarence L
Molloy James
Birdsall James E
Soffe Stephen
Lynnes Christopher
Judnich Francis A
McLellan A E
Roets Peter
McFaul E J
Budd R G
Miklas S G
Kluttz Robert
Martin James Richard
Burns John J
Gillice S J
Chisholm C
Sherwood William D
Dougherty John Lt Col
Crowley Thomas A
Claffey Jonathan
Upfold C L
King R Diane
U[illegible] - George
Casteele Roger L
Peine I
Bryan Sandra G
Kacsmarik James
Yore Robert C
Dietrich Ivan N
Newsome Joe
Zorn Robert
Spero J M
Smith R E
Stoeger Jon H
Allen Eric
Bryan J E
Burdt D
Ross William A
Curnyn Kevin O
Hotchkiss E
Szymanski Richard
Moore Robert A
Lagana L S
Mussel J F
Diegan Richard
Weiss Brad D
Cannon-Brow Sandy
Wa[illegible] Elizabeth C
Jo[illegible]son George S
Leahy William V C

FILE: SIMULATE PASCAL A THE GEORGE WASHINGTON UNIVERSITY

Lotz G B II
Ph''lips Jim
Ca gliotti Renzo
Goins M
Israelian Ruben
Matheny Arthur
Chamma Salwa
Nichols Robert R
Ringer Thomas R
Jackson Robert H
Limaye Suresh R
Alston Jerome
Pollack Peter
Hyde M
Molina Oscal
Cont C
Rosenberg David
Walter Paul
Zeman Joseph
Chaplin Marie L Mrs
Millerioux B
Monyak Robert
Kosarin Jonathan
Daly Joe
Davis K
Halpin Jas R Jr
oncannon Joseph P
S oski J M
Riemer Peter P
Frame P A
Hauver Mark W
Spruill Mark E
Perkins Osa C
Sheeley Priscilla W
Paladino R A Col
Dowdy Phil A Rev
Riddell James W
Marsden Heita
Trout Howard W Jr
Erianne Michael S
Stewart M M
Jackson Tom
Chae Won Je
Johnson Moddie
Drof John H
Mallat Jean-Yves
Pohorylo E E
Mabis Edward
Gonzalez J D
Good Fred J
Crotty Charles J
Mcintosh John L
Sm''h Samuel
I. J H
Hopkins V

FILE: SIMULATE PASCAL A THE GEORGE WASHINGTON UNIVERSITY

;mith David S Jr
Puc'et William N
Mu. B A
Jones Aubrey B III
Seaman Jos W
Costello Kevin W
Davis Thomas M
Morales N
Ellington Troy D
O'Laughlin Edward
Koizumi Don T
Linhart Richard W
Deakin Peter J
Taylor D P
Burns Dennie G
Wynne George G
Villanasco Cornelio
Saulsbury Michael P
Young T K
Bilodeau Douglas T Jr
Vollmerhausen J
McCarthy Elizabeth
Egge Robt
Burks Timothy E
Wood Hildred H
?iper L
/leming J Evans
Pe 'ns Roy F Jr
Br..field David
Talbert Wayne
Randall Mabel
Myongsun Yi
Longmire David W
Bayless Bob
Koah Gary
Artin Janet
Sikora Frank
Isaacs A L
Parker Ronald H Capt
Craig N
Price C
Justus G J
Priester J M
Peacock Charles
Dupuis Bernard E
Small Melvin D
Doying William A E
Hasnip Michael K
Norton Arthur J
Lewis Kenneth H
Burneko Paul J
;awarecki Stephen J
Klot? Lynn
Qu ga Flora
Lause Brian E

FILE: SIMULATE PASCAL A THE GEORGE WASHINGTON UNIVERSITY

Smeeton Thomas R
Kiss Joseph
T. Kemeier O
Schwalenstocker M J
Lee Robert
Nutty Combe J H
Read S J
Mortonson Theodore
Crawford Richard I Col
Friedrich Scott
Dadeh A
Theresa DiResta
Vemeyer D P
Trahant William J
Knight Harry V Jr
Pullis Kenneth W
Bell David
Even William
Barr George A
Leighton David T
Cole L K
Greenwell Ralph W
McNamara Walter A
Unruh Paula
Blake Jeffrey
Kahan Jerome
Birkhimer C
S[illegible] Timothy
Stonecypher Don E
Brandis M
Payne H E
Goldyn R
Stidham Amu
Babin F K
Tirado Miguel Angel Jr
Sweeney Katie
Nelius Carl
Forrester Beth
Hollander E P
Braden Damon
Barbera S
Buursink John
Cho Yun Ju
Stevens Bernice A
Carpel Richard J
Basil Tony
Guy D
Godbout Thomas
Dwyer Hartley G
Smith Melinda K
Karimi Mohammad
Sweet David G
Westman E
L[illegible]in S L
Cole J A

FILE: SIMULATE PASCAL A THE GEORGE WASHINGTON UNIVERSITY

Clark Jas Harry Jr
Boyd Robert C
Ga. no Anthony
Jennings J
Beatson R
Ramzi P
Rubenstein David
Kerns Merlin C Col
Ponikowski Chester
Davis H Z
Noel Richard A
Potter Andrew
Lenortarage Bernard
Boyce David
Woodring P A
Takesian Walter
Burghal Abir
DelToro A M
Aleman Steven
Van Herreweghe D J
Hicks Lynn
Nong Hung B
Emsat Nabila
Ayoub Joanne
Kramer Richard
Reyes Calixto
Kramer Nornan C
Pa' t D
Sc_ c Curtis
Bean Cheri
Quick Herbert
Magill John Pierson
Wilcox Angie
Hull Ronald
Dix A E
Miller Lois H
Stroup Robert E
Trusty Christene
Lawrence Andrew G
Bean A
Norton L C
Stephan Amy A
Greenstreet J E
Gregory E M
Pruitt E R
Sorrough Irene Lt Col
Receveur Robert A
Connolly Mary
Pfister Thomas K
Perry Wm R Lt Col
Whitely Velma
Lambert Lisa
Kolczynski Norbert
Kr r C J
Linden Henning Gen

FILE: SIMULATE PASCAL A THE GEORGE WASHINGTON UNIVERSITY

.inehan T S
Dir-een John F
Ma ,ry Duave
Davis R
Plunkett Patrick
Floyd George
Kerson Mark
Ali Ashraf
Richter S J
Godwin Anne
Bechtel Terry
Davis V M
Meadows Douglas K
Dwyer R J III
Hoff R K
Gates C V
Karel J T
Coleman Alessa
Singh Rattan
Bourie P A G
Samuels Donald L
Koumas A T
Underwood T L
Main Pat
Fields Charles W Mrs
Spetzman Lloyd A
ronfeld Jay R
Wi on F R
Th..as Tom
Chacho K M
Murphy Robert D
Stephens Jesse L
Smith Gary R
Larson N K
Page Max C
Wright J R
Manner Cos
Gorrell Felix B
Cullen Jas A
Hamarma Fawzy I
Roderique J O
Escovar Hector
Alavi Ahmads
Edwards K
Hills C D
Covill L
Davidson Gene H
Cook Harold
Omidyar Yvette
Drennan Bruce L
Brown Robert M
Thornton David L
Bac' Ron
h Chon Shik
French Donald T

FILE: SIMULATE PASCAL A THE GEORGE WASHINGTON UNIVERSITY

Jansen David A
Cunningham E D
M att P
Menz Elizabeth
Barnette C L
Briggs Winston D
Hancock Heidi
Brendel Wayne R
Ellis Howard
Weymes Gerald
Hamidi Mohammad Z
Buress S
Simmons Mark C
Dendor Paul
Waterman Ron
Dolbow Wilbur N
Rowles M E
McDermott Matthew H
Robbins Stanley D
Fehlner Christine
Hall Anne Schleiger
Akar Jacie
Park Sueng Ho
Gilbert Tracy
Brock Robert
Perez Fidel
Suarez John
P ers Michael
R bik Robert
Brill Joseph P
Place Alain
Yorkdale Alan H
Hashem Hesham
Varela Pablo
Smith Herbert E Capt
Andrews Geneva Mrs
Cox S R
Burnham Ralph L
Chankeris C
Strisik Roger
Pollack Max
Terango Marco
Russell Alberta M
Stewart William D
Rabbee Fazle
Paschal E P
Burke A
McCormack C A
Duong Tony
Denhartog D J
Shackelford E L
Spangler Brian
Lewis William P
J eson William H
Rice Richard H Jr

Richards Tony
Bowlsby Allan D
Mc ely, M
Berneking Michael D
Miceli Alex
Pham Hung M
Wojcik Tess
Coats John
Ziauddin Osman
Haynes H E
Kenerson J S
Ortmeier Michael G
Roberts Delvis Burton
Helseth Ted
Webster B D
Korsh David I
Wesling Richard M
Dorn Peter
Mrstik Bernard J
Rinker Donald W
Vaxmonsky Albert L
Pedersen Darryl N
Wilson C E
Speers Lisa N
Dunn Danny
Saikin Drew
'Connell Jos F
berg Fred M
R Frank E
Robinson John D
Oliff James A
Hart Deborah J
Koroljow Walter
Catsis B
Craddock Steven C
Simmons Alfred J
Shutt Carlton W
Peterson C S Mrs
Okey R
Corey B
German R
Jones Geor
Shin H D
Bigby Gilbert A
Schweickert Jerome
Neal Mitchell
McLaughlin L A
Hinderman Joseph O
Gustafson Marcia
Shepard Ralph
Cross E D
Foster Kelly
Freezer Claudia
stly Jack K
German Sherri

Chatfield Terry
Dermer Ellen R
Ha[illegible] C W
Verasin Narong
Mileham Harry P
Wang Chung-Hsing
Bass David
Power M E
Marcum Eldon K
Lucero A
Ellyson B
Nerio Lissette Alba
Chandler Matthew J
Gurney L
Robson D
Collins Kathleen
Robinson K M
Webb Wanda
Mazur Mojmir F
Wright Thomas H Maj
Austin Walter I Capt
Clark John
Williford D E
Barr Michael A
Stokes C L
Bell Jackson
Keener Keith
M[illegible]hione Allen
Higgins Ed
Wright Henry W
Shannon Deborah
Wirtz Robert F Jr
Aikens Debbie
Lappine John
Bailey Marshall
Zewan J
Guiton B
Kane Michael C
Hong Jang Sik
McCloud Mary
Gaskill Helen M
Tribble C Wayne Col
Jamerson Hoyt L
Vdovjak John
Ashley Lewis J
Cundiff Kevin N
Szabo R E
Ko In Chul
Throgmorton Ellen M
Theriot Donald
Rosenberger Harry D Mrs
Wiles Robert Scott
Manning Richard P
Si[illegible]ton P A
Hall Lester

FILE: SIMULATE PASCAL A THE GEORGE WASHINGTON UNIVERSITY

Gentry Wm B Jr
Thompson William E
Bo[illegible]au Alain
Coletto Ges G
Reiss Lenore
Beck Greg
Woolfenden S
Wilhoit R F
Perez Orlando
Jacob William
Brown James W Jr
Rogers John L Jr
Taplett Robert D Col
Nichols Steven
Fitzgerald John Macknight
Silva M
Elkin Ben
Moore V A
McGinnis Cathy
O'Neal A
Allbright William F Jr
Kim Myung-Seok
Guanclique Dimas
Bennett Carl M
Mills Herman
Merle Norman
[illegible]harity Harry Jr
An[illegible]son Rob L
Roberts Leon M
Nardella Michael
Shapiro A
Serra Miguel
Hills David
Kraft Eugene C
Humes Martha
Walsh C G
Reams Geraldine B
Flethcer Zola
Francke Amiel
Cordingley D M
Domdom Rolands
Huber Fred
Ananka Jos P
Mace Jack S
McCrary Nathaniel
Lev B W
Hupp Rod
Galligan Dennis
Reikowsky John
Lachat S
McCabe Kathleen
Wilson James C
Turniman Saleh
[illegible]n t Richard A
Shepherd Ronald L

FILE: SIMULATE PASCAL A THE GEORGE WASHINGTON UNIVERSITY

Garside G
Mahefky Paul
St ss Michael
Powell Anthony
Grabes D Lucas
Haluska Susanne
Frakes M
Brown R J
Gibson Paul S
Ford S L
Nelson C
Bennett C A
Roy Minati
Lundin John E
Bishop H L
Usack Lola
Katona G
McCahill Robert T
Wieboldt Richard
Dunn Keith
Hill Jack D
Boyer T H
Welsh Denise
Suchy Joyce M
Speake Jack T
Baker Paul F
Waller Forrest E
Sy onds Robert H
Gr. Robert
Mori Alferd H
Beach C S
Meadows Gary L
Montalvo A
Galbraith William R
Perkins M A
Melander D J
Powell Thomas Shane
Ryan Richard H
Lane Adrienne
Porter Patrick A
Thompson Blair A
Zambrana Paul
Hardy Michael P
Apodala Francisco
Ogg P
Clough Lawrence
Irvin Wayne M
Sprouse Ronald N
Schroeder Harold
Ray Everett E
Billings Raymond C Jr
Salerno Ronald
Galvin Nicholas
Hu j Robert Y
Watts Suzanne

FILE: SIMULATE PASCAL A THE GEORGE WASHINGTON UNIVERSITY

Munsey Robert G
Kanlan Maynard H
Cl nolm William H
Wallace Donald L
Kuhn Dave
Rogers Marion
Choi Heung Sik
Creasy Scott W
Patel S
Yost H L
Strnad Jas J
Boudreaux J C
Schever L Robert
Clark Ronald A
Adel Amanudin
Fields Brian K
Mullin J
Liston C R
Le Hung
Ness Richard A
Fray J L
Moore I S
McSpadden A C
Greenbaum Michael C
Bromley George G
Danner B G
Pieruccini Alfred Lt Col
Re ser R
Mc Louis H
Finley John H
MacKnight R L
Lairmore G
Pome A
Bywaters L
Blagrove Tony
Lake Kelly
Imler A B
Edwards Frederick W
Kraus D V
Rassin G
Bradley Wm T R
Iorio Stanley A
Earmratana P
Ballentine L R
Friedrich Carolyn
Oconner William E Jr
Barnes P A
Miron Mike
Strange Ben W
Donlan Charles J
Snyder Cecilia E
Krokus Robert R
Symonds T M
Ka ghi Maknay Arani
Smith Robin E

FILE: SIMULATE PASCAL A THE GEORGE WASHINGTON UNIVERSITY

Shields David E
Obrien S J
Oe :er Albert L
Krafft John J
Wonderly Bernice P
Schmidt Mark T
Geier C A
Kuprewicy M B
Mohajeri Manouchehr
Bronson C D
Staley D R
Shau P
Moreau Francois
O'Dell R L Jr
Scott Irene F Judge
Nadolski Michael A
Cawley Kim Paul
Shafshuf Ahmed
VanMeter D K
Jerris Wilfrid C
Mann Judy
Burns D J
Dolan K
Tinder W Michael
Leblane Hector J
Hoyle Fred W
Zachem K W
Wyckoop Roy L
T. y John R
Sabatier Michael
Quattlebaum Chas Albert
Thomson John C
Steele Lee Roy
Long Lester Laird
Cumberland Wm P
Borys B A
Ryon Marshall
Faria W C
Howie Steade S
Elmore John A Brig Gen
Friedlander Carl
Sedlak Richard M
Nelson Jane
Handerson N J
Gear James Jr
DerBogosian D
Huff Dorothy
Miercort J A
Wiggins Rudolph V
Smith Sylvester
Klein Kenneth
Johnson Claud H
Mundy R
E ı Bill F
Stepp Raymond D Jr

```
    1  0680 --  program SIMULATION (input, output);
    2  06A8 --
    3  06A8 --  const
    4  06A8 --       MAX_NUM_KEYS = 800;
    5  06A8 --       NUM_IDENT_CHAR = 16;
    6  06A8 --       MAX_BUCKET_ADDR_BITS = 8;
    7  06A8 --       MAX_BOOL_DIGIT = 13;
    8  06A8 --       NUM_ASCII_CHAR = 70;
    9  06A8 --
   10  06A8 --       NUM_FIRST_EXOR = 8;
   11  06A8 --       NUM_SECOND_EXOR = 4;
   12  06A8 --       NUM_THIRD_EXOR = 2;
   13  06A8 --
   14  06A8 --       BUCKET_SIZE = 255;
   15  06A8 --
   16  06A8 --       SIM_DEBUG  = false;
   17  06A8 --       S_DEBUGED  = false;
   18  06A8 --
   19  06A8 --       MAX_STACK_SIZE = 5;
   20  06A8 --       NUL = 999;
   21  06A8 --       EMPTY = -1;
   22  06A8 --       DISCRIMINATOR = 'C';
   23  06A8 --
   24  06A8 --
   25  06A8 --
   26  06A8 --  type
   27  06A8 --       KEY_ARRAY_TYPE = array (.1..NUM_IDENT_CHAR.) of char;
   28  06A8 --
   29  06A8 --       PRIME_BOOL_TYPE = array (.1..MAX_BOOL_DIGIT.) of boolean;
   30  06A8 --
   31  06A8 --       CHAR_RECORD_TYPE = record
   32  06A8 --                            ch : char;
   33  06A8 --                            ASCII_num : integer;
   34  06A8 --                            prime_num : integer;
   35  06A8 --                            bool_prime : PRIME_BOOL_TYPE;
   36  06A8 --                          end;
   37  06A8 --
   38  06A8 --       ASCII_TABLE = array (.1..NUM_IDENT_CHAR, 1..NUM_ASCII_CHAR.) of
   39  06A8 --                     CHAR_RECORD_TYPE;
   40  06A8 --
   41  06A8 --       BOOL_PRIME_KEY_TYPE = array (.1..NUM_IDENT_CHAR.) of
   42  06A8 --                             record
   43  06A8 --                                  bool_prime : PRIME_BOOL_TYPE;
   44  06A8 --                             end;
   45  06A8 --
   46  06A8 --       STACK_PRIME_BOOL_TYPE = array (.1..MAX_STACK_SIZE.) of
   47  06A8 --                               record
   48  06A8 --                                    Bool_Key_Arr : BOOL_PRIME_KEY_TYPE;
   49  06A8 --                               end;
   50  06A8 --
   51  06A8 --       FIRST_EXOR_ARR_TYPE = array (.1..NUM_FIRST_EXOR.) of boolean;
   52  06A8 --
   53  06A8 --       SECOND_EXOR_ARR_TYPE = array (.1..NUM_SECOND_EXOR.) of boolean;
   54  06A8 --
   55  06A8 --       THIRD_EXOR_ARR_TYPE = array (.1..NUM_THIRD_EXOR.) of boolean;
   56  06A8 --
   57  06A8 --       HASHED_KEY_REG_TYPE = array (.1..MAX_BUCKET_ADDR_BITS.) of boolean;
   58  06A8 --
```

```
   59  06A8 --          HASH_ADDR_TYPE = array (.1..MAX_BOOL_DIGIT.) of boolean;
   60  06A8 --
       06A8 --          CODER_TYPE = record
   62  06A8 --                          Hashed_Addr : HASH_ADDR_TYPE;
   63  06A8 --                       end;
   64  06A8 --
   65  06A8 --          HASH_CODER_STACK = array (.1..MAX_STACK_SIZE.) of CODER_TYPE;
   66  06A8 --
   67  06A8 --          LINK = @KEY_RECORD;
   68  06A8 --          KEY_RECORD = record
   69  06A8 --                          Key_Arr : KEY_ARRAY_TYPE;
   70  06A8 --                          First_Name_Arr : KEY_ARRAY_TYPE;
   71  06A8 --                          next : LINK;
   72  06A8 --                       end;
   73  06A8 --
   74  06A8 --          BUCKET_POINTER_ARRAY = array (.0..BUCKET_SIZE.) of LINK;
   75  06A8 --
   76  06A8 --          BIT_ARR_TYPE = array (.0..BUCKET_SIZE.) of boolean;
   77  06A8 --
   78  06A8 --          BIT_ARR_RECORD = record
   79  06A8 --                             Bit_Arr : BIT_ARR_TYPE;
   80  06A8 --                             Next_Bit : integer;
   81  06A8 --                             Source_Bucket,
   82  06A8 --                             Target_Bucket : BUCKET_POINTER_ARRAY;
   83  06A8 --                           end;
   84  06A8 --
   85  06A8 --          BIT_ARR_TABLE = array (.1..MAX_STACK_SIZE.) of BIT_ARR_RECORD;
   86  06A8 --
    7  06A8 --          ADDR_TYPE_ARRAY = array (.1..MAX_STACK_SIZE.) of integer;
   88  06A8 --
   89  06A8 --          ASCII_RECORD = record
   90  06A8 --                           ASCII_Arr : ASCII_TABLE;
   91  06A8 --                         end;
   92  06A8 --
   93  06A8 --          TYPE_ASCII_STACK = array (.1..MAX_STACK_SIZE.) of ASCII_RECORD;
   94  06A8 --
   95  06A8 --
   96  06A8 --
   97  06A8 --
   98  06A8 --
   99  06A8 --    var
  100  06A8 --
  101  06A8 --          ASCII_Arr : ASCII_TABLE;
  102  11EA8 --
  103  11EA8 --          Key_Char : char;
  104  11EAC --
  105  11EAC --          Bool_Stack : STACK_PRIME_BOOL_TYPE;
  106  12EEC --
  107  12EEC --          EXOR1_Arr : FIRST_EXOR_ARR_TYPE;
  108  12F0C --          EXOR2_Arr : SECOND_EXOR_ARR_TYPE;
  109  12F1C --          EXOR3_Arr : THIRD_EXOR_ARR_TYPE;
  110  12F24 --
  111  12F24 --          Code_Stack : HASH_CODER_STACK;
    ?  13028 --
  113  13028 --          Stk_Pt : integer;
  114  1302C --          Hd_Source_Pt, Hd_Target_Pt : LINK;
  115  13034 --          stack : BIT_ARR_TABLE;
  116  16C48 --          finish : boolean;
```

```
117  16C4C --          Addr_Num : integer;
118  16C50 --          Hash_Value: integer;
119  16C54 --
     16C54 --          Source_Count, Target_Count, Result_Count, Hash_Count : integer;
121  16C64 --
122  16C64 --          ASCII_Stack : TYPE_ASCII_STACK;
123  6E464 --
124  6E464 --
125  6E464 --
126  6E464 --
127  6E464 --
128  6E464 -- A  procedure Int_To_Bool_Convert (number:integer; var Bool_Arr:
129   0040 --                                                    PRIME_BOOL_TYPE);
130   0048 --
131   0048 --     var  i : integer;
132   004C --
133   0000 0- A  begin
134   0012 --          for i := 1 to MAX_BOOL_DIGIT do
135   0030 --               Bool_Arr(.i.) := false;
136   0072 --
137   0072 --          i := 1;
138   007A --          while (number >= 2) and (i <= MAX_BOOL_DIGIT) do
139   00B2 1-            begin
140   00B2 --
141   00B2 --               if (number mod 2) =1 then
142   00CC --                    Bool_Arr(.i.) := true
143   00DE --               else
144   00F2 --                    Bool_Arr(.i.) := false;
 5    0112 --
.6    0112 --               number := number div 2;
147   0128 --               i := i + 1;
148   013A -1            end;
149   013E --
150   013E --          if (number = 1) and (i <= MAX_BOOL_DIGIT) then
151   0176 --               Bool_Arr(.i.) := true;
152   0198 --
153   0198 -0 A  end;
154   01B0 --
155   01B0 --
156   01B0 --
157   01B0 --
158   01B0 --
159   01B0 -- A  procedure Initialization;
160   0040 --
161   0040 --     const
162   0040 --          char_divisor = 20;
163   0040 --          number_divisor = 10;
164   0040 --
165   0040 --     var  i, j, k, n : integer;
166   0050 --          number : integer;
167   0054 --          character : char;
168   0058 --
169   0000 0- A  begin
170   0012 --
      0012 --     for n := 1 to MAX_STACK_SIZE do
172   0030 1-       begin
173   0030 --
174   0030 --          with ASCII_Stack(.n.) do
```

```
  175   0050 2-          begin
  176   0050 --
  177   0050 --             for i := 1 to NUM_IDENT_CHAR do
    3   006E 3-               begin
  179   006E --
  180   006E --                 for j := 1 to NUM_ASCII_CHAR do
  181   008C 4-                   begin
  182   008C --
  183   008C --                     with ASCII_Arr(.i,j.) do
  184   00C0 5-                       begin
  185   00C0 --                         ch := '?';
  186   00C8 --                         prime_num := NUL;
  187   00D0 --                         ASCII_num := NUL;
  188   00D8 --
  189   00D8 --                         for k := 1 to MAX_BOOL_DIGIT do
  190   00F6 --                              bool_prime(.k.) := false;
  191   0130 -5                       end;
  192   0130 -4                   end;
  193   0152 --
  194   0152 -3               end;
  195   0174 --
  196   0174 -2        end;
  197   0174 -1     end;
  198   0196 --
  199   0196 --        for j := 1 to NUM_ASCII_CHAR do
  200   01B4 1-          begin
  201   01B4 --             read(character);
  202   01F0 --
  203   01F0 --             for i := 1 to NUM_IDENT_CHAR do
  204   020E 2-               begin
  205   020E --                 for k := 1 to MAX_STACK_SIZE do
  206   022C 3-                   begin
  207   022C --                     with ASCII_Stack(.k.) do
  208   024C --                     ASCII_Arr(.i,j.).ch := character;
  209   028C -3                   end;
  210   02AE -2               end;
  211   02D0 --
  212   02D0 --             if j mod char_divisor = 0 then
  213   02E4 --                 readln;
  214   02F0 -1        end;
  215   0312 --
  216   0312 --        readln;
  217   031E --
  218   031E --
  219   031E --
  220   031E --        for j :=1 to NUM_ASCII_CHAR do
  221   033C 1-          begin
  222   033C --             read(number);
  223   034E --             if number > 64 then number := number - 64;
  224   0372 --
  225   0372 --             for i := 1 to NUM_IDENT_CHAR do
  226   0390 2-               begin
  227   0390 --                 for k := 1 to MAX_STACK_SIZE do
  228   03AE 3-                   begin
    9   03AE --                     with ASCII_Stack(.k.) do
  230   03CE --                     ASCII_Arr(.i,j.).ASCII_num := number;
  231   040E -3                   end;
  232   0430 -2               end;
```

```
233  0452 --
234  0452 --              if j mod number_divisor = 0 then
235  0466 --                   readln;
     0472 --
237  0472 -1           end;
238  0494 --
239  0494 --
240  0494 --       for i := 1 to NUM_IDENT_CHAR do
241  04B2 1-         begin
242  04B2 --
243  04B2 --           for j := 1 to NUM_ASCII_CHAR do
244  04D0 2-             begin
245  04D0 --
246  04D0 --               with ASCII_Stack(.1.).ASCII_Arr(.i, j.) do
247  0504 3-                 begin
248  0504 --
249  0504 --                     read(prime_num);
250  0516 --
251  0516 --                     if j mod number_divisor = 0 then
252  052A --                          readln;
253  0536 --
254  0536 --                     Int_To_Bool_Convert(prime_num, bool_prime);
255  0550 --
256  0550 -3                 end;  {with}
257  0550 -2             end;  {for}
258  0572 --
259  0572 --           readln;
260  057E -1           end;    {for}
261  05A0 --
262  05A0 --           for n := 2 to MAX_STACK_SIZE do
263  05BE 1-             begin
264  05BE --               with ASCII_STACK(.n.) do
265  05DE 2-                 begin
266  05DE --                   for j := 1 to NUM_ASCII_CHAR do
267  05FC 3-                     begin
268  05FC --                       with ASCII_Arr(.1,j.) do
269  0616 4-                         begin
270  0616 --                           read(prime_num);
271  0628 --                           if j mod number_divisor = 0 then
272  063C --                                readln;
273  0648 --                           Int_To_Bool_Convert(prime_num, bool_prime);
274  0662 -4                         end;
275  0662 -3                     end;
276  0684 --
277  0684 --                   readln;
278  0690 --
279  0690 --                   for i := 1 to NUM_IDENT_CHAR do
280  06AE 3-                     begin
281  06AE --                       for j := 1 to NUM_ASCII_CHAR do
282  06CC 4-                         begin
283  06CC --                           ASCII_Arr(.i,j.).prime_num :=
284  06F8 --                                    ASCII_Arr(.1,j.).prime_num;
285  0724 --
286  0724 --                           for k := 1 to MAX_BOOL_DIGIT do
287  0742 --                                ASCII_Arr(.i,j.).bool_prime(.k.) :=
288  0782 --                                         ASCII_Arr(.1,j.).bool_prime(.k.);
289  07E4 -4                         end;
290  0806 -3                     end;
```

```
291  0828 -2              end;  {with}
292  0828 -1          end;     {for}
293  084A --
     084A --
295  084A --        Stk_Pt := 1;
296  0858 --        finish := false;
297  0864 --        Addr_Num := NUL;
298  0872 --
299  0872 --        for i := 1 to MAX_STACK_SIZE do
300  0890 1-          begin
301  0890 --             with stack(.i.) do
302  08B0 2-               begin
303  08B0 --                  for j := 0 to BUCKET_SIZE do
304  08CC 3-                    begin
305  08CC --                       Bit_Arr(.j.) := false;
306  08E4 --                       Source_Bucket(.j.) := nil;
307  08FC --                       Target_Bucket(.j.) := nil;
308  0914 -3                    end;
309  0936 --                 Next_Bit := NUL;
310  093E -2              end;  {with}
311  093E -1          end;   {for}
312  0960 --
313  0960 --        Hd_Source_Pt := nil;
314  096C --        Hd_Target_Pt := nil;
315  0978 --
316  0978 --        Hash_Count := 0;
317  0984 --        Source_Count := 0;
318  0990 --        Target_Count := 0;
319  099C --        Result_Count := 0;
320  09A8 --
321  09A8 -0 A  end;
322  0A04 --
323  0A04 --
324  0A04 --
325  0A04 --
326  0A04 --
327  0A04 --
328  0A04 --
329  0A04 --
330  0A04 --
331  0A04 -- A  procedure Form_Source_And_Target_Relations;
332  0040 --
333  0040 --    var
334  0040 --         Key_Pt : LINK;
335  0044 --         Char_No_First, Char_No : integer;
336  004C --         Source_Target_Flag, Target_Flag, Last_Name_Flag : boolean;
337  0058 --         Key_No : integer;
338  005C --
339  005C --         {For Debuggin Purpose}
340  005C --         k1, k2 : integer;
341  0064 --         pt1, pt2 : LINK;
342  006C --
343  006C --
344  006C --
     006C --
346  006C --
347  006C --
348  006C --
```

```
  349   006C -- B  procedure Init_While_Do_Loop;
  350   0040 --
  351   0040 --     var  i: integer;
   52   0044 --
  353   0044 --
  354   0000 0- B  begin
  355   0012 --
  356   0012 --          Char_No := 1;
  357   001A --
  358   001A --          new(Key_Pt);
  359   0028 --
  360   0028 --          for i := 1 to NUM_IDENT_CHAR do
  361   0046 1-            begin
  362   0046 --               Key_Pt@.Key_Arr(.i.) := ' ';
  363   0070 --               Key_Pt@.First_Name_Arr(.i.) := ' ';
  364   0096 -1            end;
  365   00B8 --
  366   00B8 --          Source_Target_Flag := false;
  367   00BE --          Target_Flag := false;
  368   00C4 --          Last_Name_Flag := true;
  369   00CC --          Char_No_First := 1;
  370   00D4 -0 B  end;
  371   00E8 --
  372   00E8 --
  373   00E8 --
  374   00E8 --
  375   00E8 -- B  function More_Chars_Left_For_Key : boolean;
  376   0048 --
  377   0000 0- B  begin
    8   0018 --
  379   0018 --          if (((Char_No > NUM_IDENT_CHAR)or(Char_No_First > NUM_IDENT_CHAR))
  380   004A --             or eoln) or eof then
  381   0094 1-            begin
  382   0094 --               More_Chars_Left_For_Key := false;
  383   009A --               readln;
  384   00A6 -1            end
  385   00A6 --          else
  386   00AA --               More_Chars_Left_For_Key := true;
  387   00B2 -0 B  end;
  388   00D4 --
  389   00D4 --
  390   00D4 --
  391   00D4 --
  392   00D4 --
  393   00D4 --
  394   00D4 --
  395   00D4 --
  396   00D4 -- B  procedure Read_A_Char;
  397   0000 0- B  begin
  398   0012 --          read(Key_Char);
  399   0054 --
  400   0054 --          if Last_Name_Flag then
  401   0064 1-            begin
  402   0064 --               Key_Pt@.Key_Arr(.Char_No.) := Key_Char;
  4'    009A --               Char_No := Char_No + 1;
  404   00AC -1            end
  405   00AC --          else
  406   00B0 1-            begin
```

```
  581   00A6 -2                end
  582   00A6 --              else
    3   00AA 2-                begin
  584   00AA --                     j := j + 1;
  585   00BC -2                end;
  586   00BC -1            until found or (j > NUM_ASCII_CHAR);
  587   00E6 --
  588   00E6 --             if (j > NUM_ASCII_CHAR) and (not found) then
  589   0114 --                  idx := 64;
  590   0120 --
  591   0120 -0 C  end;
  592   0140 --
  593   0140 --
  594   0140 --
  595   0140 --
  596   0140 --
  597   0140 --
  598   0140 -- C  procedure Save_Binary_Prime_Num(idx: integer);
  599   0044 --
  600   0044 --    var  i, j : integer;
  601   004C --
  602   0000 0- C  begin
  603   0012 --
  604   0012 --    for i := Stk_Pt to MAX_STACK_SIZE do
  605   003C 1-      begin
  606   003C --        with Bool_Stack(.i.).Bool_Key_Arr(.Char_No.) do
  607   0076 2-        begin
  608   0076 --
  609   0076 --        for j := 1 to MAX_BOOL_DIGIT do
  610   0094 3-          begin
  611   0094 --            with ASCII_Stack(.i.) do
  612   00B4 --            bool_prime(.j.) := ASCII_Arr(.Char_No,idx.).bool_prime(.j.);
  613   0120 -3          end;
  614   0142 --
  615   0142 -2        end;  {with}
  616   0142 -1      end;
  617   0164 --
  618   0164 -0 C  end;
  619   019C --
  620   019C --
  621   019C --
  622   019C --
  623   019C --
  624   019C --
  625   019C --
  626   019C --
  627   019C --
  628   019C --
  629   019C --
  630   019C --
  631   019C -- C  function EX_OR (Bit_X, .Bit_Y: boolean): boolean;
  632   0050 --
  633   0000 0- C  begin
    4   0018 --
  635   0018 --         if Bit_X and Bit_Y then EX_OR := false
  636   0034 --
  637   0034 --         else if Bit_X and (not Bit_Y) then EX_OR := true
  638   005E --
```

```
639  005E --          else if (not Bit_X) and Bit_Y then EX_OR := true
640  008A --
     008A --          else if (not Bit_X) and (not Bit_Y) then EX_OR := false;
642  00C0 --
643  00C0 -0 C  end;
644  00DC --
645  00DC --
646  00DC --
647  00DC --
648  00DC --
649  00DC --
650  00DC --
651  00DC -- C  procedure First_Level_Ex_Oring(Code_No : integer);
652  0044 --
653  0044 --    var  i, j : integer;
654  004C --
655  0000 0- C  begin
656  0012 --
657  0012 --          i := 1;
658  001A --          j := 1;
659  0022 --
660  0022 1-          repeat
661  0022 --             with Bool_Stack(.Code_No.) do
662  0042 --               EXOR1_Arr(.j.) := EX_OR(Bool_Key_Arr(.i.).bool_prime(.Bit_No.)
663  0080 --                   ,Bool_Key_Arr(.i+1.).bool_prime(.Bit_No.));
664  00F0 --               i := i + 2;
665  0102 --               j := j + 1;
666  0114 -1          until (j > NUM_FIRST_EXOR);
667  0126 --
668  0126 -0 C  end;
669  0150 --
670  0150 --
671  0150 --
672  0150 --
673  0150 --
674  0150 --
675  0150 --
676  0150 --
677  0150 --
678  0150 -- C  procedure Second_Level_Ex_Oring;
679  0040 --
680  0040 --    var  i, j : integer;
681  0048 --
682  0000 0- C  begin
683  0012 --
684  0012 --          i := 1;
685  001A --          j := 1;
686  0022 --
687  0022 1-          repeat
688  0022 --               EXOR2_Arr(.j.) := EX_OR(EXOR1_Arr(.i.), EXOR1_Arr(.i+1.));
689  009C --               i := i + 2;
690  00AE --               j := j + 1;
691  00C0 -1          until (j > NUM_SECOND_EXOR);
692  00D2 --
693  00D2 -0 C  end;
694  00EC --
695  00EC --
696  00EC --
```

```
00EC --
00EC --
00EC --
00EC --
00EC --
00EC -- C  procedure Third_Level_Ex_Oring;
0040 --    var  i, j : integer;
0048 --
0000 0- C  begin
0012 --
0012 --         i := 1;
001A --         j := 1;
0022 --
0022 1-         repeat
0022 --              EXOR3_Arr(.j.) := EX_OR(EXOR2_Arr(.i.), EXOR2_Arr(.i+1.));
009C --
009C --              i := i + 2;
00AE --              j := j + 1;
00C0 -1         until (j > NUM_THIRD_EXOR);
00D2 --
00D2 -0 C  end;
00E8 --
00E8 --
00E8 --
00E8 --
00E8 --
00E8 --
00E8 --
00E8 --
00E8 --
00E8 -- C  procedure Last_Ex_Oring_And_Store_An_Addr_Bit(Code_No : integer);
0044 --
0000 0- C  begin
0012 --      with Code_Stack(.Code_No.) do
0032 --         Hashed_Addr(.Bit_No.) := EX_OR(EXOR3_Arr(.1.), EXOR3_Arr(.2.));
007E -0 C  end;
0098 --
0098 --
0098 --
0098 --
0098 --
0098 --
0098 --
0098 --
0098 --
0098 -- C  procedure Bool_To_Int_Convert (var addrs : Addr_Type_Array);
0044 --
0044 --    var  sum : integer;
0048 --         i, j : integer;
0050 --         offset : integer;
0054 --
0000 0- C  begin
0012 --         for i := 1 to MAX_STACK_SIZE do
0030 1-           begin
0030 --              addrs(.i.) := NUL;
0052 -1           end;
0074 --
0074 --         for i := Stk_Pt to MAX_STACK_SIZE do
```

```
755  009E 1-           begin
756  009E --              sum := 0;
     00A4 --              for j := MAX_BUCKET_ADDR_BITS+Stk_Pt downto 1+Stk_Pt do
758  00E8 2-                begin
759  00E8 --                   with Code_Stack(.i.) do
760  0108 3-                     begin
761  0108 --                        if Hashed_Addr(.j.) then
762  0130 --                             sum := 2 * sum + 1
763  0144 --                        else
764  0150 --                             sum := 2 * sum;
765  0168 -3                     end;
766  0168 -2                end;
767  018A --              addrs(.i.) := sum;
768  01B2 -1           end;
769  01D4 -0 C  end;
770  01FC --
771  01FC --
772  01FC --
773  01FC --
774  01FC --
775  01FC --
776  01FC --
777  01FC --
778  01FC --
779  01FC --
780  01FC --
781  01FC --
782  01FC --
   3 01FC --     {------------------- Hash_Relation -------------------------------}
784  01FC --
785  0000 0- B  begin
786  0012 --         while pt <> nil do
787  0024 1-           begin
788  0024 --              for Char_No := 1 to NUM_IDENT_CHAR do
789  0042 2-                begin
790  0042 --                   Key_Char := pt@.Key_Arr(.Char_No.);
791  0072 --
792  0072 --                   {Read character by character for a key looking up the
793  0072 --                   corresponding prime number and convert it to binary number}
794  0072 --                   Look_Up_Char_In_Prime_Num_Table(index);
795  007E --
796  007E --                   Save_Binary_Prime_Num(index);
797  0090 --
798  0090 -2                end;
799  00B2 --
800  00B2 --
801  00B2 --              for Coder_No := Stk_Pt to MAX_STACK_SIZE do
802  00DC 2-                begin
803  00DC --
804  00DC --              {Get the first digit of the binary prime numbers which
805  00DC --               converted from a character, and do the Exclusive-Or operation
806  00DC --               to get the first bit for the hashed address. Repeat this
807  00DC --               process up to the last digit.}
   3 00DC --
809  00DC --                  for Bit_No :=1 to MAX_BOOL_DIGIT do
810  00FA --
811  00FA 3-                    begin
812  00FA --                       First_Level_Ex_Oring(Coder_No);
```

```
  813  0106 --                    Second_Level_Ex_Oring;
  814  010A --                    Third_Level_Ex_Oring;
    ;  010E --                    Last_Ex_Oring_And_Store_An_Addr_Bit(Coder_No);
  816  011A -3                  end;
  817  013C -2                end;
  818  015E --
  819  015E --              Bool_To_Int_Convert(Addr_Array);
  820  016A --
  821  016A --              Hash_Count := Hash_Count + 1;
  822  0188 --
  823  0188 --    if SIM_DEBUG then
  824  0190 2-     begin
  825  0190 --       write(' ');
  826  01C8 --       for i:= 1 to NUM_IDENT_CHAR do
  827  01E6 --            write(pt@.Key_Arr(.i.));
  828  0268 --
  829  0268 --
  830  0268 --       for i := 1 to MAX_STACK_SIZE do
  831  0286 --            write(Addr_Array(.i.):7);
  832  02D8 --       writeln;
  833  02E4 -2     end;
  834  02E4 --
  835  02E4 --              Key_Pt := pt;
  836  02F6 --              Next_Pt := pt@.next;
  837  0312 --              if Source_Relation then
  838  0322 2-               begin
  839  0322 --                with stack(.Stk_Pt.) do
  840  0348 3-                   begin
    1  0348 --                      Key_Pt@.next := Source_Bucket
  842  0352 --                                          (.Addr_Array(.Stk_Pt.).);
  843  038E --                      Source_Bucket(.Addr_Array(.Stk_Pt.).)
  844  03B8 --                              := Key_Pt;
  845  03CA -3                   end;
  846  03CA --                      for i := Stk_Pt to MAX_STACK_SIZE do
  847  03F4 --                           stack(.i.).Bit_Arr(.Addr_Array(.i.).) := true;
  848  0462 -2               end
  849  0462 --              else
  850  0466 2-               begin
  851  0466 --                  ok := true;
  852  046E --                  i := Stk_Pt;
  853  0482 --                  while ok and (i <= MAX_STACK_SIZE) do
  854  04AC 3-                   begin
  855  04AC --                      with stack(.i.) do
  856  04CC 4-                       begin
  857  04CC --                       if not Bit_Arr(.Addr_Array(.i.).) then
  858  0504 --                             ok := false;
  859  050A -4                       end;
  860  050A --                      i := i + 1;
  861  051C -3                   end;
  862  0520 --
  863  0520 --                  if ok then
  864  0530 3-                   begin
  865  0530 --                      with stack(.Stk_Pt.) do
    ;  0556 4-                       begin
  867  0556 --                          Key_Pt@.next :=
  868  0560 --                                  Target_Bucket(.Addr_Array(.Stk_Pt.).);
  869  059C --                          Target_Bucket(.Addr_Array(.Stk_Pt.).)
  870  05C6 --                                  := Key_Pt;
```

```
871   05D8 -4                          end;
872   05D8 -3                      end;
873   05D8 -2                  end;
874   05D8 --                  pt := Next_Pt;
875   05EA -1              end;
876   05EE -0 B  end;
877   0624 --
878   0624 --
879   0624 --
880   0624 --
881   0624 --
882   0624 --
883   0624 --
884   0624 --
885   0624 -- B  procedure Eliminate_Needless_Source_Relations;
886   0040 --
887   0040 --    var
888   0040 --        i : integer;
889   0044 --
890   0000 0- B  begin
891   0012 --        for i := 0 to BUCKET_SIZE do
892   002E 1-          begin
893   002E --             with stack(.Stk_Pt.) do
894   0054 2-               begin
895   0054 --                  if Target_Bucket(.i.) = nil then
896   0078 3-                    begin
897   0078 --                       Source_Bucket(.i.) := nil;
898   0090 --                       Bit_Arr(.i.) := false;
899   00A8 -3                    end;
900   00A8 -2               end;
901   00A8 -1          end;
902   00CA -0 B  end;
903   00EC --
904   00EC --
905   00EC --
906   00EC --
907   00EC --
908   00EC --
909   00EC --    {------- Hash_Source_And_Target_Relations start from here --------}
910   00EC --
911   0000 0- A  begin   {Hash_Source_And_Target_Relations}
912   0012 --        Source_Flag := true;
913   001A --        point := Hd_Source_Pt;
914   0032 --        Hash_Relation (point, Source_Flag);
915   0056 --
916   0056 --
917   0056 --    if SIM_DEBUG then
918   005E 1-      begin
919   005E --        for j := Stk_Pt to MAX_STACK_SIZE do
920   0088 2-          begin
921   0088 --             writeln;
922   0094 --             write(' ------------');
923   00AC --             writeln(' BIT_ARRAY at Stack Level : ',j:1,' ------------');
924   00EE --             write(' ');
925   0126 --             for i := 0 to BUCKET_SIZE do
926   0142 3-               begin
927   0142 --
928   0142 --                  if stack(.j.).Bit_Arr(.i.) then
```

```
929  0184 --                      write('1')
930  01BC --                  else
     01C0 --                      write('0');
932  01F8 --
933  01F8 --                  if (i mod 50) = 49 then
934  020C 4-                    begin;
935  020C --                      writeln;
936  0218 -4                    end;
937  0218 --                  if (i mod 10) = 9 then
938  022C --                      write(' ');
939  0264 -3              end;
940  0286 --             writeln;
941  0292 -2          end;
942  02B4 --          writeln;
943  02C0 --
944  02C0 -1     end;
945  02C0 --
946  02C0 --
947  02C0 --          Source_Flag := false;
948  02C6 --          point := Hd_Target_Pt;
949  02DE --          Hash_Relation (point, Source_Flag);
950  0302 --
951  0302 --          Eliminate_Needless_Source_Relations;
952  0306 --
953  0306 --
954  0306 --
955  0306 --     if SIM_DEBUG then
956  030E 1-       begin
957  030E --       writeln;
958  031A --       writeln(' Hash_Source_And_Target called at Stack Level : ',Stk_Pt :1);
959  0356 --       with stack(.Stk_Pt.) do
960  037C 2-            begin
961  037C --              for i := 0 to BUCKET_SIZE do
962  0398 3-                begin
963  0398 --
964  0398 --                   ptl := Source_Bucket(.i.);
965  03BC --                   if ptl <> nil then
966  03CE 4-                     begin
967  03CE --                      writeln;
968  03DA --                      writeln(' --- Source Bucket No. ',i:1,' ---');
969  041C -4                     end;
970  041C --                   while ptl <> nil do
971  042E 4-                     begin
972  042E --                      write(' ');
973  0466 --                      for j := 1 to NUM_IDENT_CHAR do
974  0484 --                           write(ptl@.Key_Arr(.j.));
975  0506 --
976  0506 --                      for j := 1 to NUM_IDENT_CHAR do
977  0524 --                           write(ptl@.First_Name_Arr(.j.));
978  05A2 --                      writeln;
979  05AE --                      ptl := ptl@.next;
980  05CA -4                     end;
981  05CE --
     05CE --                   ptl := Target_Bucket(.i.);
983  05F2 --                   if ptl <> nil then
984  0604 4-                     begin
985  0604 --                      writeln;
986  0610 --                      writeln(' --- Target Bucket No. ',i:3,' ---');
```

```
  987   0652 -4                    end;
  988   0652 --                  while ptl <> nil do
        0664 4-                    begin
  990   0664 --                     write(' ');
  991   069C --                     for j := 1 to NUM_IDENT_CHAR do
  992   06BA --                          write(ptl@.Key_Arr(.j.));
  993   073C --
  994   073C --                     for j := 1 to NUM_IDENT_CHAR do
  995   075A --                          write(ptl@.First_Name_Arr(.j.));
  996   07D8 --                     writeln;
  997   07E4 --                     ptl := ptl@.next;
  998   0800 -4                    end;
  999   0804 -3                end;  {for}
 1000   0826 -2            end;  {with}
 1001   0826 -1      end;
 1002   0826 --
 1003   0826 --
 1004   0826 -0 A  end;
 1005   0904 --
 1006   0904 --
 1007   0904 --
 1008   0904 --
 1009   0904 --
 1010   0904 --
 1011   0904 --
 1012   0904 --
 1013   0904 -- A  procedure Clear_Current_Upper_Part_Of_Stack;
 1014   0040 --
        0040 --    var
 1016   0040 --         i, j : integer;
 1017   0000 0- A  begin
 1018   0012 --         for i := Stk_Pt to MAX_STACK_SIZE do
 1019   003C 1-           begin
 1020   003C --             with stack(.i.) do
 1021   005C 2-               begin
 1022   005C --                 for j := 0 to BUCKET_SIZE do
 1023   0078 3-                   begin
 1024   0078 --                     Bit_Arr(.j.) := false;
 1025   0090 --                     Source_Bucket_Arr(.j.) := nil;
 1026   00A8 --                     Target_Bucket_Arr(.j.) := nil;
 1027   00C0 -3                   end;
 1028   00E2 --                 Next_Bit := NUL;
 1029   00EA -2               end;
 1030   00EA -1           end;
 1031   010C -0 A  end;
 1032   012C --
 1033   012C --
 1034   012C --
 1035   012C --
 1036   012C --
 1037   012C --
 1038   012C --
 1039   012C --
        012C -- A  function Only_One_Bit_Set_After_Hash(var addr: integer) : boolean;
 1041   004C --
 1042   004C --    var
 1043   004C --         flag : boolean;
 1044   0050 --         done : boolean;
```

```
1045  0054 --          i, j : integer;
1046  005C --
  7   0000 0- A  begin
1048  0018 --          i := Stk_Pt;
1049  002C --          addr := NUL;
1050  0038 --          done := false;
1051  003E --          while (not done) and (i <= MAX_STACK_SIZE) do
1052  006C 1-            begin
1053  006C --              flag := false;
1054  0072 --              with stack(.i.) do
1055  0092 2-                begin
1056  0092 --                  j := 0;
1057  0098 --                  while (not done) and (j <= BUCKET_SIZE) do
1058  00C6 3-                    begin
1059  00C6 --                      if Bit_Arr(.j.) and (not flag) then
1060  00F8 4-                        begin
1061  00F8 --                          flag := true;
1062  0100 --                          if i = Stk_Pt then
1063  0120 5-                            begin
1064  0120 --                              addr := j;
1065  0132 --                              Addr_Num := j;
1066  0146 -5                            end;
1067  0146 -4                        end
1068  0146 --                      else
1069  014A --                          if Bit_Arr(.j.) and flag then
1070  0178 4-                            begin
1071  0178 --                              done := true;
1072  0180 --                              if i = Stk_Pt then
   3  01A0 --                                  Next_Bit := j;
1074  01AE -4                            end;
1075  01AE --                      j := j + 1;
1076  01C0 -3                    end;
1077  01C4 -2                end;
1078  01C4 --
1079  01C4 --              i := i + 1;
1080  01D6 -1            end;
1081  01DA --           if done then
1082  01EA 1-            begin
1083  01EA --              Only_One_Bit_Set_After_Hash := false;
1084  01F0 --       if SIM_DEBUG then
1085  01F8 2-          begin
1086  01F8 --            writeln;
1087  0204 --            write(' Only_One_Bit_Set? is false with an address : ', addr:1);
1088  0238 --            writeln(' and the next address : ',stack(.Stk_Pt.).Next_Bit:1);
1089  0292 -2          end;
1090  0292 -1             end
1091  0292 --           else
1092  0296 1-            begin
1093  0296 --              Only_One_Bit_Set_After_Hash := true;
1094  029E --              if addr = NUL then
1095  02B4 --                  addr := EMPTY;
1096  02C0 --       if SIM_DEBUG then
1097  02C8 2-          begin
  8   02C8 --            writeln;
1099  02D4 --            write(' Only_One_Bit_Set? is true with an address : ', addr:1);
1100  0308 --            writeln(' and the next address : ',stack(.Stk_Pt.).Next_Bit:1);
1101  0362 -2          end;
1102  0362 -1             end;
```

```
1103  0362 -0 A  end;
1104  0428 --
1     0428 --
1106  0428 --
1107  0428 --
1108  0428 --
1109  0428 --
1110  0428 -- A  procedure Merge_Relations_And_Print_Out (addr: integer);
1111  0044 --
1112  0044 --    var
1113  0044 --         Pt_Source, Pt_Target, Pt_T : LINK;
1114  0050 --         i : integer;
1115  0054 --
1116  0000 0- A  begin
1117  0012 --      if addr <> EMPTY then
1118  0024 1-       begin
1119  0024 --         if SIM_DEBUG then
1120  002C 2-           begin
1121  002C --             write(' Merge Relations At Stack Level ',Stk_Pt : 1);
1122  0062 --             writeln(' with  Bucket Number ',addr : 3);
1123  0098 -2           end;
1124  0098 --         writeln;
1125  00A4 --         with stack(.Stk_Pt.) do
1126  00CA 2-           begin
1127  00CA --             Pt_Source := Source_Bucket(.addr.);
1128  00EE --             Pt_Target := Target_Bucket(.addr.);
1129  0112 -2           end;
1130  0112 --         while Pt_Source <> nil do
1     0124 2-           begin
1132  0124 --             Pt_T := Pt_Target;
1133  0136 --             while Pt_T <> nil do
1134  0148 3-               begin
1135  0148 --                 write(' ');
1136  0180 --                 for i := 1 to NUM_IDENT_CHAR do
1137  019E --                      write(Pt_Source@.Key_Arr(.i.));
1138  0220 --
1139  0220 --                 for i := 1 to NUM_IDENT_CHAR do
1140  023E --                      write(Pt_Source@.First_Name_Arr(.i.));
1141  02BC --
1142  02BC --                 for i := 1 to NUM_IDENT_CHAR do
1143  02DA --                      write(Pt_T@.Key_Arr(.i.));
1144  035C --
1145  035C --                 for i := 1 to NUM_IDENT_CHAR do
1146  037A --                      write(Pt_T@.First_Name_Arr(.i.));
1147  03F8 --
1148  03F8 --                 Pt_T := Pt_T@.next;
1149  0414 --                 Result_Count := Result_Count + 1;
1150  0432 --                 writeln;
1151  043E -3               end;
1152  0442 --             writeln;
1153  044E --             writeln;
1154  045A --             Pt_Source := Pt_Source@.next;
1155  0476 -2           end;
1     047A -1       end
1157  047A --      else
1158  047E 1-       begin
1159  047E --        if SIM_DEBUG then
1160  0486 --             writeln(' There are no keys to be merged in this level.');
```

```
1161  04A4 -1        end;
1162  04A4 -0 A  end;
1     0544 --
1164  0544 --
1165  0544 --
1166  0544 --
1167  0544 --
1168  0544 --
1169  0544 --
1170  0544 --
1171  0544 -- A  procedure Save_Next_Addr_Bit;
1172  0040 --
1173  0040 --    var
1174  0040 --          i : integer;
1175  0044 --          found : boolean;
1176  0048 --          num : integer;
1177  004C --
1178  0000 0- A  begin
1179  0012 --          num := NUL;
1180  001A --          found := false;
1181  0020 --          i := Addr_Num + 1;
1182  0038 --          while not found and (i <= BUCKET_SIZE) do
1183  0066 1-            begin
1184  0066 --               if stack(.Stk_Pt.).Bit_Arr(.i.) then
1185  00AE 2-                 begin
1186  00AE --                    stack(.Stk_Pt.).Next_Bit := i;
1187  00E0 --                    found := true;
1188  00E8 -2                 end;
1189  00E8 --               i := i + 1;
1190  00FA -1            end;
1191  00FE --
1192  00FE --          if (not found) and (i > BUCKET_SIZE) then
1193  012C --               stack(.Stk_Pt.).Next_Bit := NUL;
1194  0158 -0 A  end;
1195  0178 --
1196  0178 --
1197  0178 --
1198  0178 --
1199  0178 --
1200  0178 --
1201  0178 -- A  function No_More_Next_Addr: boolean;
1202  0048 --
1203  0000 0- A  begin
1204  0018 --          if stack(.Stk_Pt.).Next_Bit = NUL then
1205  004E 1-            begin
1206  004E --               Addr_Num := NUL;
1207  005C --               No_More_Next_Addr := true;
1208  0064 -1            end
1209  0064 --          else
1210  0068 1-            begin
1211  0068 --               No_More_Next_Addr := false;
1212  006E --               Addr_Num := stack(.Stk_Pt.).Next_Bit;
1213  00A6 -1            end;
14    00A6 -0 A  end;
      00D4 --
      00D4 --
      00D4 --
      00D4 --
```

```
1219   00D4 --
1220   00D4 --
1  1   00D4 --
1222   00D4 --
1223   00D4 -- A  procedure Assign_Source_And_Target;
1224   0040 --
1225   0000 0- A  begin
1226   0012 --      if SIM_DEBUG then
1227   001A 1-      begin
1228   001A --        writeln;
1229   0026 --        write(' Assign_Source_And_T ==> Address Number : ',Addr_Num : 3);
1230   005C --        writeln('  Stack Number : ',Stk_Pt :1);
1231   0098 -1      end;
1232   0098 --        Hd_Source_Pt := stack(.Stk_Pt.).Source_Bucket(.Addr_Num.);
1233   00EE --        Hd_Target_Pt := stack(.Stk_Pt.).Target_Bucket(.Addr_Num.);
1234   0144 -0 A  end;
1235   01A4 --
1236   01A4 --
1237   01A4 --
1238   01A4 --
1239   01A4 --
1240   01A4 --
1241   01A4 --
1242   01A4 --
1243   01A4 --
1244   01A4 --
1245   01A4 --
1246   01A4 --
   7   01A4 -- A  procedure pop;
1248   0040 --
1249   0000 0- A  begin
1250   0012 --        Stk_Pt := Stk_Pt - 1;
1251   0030 --        if Stk_Pt < 1 then
1252   0048 --             writeln(' Stack Underflow !');
1253   0066 --
1254   0066 --    if SIM_DEBUG then
1255   006E 1-    begin
1256   006E --    writeln;
1257   007A --    writeln(' Popped, Stack Pointer is ',Stk_Pt:1,' from ',Stk_Pt+1:1);
1258   00EA -1    end;
1259   00EA -0 A  end;
1260   013C --
1261   013C --
1262   013C --
1263   013C --
1264   013C --
1265   013C --
1266   013C --
1267   013C --
1268   013C --
1269   013C --
1270   013C --
1271   013C --
'  2   013C --
1273   013C --
1274   013C --
1275   013C -- A  procedure push;
1276   0040 --
```

```
1277  0000 0- A  begin
1278  0012 --         Stk_Pt := Stk_Pt + 1;
1279  0030 --         if Stk_Pt > MAX_STACK_SIZE then
1280  0048 --              writeln(' Stack Overflow !');
1281  0066 --    if SIM_DEBUG then
1282  006E 1-    begin
1283  006E --    writeln;
1284  007A --    writeln(' Pushed, Stack Pointer is ',Stk_Pt:1,' from ',Stk_Pt-1:1);
1285  00EA -1    end;
1286  00EA -0 A  end;
1287  0140 --
1288  0140 --
1289  0140 --
1290  0140 --
1291  0140 --
1292  0140 --
1293  0140 --
1294  0140 --
1295  0140 --
1296  0140 -- A  function Bottom_Of_Stack : boolean;
1297  0048 --
1298  0000 0- A  begin
1299  0018 --         if Stk_Pt = 1 then
1300  0030 --              Bottom_Of_Stack := true
1301  0030 --         else
1302  003C --              Bottom_Of_Stack := false;
1303  0042 -0 A  end;
1304  0064 --
1305  0064 --
1306  0064 --
1307  0064 --
1308  0064 --
1309  0064 --
1310  0064 --
1311  0064 --
1312  0064 --
1313  0064 --
1314  0064 --
1315  0064 -- A  procedure Print_Statistics;
1316  0040 --
1317  0000 0- A  begin
1318  0012 --         writeln;
1319  001E --         writeln;
1320  002A --         writeln;
1321  0036 --         write(' THE TOTAL NUMBER OF KEYS IN THE SOURCE RELATION : ');
1322  004E --         writeln(Source_Count:4);
1323  0078 --         writeln;
1324  0084 --         write(' THE TOTAL NUMBER OF KEYS IN THE TARGET RELATION : ');
1325  009C --         writeln(Target_Count:4);
1326  00C6 --         writeln;
1327  00D2 --         write(' THE TOTAL NUMBER OF KEYS IN THE RESULT RELATION : ');
1328  00EA --         writeln(Result_Count:4);
1329  0114 --         writeln;
1330  0120 --         writeln;
1331  012C --         writeln;
1332  0138 --         write(' THE TOTAL NUMBER OF HASH CODER USED IN THE JOIN : ');
1333  0150 --         writeln(Hash_Count:4);
1334  017A --         writeln;
```

```
1335  0186 -0 A  end;
1336  0268 --
1337  0268 --
1338  0268 --
1339  0268 --
1340  0268 --
1341  0268 --
1342  0268 --
1343  0268 --
1344  0268 --
1345  0268 --    {***************** MAIN PROGRAM STARTS HERE ******************}
1346  0268 --
1347  0000 0-    begin
1348  0062 --
1349  0062 --    Initialization;
1350  0066 --
1351  0066 --    Form_Source_And_Target_Relations;
1352  006A --
1353  006A 1-    repeat
1354  006A --       Clear_Current_Upper_Part_Of_Stack;
1355  006E --
1356  006E --       Hash_Source_And_Target_Relations;
1357  0072 --
1358  0072 --       if Only_One_Bit_Set_After_Hash(Hash_Value) then
1359  008A 2-       begin
1360  008A --          Merge_Relations_And_Print_Out(Hash_Value);
1361  00A2 --          if No_More_Next_Addr then
1362  00B0 3-          begin
1363  00B0 --             if Bottom_Of_Stack then
1364  00BE --                finish := true
1365  00BE --             else
1366  00D0 4-             begin
1367  00D0 --                pop;
1368  00D4 --                if No_More_Next_Addr then
1369  00E2 5-                begin
1370  00E2 --                   if Bottom_Of_Stack then
1371  00F0 --                      finish := true
1372  00F0 --                   else
1373  0102 6-                   begin
1374  0102 --                      pop;
1375  0106 --                      if No_More_Next_Addr then
1376  0114 7-                      begin
1377  0114 --                         if Bottom_Of_Stack then
1378  0122 --                            finish := true
1379  0122 --                         else
1380  0134 8-                         begin
1381  0134 --                            pop;
1382  0138 --                            if No_More_Next_Addr then
1383  0146 9-                            begin
1384  0146 --                               if Bottom_Of_Stack then
1385  0154 --                                  finish := true
1386  0154 --                               else
1387  0166 0-                               begin
1388  0166 --                                  pop;
1389  016A --                                  if No_More_Next_Addr then
1390  0178 1-                                  begin
1391  0178 --                                     if Bottom_Of_Stack then
1392  0186 --                                        finish := true
```

```
1393  0186 --                                    else
1394  0198 2-                                    begin
1395  0198 --                                       Assign_Source_And_Target;
1396  019C --                                       Save_Next_Addr_Bit;
1397  01A0 --                                       push;
1398  01A4 -2                                    end;
1399  01A4 -1                                 end;
1400  01A4 -0                              end;
1401  01A4 -9                           end
1402  01A4 --                           else
1403  01A8 9-                           begin
1404  01A8 --                              Assign_Source_And_Target;
1405  01AC --                              Save_Next_Addr_Bit;
1406  01B0 --                              push;
1407  01B4 -9                           end;
1408  01B4 -8                        end;
1409  01B4 -7                     end
1410  01B4 --                     else
1411  01B8 7-                     begin
1412  01B8 --                        Assign_Source_And_Target;
1413  01BC --                        Save_Next_Addr_Bit;
1414  01C0 --                        push;
1415  01C4 -7                     end;
1416  01C4 -6                  end;
1417  01C4 -5               end
1418  01C4 --               else
1419  01C8 5-               begin
1420  01C8 --                  Assign_Source_And_Target;
1421  01CC --                  Save_Next_Addr_Bit;
1422  01D0 --                  push;
1423  01D4 -5               end;
1424  01D4 -4            end;
1425  01D4 -3         end
1426  01D4 --         else
1427  01D8 3-         begin
1428  01D8 --            Assign_Source_And_Target;
1429  01DC --            Save_Next_Addr_Bit;
1430  01E0 --            push;
1431  01E4 -3         end;
1432  01E4 -2      end
1433  01E4 --      else
1434  01E8 2-      begin
1435  01E8 --         Assign_Source_And_Target;
1436  01EC --         Save_Next_Addr_Bit;
1437  01F0 --         push;
1438  01F4 -2      end
1439  01F4 -1   until finish;
1440  020A --
1441  020A --   Print_Statistics;
1442  020E --
1443  020E -0   end.

:AAEC PASCAL 2.0 COMPILATION CONCLUDED*

*NO ERRORS DETECTED IN PASCAL PROGRAM*
```

If the first character of a first name is between "A" and "V",
the name will be included in the source relation.
Otherwise, the name will be included in the target relation.

Beall	Preston	Beall	Walter S
Beall	Preston	Beall	Willie Jr
Beall	Richard L	Beall	Walter S
Beall	Richard L	Beall	Willie Jr
Beall	Russell	Beall	Walter S
Beall	Russell	Beall	Willie Jr
Beall	Ruth	Beall	Walter S
Beall	Ruth	Beall	Willie Jr
Beall	Sarah	Beall	Walter S
Beall	Sarah	Beall	Willie Jr
Beall	Sidney L	Beall	Walter S
Beall	Sidney L	Beall	Willie Jr
B ll	Stuart J	Beall	Walter S
Beall	Stuart J	Beall	Willie Jr
Beall	Thomas C	Beall	Walter S
Beall	Thomas C	Beall	Willie Jr
Beall	Timothy R	Beall	Walter S
Beall	Timothy R	Beall	Willie Jr
Beall	Todd	Beall	Walter S
Beall	Todd	Beall	Willie Jr
Beall	Vernon L	Beall	Walter S
Beall	Vernon L	Beall	Willie Jr
Macey	E Christopher	Macey	Wayne
Cooper	Bruce Michael	Cooper	William H
C er	Bruce Michael	Cooper	Williams
Cooper	Adren	Cooper	William H
Cooper	Adren	Cooper	Williams

Cooper	Aileen	Cooper	William H
Cooper	Aileen	Cooper	Williams
Cooper	Albert	Cooper	William H
Cooper	Albert	Cooper	Williams
Cooper	Alfred D	Cooper	William H
Cooper	Alfred D	Cooper	Williams
Cooper	Alla	Cooper	William H
Cooper	Alla	Cooper	Williams
Cooper	Alvin	Cooper	William H
Cooper	Alvin	Cooper	Williams
Cooper	Arthur I	Cooper	William H
Cooper	Arthur I	Cooper	Williams
Cooper	Benjamin	Cooper	William H
Cooper	Benjamin	Cooper	Williams
Cooper	Bradford	Cooper	William H
Cooper	Bradford	Cooper	Williams
Cooper	Brooks	Cooper	William H
Cooper	Brooks	Cooper	Williams
Cooper	Carleton	Cooper	William H
Cooper	Carleton	Cooper	Williams
Cooper	Charles E	Cooper	William H
Cooper	Charles E	Cooper	Williams
Cooper	Christine	Cooper	William H
Cooper	Christine	Cooper	Williams
Cooper	Thomas E	Cooper	William H
Cooper	Thomas E	Cooper	Williams
Cooper	Timothy	Cooper	William H
Cooper	Timothy	Cooper	Williams
Cooper	Tommye	Cooper	William H
Cooper	Tommye	Cooper	Williams
Knox	Robert Jr	Knox	Wilfred C
Knox	Tommy	Knox	Wilfred C

Knox	Tonya	Knox	Wilfred C
,mith	Marian	Smith	Wm Lewis
Sm. .n	Mary	Smith	Wm Lewis
Smith	Alan	Smith	Wm Lewis
Smith	Chip	Smith	Wm Lewis
Smith	Caroline	Smith	Wm Lewis
Smith	Chester H	Smith	Wm Lewis
Smith	Christine	Smith	Wm Lewis
Smith	Dana	Smith	Wm Lewis
Smith	Darrell W	Smith	Wm Lewis
;mith	Jefferson S	Smith	Wm Lewis
Smith	Kevin R	Smith	Wm Lewis
Smith	Mark E	Smith	Wm Lewis
Smith	Merrill L	Smith	Wm Lewis
Smith	Navarro	Smith	Wm Lewis
Smith	Theodore H	Smith	Wm Lewis
Holden	Vincent E	Holden	Wesley M
Holden	Vincent E	Holden	Warner
Holden	Vincent E	Holden	Wallace
Johnson	Byron	Johnson	Winston E
Johnson	Carrie	Johnson	Winston E
Johnson	Clifford P	Johnson	Winston E

Johnson	Darrell	Johnson	Winston E
Johnson	Dixie L	Johnson	Winston E
Johnson	Edgar M	Johnson	Winston E
Johnson	Emery	Johnson	Winston E
Johnson	Greg	Johnson	Winston E
Johnson	Howard	Johnson	Winston E
Irvin	Harry G	Irvin	Wayne M
Irvin	Harry P	Irvin	Wayne M
Kapur	Vernice	Kapur	William R
Kapur	Vernice	Kapur	Wallace W
Kapur	Vernice	Kapur	Wade L
Kapur	Truman	Kapur	William R
Kapur	Truman	Kapur	Wallace W
K ur	Truman	Kapur	Wade L
Kapur	Tom E	Kapur	William R
Kapur	Tom E	Kapur	Wallace W
Kapur	Tom E	Kapur	Wade L

THE TOTAL NUMBER OF KEYS IN THE SOURCE RELATION : 773

THE TOTAL NUMBER OF KEYS IN THE TARGET RELATION : 27

THE TOTAL NUMBER OF KEYS IN THE RESULT RELATION : 98

THE TOTAL NUMBER OF HASH CODER USED IN THE JOIN : 976

STORAGE SUMMARY IN DECIMAL (HEX) BYTES

STACK	HEAP	UNUSED
452320 (06E6E0)	109208 (01AA98)	437832 (06AE48)

```
 6666666      00000      6666666                EEEEEEEEE  N       NN  DDDDDDDD
66666666    0000000    666666666               EEEEEEEEE  NN      NN  DDDDDDDDD
66      66  00    00   66      66              EE         NNN     NN  DD      DD
66          00    00   66                      EE         NNNN    NN  DD      DD
66666666    00    00   66666666    -------     EEEEEE     NN NN   NN  DD      DD
666666666   00    00   666666666   -------     EEEEEE     NN  NN  NN  DD      DD
66      66  00    00   66      66              EE         NN   NNNN  DD      DD
66      66  00    00   66      66              EE         NN    NNN  DD      DD
666666666    0000000    666666666               EEEEEEEEE  NN      NN  DDDDDDDDD
 6666666      00000      6666666                EEEEEEEEE  NN       N  DDDDDDDD
```

PRINT COMPLETED AT 20:53:21 FOR USER: RSCSREC DIST: SM993C

```
EEEEEEEEE  N       NN  DDDDDDDD                6666666      00000      6666666
EEEEEEEEE  NN      NN  DDDDDDDDD              666666666    0000000    666666666
EE         NNN     NN  DD      DD              66      66  00    00   66      66
EE         NNNN    NN  DD      DD              66          00    00   66
EEEEEE     NN NN   NN  DD      DD    -------   66666666    00    00   66666666
EEEEEE     NN  NN  NN  DD      DD    -------   666666666   00    00   666666666
EE         NN   NNNN  DD      DD              66      66  00    00   66      66
EE         NN    NNN  DD      DD              66      66  00    00   66      66
EEEEEEEEE  NN      NN  DDDDDDDDD              666666666    0000000    666666666
EEEEEEEEE  NN       N  DDDDDDDD                6666666      00000      6666666
```