```
{ -- FLORIDA HIGH SCHOOLS COMPUTING COMPETITION '96 }
{ -- PASCAL PROGRAM SOLUTIONS }


{1.1}
program One1T96;
{ -- This program displays a phrase of the form FHSCC '##. }
  var
    Year: String[4];

begin
  Write ('Enter year: ');  Readln (Year);
  Writeln ('FHSCC ''', Copy(Year,3,2));
end.


{1.2}
program One2T96;
{ -- This program tallies number of frequent flier miles. }
  var
    X, Y: Integer;

begin
  Write ('Enter X: ');  Readln (X);
  Write ('Enter Y: ');  Readln (Y);
  Writeln (X * (1300 + 1300 + 500) + (Y * 5));
end.


{1.3}
program One3T96;
{ -- This program displays middle letter(s) of a word. }
  var
    Word: String[20];
    L, M: Integer;

begin
  Write ('Enter word: ');  Readln (Word);
  L := Length(Word);  M := L div 2;
  If (L mod 2) = 0 then Write (Copy(Word, M, 1));
  Writeln (Copy(Word, M+1, 1));
end.
```

```
{1.4}
program One4T96;
{ -- This program displays area and perimeter of a rectangle. }
  var
    X1, Y1, X2, Y2, Area, Perim: Integer;

begin
  Write ('Enter coordinate 1: ');  Readln (X1, Y1);
  Write ('Enter coordinate 2: ');  Readln (X2, Y2);
  Area  := Abs((X1 - X2) * (Y1 - Y2));
  Perim := (Abs(X1 - X2) + Abs(Y1 - Y2)) * 2;
  Writeln ('AREA = ', Area);
  Writeln ('PERIMETER = ', Perim);
end.


{1.5}
program One5T96;
{ -- This program code-breaks an encrypted secret message. }
  var
    E: String[40];
    M: Char;
    I: Integer;

begin
  Write ('Enter encryption: ');  Readln (E);
  for I := 1 to Length(E) do begin
    M := E[I];
    if M = ' ' then
      Write(M)
    else
      Write (Chr( Ord('Z') - Ord(M) + Ord('A') ));
  end;
  Writeln;
end.


{1.6}
program One6T96;
{ -- This program displays number of floors elevator touches. }
  var
    Floor, Total, Max, LastFloor: Integer;

begin
  repeat
    Write ('Enter floor: ');  Readln (Floor);
    Total := Total + Abs(Floor - LastFloor);
    if Floor > Max then Max := Floor;
    LastFloor := Floor;
  until (Floor = 0);
  { -- 1 is added for the starting ground floor }
  Writeln ('TOTAL FLOORS TOUCHED = ', Total + 1);
  Writeln ('UNIQUE FLOORS TOUCHED = ', Max + 1);
end.
```

```
{1.7}
program One7T96;
{ -- This program displays a person's ratios for buying a house.}
  var
    Loan, Debts, Income, Ratio1, Ratio2: Real;

begin
  Write ('Enter amount of loan: ');   Readln (Loan);
  Write ('Enter amount of debts: ');  Readln (Debts);
  Write ('Enter amount of income: '); Readln (Income);
  Ratio1 := (Loan / Income) * 100;
  Ratio2 := ((Loan + Debts) / Income) * 100;
  Writeln ('RATIOS = ', Ratio1: 4:1, '% / ', Ratio2: 4:1, '%');
  Write   ('DOES ');
  if (Ratio1 > 33) or (Ratio2 > 38) then Write ('NOT ');
  Writeln ('QUALIFY');
end.


{1.8}
program One8T96;
{ -- This program will convert numbers to English or Spanish.}
  const
    N: Array [1..20] of String[6] = ('ONE','TWO','THREE',
    'FOUR','FIVE','SIX','SEVEN','EIGHT','NINE','TEN',
    'UNO','DOS','TRES','CUATRO','CINCO','SEIS','SIETE',
    'OCHO','NUEVE','DIEZ');
  var
    Lang:   Char;
    Num, I: Byte;

begin
  Write ('Enter E or S: ');  Readln (Lang);
  Write ('Enter number: ' ); Readln (Num);
  if Lang = 'S' then I := 10 else I := 0;
  Writeln (N[I + Num]);
end.

{1.9}
program One9T96;
{ -- This program forms a cross from word(s). }
  var
    W: String[20];
    I, L, M: Byte;

begin
  Write ('Enter word(s): ');  Readln (W);
  L := Length(W);  M := (L div 2) + 1;
  for I := 1 to L do
    If I <> M then
      Writeln (' ': M - 1, Copy(W, I, 1))
    else
      Writeln (W);
end.
```

```
{1.10}
program One10T96;
{ -- This program simulates the PRICE IS RIGHT game. }
  var
    Price, Min, I, Dif, Index: Integer;
    A:     Array[1..4] of Integer;

begin
  Write ('Enter actual price: ');  Readln (Price);
  Write ('Enter guesses A, B, C, D: ');
  Readln (A[1], A[2], A[3], A[4]);
  Min := 32000;
  for I := 1 to 4 do
    if A[I] <= Price then begin
      Dif := Price - A[I];
      if Dif < Min then begin
        Min := Dif;  Index := I;
      end;
    end;
  if Index > 0 then
    Writeln ('PERSON ', Copy ('ABCD', Index, 1))
  else
    Writeln ('EVERYONE IS OVER');
end.
```

```
{2.1}
program Two1T96;
{ -- This program will emulate random dart throws. }
  const
    S: Array[1..7] of Byte = (0,2,4,5,10,20,50);
  var
    X, Throw, Total: Byte;

begin
  Randomize;  Throw := 0;
  repeat
    X := Random(7) + 1;  Inc(Throw);
    Write(S[X]);
    Inc(Total, S[X]);
    If Total < 100 then Write (',');
  until (Total >= 100);
  Writeln;
  Writeln (Throw, ' THROWS ACHIEVED A SCORE OF ', Total);
  Writeln;
end.


{2.2}
program Two2T96;
{ -- This program compresses information to save space. }
  var
    S:      String[80];
    I, Ast: Byte;
    Md:     Char;

begin
  Write ('Enter string: ');  Readln (S);
  Ast := 0;
  for I := 1 to Length(S) do begin
    Md := S[I];
    if Md <> '*' then
      begin
        if Ast > 0 then
          begin
            if Ast = 1 then
              Write ('*')
            else
              Write (Ast);
            Ast := 0;
          end;
        Write(Md);
      end
    else
      Inc(Ast)
  end;  { -- for I }
  Writeln;
end.
```

```
{2.3}
program Two3T96;
{ -- This program finds 2 numbers to add to the set 1,3,8. }
  var
    A:            Array[1..5] of Integer;
    I, J, Num, N: Integer;
    Found:        Boolean;

begin
  A[1] := 1; A[2] := 3; A[3] := 8;  N := 3;  I := 0;
  for I := 0 to 999 do begin
    Found := True;
    for J := 1 to N do begin
      Num := A[J] * I + 1;
      if Sqrt(Num) - Trunc(Sqrt(Num + 0.0001)) > 0.0001 then
        Found := False;
    end;
    if Found then begin
      Writeln (I);  Inc(N);  A[N] := I;  if N = 5 then Exit;
    end;
  end;
end.


{2.4}
program Two4T96;
{ -- This program displays the LCM of the first N integers. }
  var
    A:       Array[1..31] of Integer;
    I, J, N: Integer;
    Prod:    Real;

begin
  Write ('Enter N: ');  Readln (N);
  for I := 2 to N do  A[I] := I;
  { -- Produce all the necessary prime factors. }
  for I := 2 to N do
    for J := I + 1 to N do
      if (A[J] Mod A[I]) = 0 then A[J] := A[J] div A[I];

  Prod := 1;
  For I := 2 to N do
    Prod := Prod * A[I];

  Writeln (Prod: 13:0);
end.
```

```
{2.5}
program Two5T96;
{ -- This program will calculate the fractional value. }
  var
    Word:    String[3];
    A:       Array[1..3] of Integer;
    I, N, D: Integer;

begin
  Write ('Enter word: ');  Readln (Word);
  for I := 1 to 3 do
    A[I] := Ord(Word[I]) - Ord('A') + 1;

  N := A[1] * A[2] + A[2] * A[3] + A[1] * A[3];
  D := A[1] * A[2] * A[3];
  for I := D downto 1 do
    if (N mod I = 0) and (D mod I = 0) then begin
      Writeln (N div I, '/', D div I);  Exit
    end;
end.


{2.6}
program Two6T96;
{ -- This program displays the Nth prime in Fibonacci sequence. }
  var
    F:              Array[1..99] of LongInt;
    I, N, J, PNum: Integer;
    Prime:          Boolean;

begin
  F[1] := 1;  F[2] := 1;  F[3] := 2;  PNum := 1;  I := 3;
  Write ('Enter N: ');  Readln (N);
  while (PNum < N) do begin
    Inc(I);
    F[I] := F[I-1] + F[I-2];  Prime := True;
    { -- Check if Fibonacci # is prime (not divis by 2 or odd#) }
    if (F[I] Mod 2 = 0) then Prime := False;
    if Prime then begin
      for J := 3 to Trunc(Sqrt(F[I])) do
        if (F[I] mod J = 0) then Prime := False;
      if Prime then Inc(PNum);
    end;
  end;
  Writeln(F[I]);
end.
```

```
{2.7}
program Two7T96;
{ -- This program sorts phone bills by zip code and phone #. }
  var
    P, Z, PZ: Array[1..8] of LongInt;
    X:        LongInt;
    N, I, J:  Integer;

begin
  N := 0;
  repeat
    Inc(N);
    Write ('Enter phone #, zip: ');  Readln (P[N], Z[N]);
    PZ[N] := Z[N] * 10000 + P[N];
  until (P[N] = 0) and (Z[N] = 0);
  Dec(N);
  for I := 1 to N - 1 do
    for J := I + 1 to N do
      if PZ[I] > PZ[J] then begin
        X := PZ[I];  PZ[I] := PZ[J];  PZ[J] := X;
        X := P[I];  P[I] := P[J];  P[J] := X;
        X := Z[I];  Z[I] := Z[J];  Z[J] := X;
      end;
  for I := 1 to N do  Writeln (P[I]);
end.


{2.8}
program Two8T96;
{ -- This program will display number of runs of letters. }
  var
    Let:          String[80];
    Ch:           Char;
    I, H1, H2:    Integer;
    Half1, Half2: Boolean;

begin
  Write ('Enter letters: ');  Readln (Let);
  Half1 := False;  Half2 := False;
  for I := 1 to Length(Let) do begin
    Ch := Let[I];
    if Pos(Ch, 'ABCDEFGHIJKLM') > 0 then begin
      if Half2 then begin
        Inc(H2);  Half2 := False;
      end;
      Half1 := True;
      end
    else begin
      if Half1 then begin
        Inc(H1);  Half1 := False;
      end;
      Half2 := True;
    end;
  end;
  if Half1 then Inc(H1);
```

```
  if Half2 then Inc(H2);
  Writeln ('RUNS IN 1ST HALF = ', H1);
  Writeln ('RUNS IN 2ND HALF = ', H2);
end.
```

```
{2.9}
program Two9T96;
{ -- This program reverses the order of letters in each word. }
  var
    S:       String[80];
    Md:      Char;
    I, J, L: Integer;
    W:       String[20];
    Pal:     Boolean;

begin
  Write ('Enter string: ');  Readln (S);  S := S + ' ';
  for I := 1 to Length(S) do begin
    Md := S[I];
    if Md = ' ' then begin
      L := Length(W);  Pal := True;
      for J := 1 to L div 2 do
        if Copy(W, J, 1) <> Copy (W, L-J+1, 1) then Pal := False;
      if Pal then
        for J := 1 to Length(W) do  Write('?')
      else
        for J := L downto 1 do  Write(Copy(W,J,1));
      Write (' ');  W := '';
      end
    else
      W := W + Md;
  end;
  Writeln;
end.
```

```
{2.10}
program Two10T96;
{ -- This program determines day of week for a given date. }
  const
    MonNum: Array[1..12] of Byte = (1,4,4,0,2,5,0,3,6,1,4,6);
    D:      Array[1..7]  of String[9] = ('SATURDAY',
            'SUNDAY', 'MONDAY', 'TUESDAY', 'WEDNESDAY',
            'THURSDAY', 'FRIDAY');
  var
    Month, Day, Year, Last2, Sum, R: Integer;
    LeapYear: Boolean;

begin
  Write ('Enter month, day, year: ');  Readln (Month, Day, Year);
  Last2 := Year mod 100;
  Sum := Last2 + (Last2 div 4);
  LeapYear := (Year Mod 4 = 0) and (Year mod 100 > 0);
  LeapYear := LeapYear or (Year mod 400 = 0);
  if (Month < 3) and LeapYear then
    if (Month = 2) then Inc(Sum,3) else  {-- New Month Number }
  else
    Inc(Sum, MonNum[Month]);
  Inc(Sum, Day);
  Case Year of
    1753..1799: Inc(Sum, 4);
    1800..1899: Inc(Sum, 2);
    2000..2099: Inc(Sum, 6);
    2100..2199: Inc(Sum, 4);
  end;
  R := Sum mod 7;
  Writeln (D[R+1]);
end.
```

```
{3.1}
program Thr1T96;
{ -- This program displays the appearance of 3-dimensional book.}
uses Crt;
  const
    Spaces:        String[16] = '                ';
  var
    T1, T2:        String[17];
    Max, Dif, Row: Byte;

begin
  Write ('Enter title 1: ');  Readln (T1);
  Write ('Enter title 2: ');  Readln (T2);
  if Length(T1) > Length(T2) then
    begin
      Max := Length(T1);  Dif := (Max - Length(T2)) div 2;
      T2 := Copy(Spaces, 1, Dif) + T2 + Copy(Spaces, 1, Dif + 1);
    end
  else
    begin
      Max := Length(T2);  Dif := (Max - Length(T1)) div 2;
      T1 := Copy(Spaces, 1, Dif) + T1 + Copy(Spaces, 1, Dif + 1);
    end;
  ClrScr;
  Writeln ('    /---/!');
  Writeln ('   /   / !');
  Writeln ('  /   /  !');
  Writeln (' /   /   !');
  Writeln ('!---!    !');
  for Row := 1 to Max do begin
    Write ('!');
    Write (Copy (T2, Row, 1), ' ');
    Write (Copy (T1, Row, 1), '!');
    if Row < Max - 3 then
      Writeln (' ':4, '!')
    else
      Writeln (' ': Max - Row + 1, '/');
  end;
  Writeln ('!---!/');
end.
```

```
{3.2}
program Thr2T96;
{ -- This program produces a prime factors tree. }
uses Crt;
  var
    P:                        Array[1..100] of Integer;
    Num, Left, Right, Row, Pr, Dividend, L, R: Integer;

begin
  Write ('Enter number: ');  Readln (NUM);
  ClrScr;  Row := 1;  Writeln (' ':5, Num);
  {-- Position of / and \, determine length of Num }
  Left := 5;  Right := Left + Trunc(Ln(Num) / Ln(10)) + 2;
  repeat
    { -- Find smallest prime that divides number }
    if Num mod 2 = 0 then
      Pr := 2
    else begin
      Pr := 1;
      repeat
        Inc(Pr, 2);
      until (Num mod Pr = 0);
    end;
    Dividend := Num div Pr;
    if Dividend > 1 then begin
      Inc(Row);
      GotoXY (Left, Row);   Write ('/');
      GotoXY (Right, Row);  Writeln ('\');
      L := Trunc(Ln(Pr) / Ln(10));
      R := Trunc(Ln(Dividend) / Ln(10));
      Inc(Row);
      GotoXY (Left - L - 1, Row);  Write (Pr);
      GotoXY (Right + 1, Row);     Writeln (Dividend);
      Left := Right;  Right := Right + R + 2;
    end;
    Num := Dividend;
  until Num = 1;
end.
```

```
{3.3}
program Thr3T96;
{ -- This program simulates a "base four" calculator. }
  var
    Num:                 Array[1..10] of String[6];
    Sym:                 Array[1..10] of Char;
    Ch:                  Char;
    N:                   String[6];
    E:                   String[40];
    I, J, K, L, Dig, X: Byte;
    B10, Total, Pow:     LongInt;

begin
  Write ('Enter base 4 expression: ');  Readln (E);
  E := E + '+';  Sym[1] := '+';
  for I := 1 to Length(E) do begin
    Ch := E[I];
    if (Ch = '+') or (Ch = '-') then
      begin
        Inc(J);  Num[J] := N;  Sym[J+1] := Ch;  N := '';
      end
    else
      N := N + Ch;
  end;
  { -- Convert base 4 numbers to base 10 and perform arithmetic }
  for I := 1 to J do begin
    L := Length(Num[I]);  B10 := 0;
    for J := 1 to L do begin
      Dig := Ord(Num[I,J]) - Ord('0');
      Pow := 1;
      for K := 1 to (L - J) do Pow := Pow * 4;
      B10 := B10 + Dig * Pow;
    end;
    if (Sym[I] = '-') then B10 := (-B10);
    Inc(Total, B10);
  end;
  { -- Convert base 10 number to base 4 }
  if Total < 0 then begin
    Write ('-');  Total := (-Total);
  end;
  J := Trunc(Ln(Total) / Ln(4) + 0.001);
  for I := J downto 0 do begin
    Pow := 1;
    for K := 1 to I do Pow := Pow * 4;
    X := Total div Pow;
    Write (X);
    Total := Total - X * Pow;
  end;
  Writeln;
end.
```

```
{3.4}
program Thr4T96;
{ -- This program calculates contractor's pay=time * rate. }
  var
    Rate, Time: Real;
    St, Fi:     String[7];
    FiHour, StHour, StMin, FiMin, Code: Integer;

begin
  Write ('Enter pay/hour: ');     Readln (Rate);
  Write ('Enter start time: ');   Readln (St);
  Write ('Enter finish time: ');  Readln (Fi);
  Val(Copy(St,1,2), StHour, Code);
  Val(Copy(Fi,1,2), FiHour, Code);
  Val(Copy(St,4,2), StMin,  Code);
  Val(Copy(Fi,4,2), FiMin,  Code);
  { -- Adjust for 12AM and times from 1PM - 11PM }
  if StHour = 12 then
    if Copy(St, 6, 2) = 'AM' then Dec(StHour, 12) else
  else
    if Copy(St, 6, 2) = 'PM' then Inc(StHour, 12);

  if FiHour = 12 then
    if Copy(Fi, 6, 2) = 'AM' then Dec(FiHour, 12) else
  else
    if Copy(Fi, 6, 2) = 'PM' then Inc(FiHour, 12);
  {-- Adjust for a late starting time and early morning finish.}
  if StHour > FiHour then Inc(FiHour, 24);
  {-- Compute difference in time (finish - start) }
  Time := (FiHour - StHour) + (FiMin - StMin) / 60;
  {-- If more than half of time is outside normal hours (7AM-5PM)
   -- then add a shift differential of 10% to rate. }
  if ((7 - StHour) + (0 - StMin) / 60) >= (Time / 2) then
    { -- More than half of time is worked before 7AM }
    Rate := Rate * 1.1;
  if ((FiHour - 17) + (FiMin) / 60) >= (Time / 2) then
    { -- More than half of time is worked after 5PM }
    Rate := Rate * 1.1;
  Writeln ('$', Time * Rate: 6:2);
end.
```

```
{3.5}
program Thr5T96;
{ -- This program displays the button that leads to the others. }
  var
    I, J, K, L, R, C, Press:   Byte;
    N:    Array[1..4, 1..4] of Byte;
    D:    Array[1..4, 1..4] of Char;
    A:    Array[1..4, 1..4] of Boolean;
    Row:  String[12];
    Code: Integer;
    Good: Boolean;

begin
  for I := 1 to 4 do begin
    Write ('Enter row: ');  Readln (Row);
    for J := 1 to 4 do begin
      Val(Row[J*3-2], N[I,J], Code);
      D[I,J] := Row[J*3-1];
    end;
  end;
  for I := 1 to 4 do
    for J := 1 to 4 do begin
      for K := 1 to 4 do
        for L := 1 to 4 do  A[K, L] := False;
      R := I;  C := J;  A[R, C] := True;
      Press := 1;  Good := True;
      repeat
        Case D[R,C] of
          'D': Inc(R, N[R,C]);
          'U': Dec(R, N[R,C]);
          'L': Dec(C, N[R,C]);
          'R': Inc(C, N[R,C]);
        end;
        if A[R, C] then
          Good := False
        else begin
          A[R,C] := True;  Inc(Press);
        end;
      until (not Good) or (Press = 16);
      if Press = 16 then begin
        Writeln ('FIRST BUTTON = ', N[I,J], D[I,J]);
        Writeln ('AT ROW = ', I, ', COL = ', J);
        Exit
      end;
    end;  { -- for J }
end.
```

```
{3.6}
program Thr6T96;
{ -- This program will generate odd size magic squares. }
  var
    N, First, Incr, X, Y, I, J, MagicNum: Integer;
    A:              Array[1..13, 1..13] of Integer;

begin
  Write ('Enter order, first number, increment: ');
  Readln (N, First, Incr);
  X := 1;  Y := (N + 1) div 2;  A[X,Y] := First;
  for I := 2 to N * N do begin
    Dec(X);  Inc(Y);
    if X = 0 then X := N;
    if Y > N then Y := 1;
    if A[X,Y] = 0 then
      A[X,Y] := First + Incr * (I - 1)
    else begin
      Inc(X,2);  Dec(Y);
      if X > N then Dec(X, N);
      if Y = 0 then Y := N;
      A[X,Y] := First + Incr * (I - 1);
    end;
  end;
  { -- Display Magic Number and Square }
  MagicNum := 0;
  for I := 1 to N do  Inc(MagicNum, A[I,1]);
  Writeln ('MAGIC NUMBER = ', MagicNum);
  for I := 1 to N do begin
    for J := 1 to N do
      Write (A[I,J]: 4);
    Writeln;
  end;
end.
```

```
{3.7}
program Thr7T96;
{ -- This program will generate 6x6 magic squares. }
  const
    R: Array[1..4] of Byte = (0, 1, 0, 1);
    C: Array[1..4] of Byte = (0, 1, 1, 0);
  var
    N, First, Incr, X, Y, I, J: Integer;
    FirstN, MagicNum, Sq, Temp: Integer;
    A:              Array[1..3, 1..3] of Integer;
    B:              Array[1..6, 1..6] of Integer;

procedure Generate3x3;
{ -- Generate a 3x3 magic square in A[1..3,1..3] }
begin
  for I := 1 to 3 do
    for J := 1 to 3 do A[I,J] := 0;
  N := 3;
  X := 1;  Y := (N + 1) div 2;  A[X,Y] := First;
  for I := 2 to N * N do begin
    Dec(X);  Inc(Y);
    if X = 0 then X := N;
    if Y > N then Y := 1;
    if A[X,Y] = 0 then
      A[X,Y] := First + Incr * (I - 1)
    else begin
      Inc(X,2);  Dec(Y);
      if X > N then Dec(X, N);
      if Y = 0 then Y := N;
      A[X,Y] := First + Incr * (I - 1);
    end;
  end;
end;

begin
  Write ('Enter first number, increment: ');
  Readln (FirstN, Incr);
  { -- Four 3x3 squares are made for the 6x6 matrix B[]
    -- upper-left, bottom-right, upper-right, bottom-left. }
  for Sq := 0 to 3 do begin
    First := FirstN + Sq * 9 * Incr;
    Generate3x3;
    for I := 1 to 3 do
      for J := 1 to 3 do
        B[R[Sq+1] * 3 + I, C[Sq+1] * 3 + J] := A[I,J];
  end;
  { -- Transpose three cells }
  Temp := B[1,1];  B[1,1] := B[4,1];  B[4,1] := Temp;
  Temp := B[2,2];  B[2,2] := B[5,2];  B[5,2] := Temp;
  Temp := B[3,1];  B[3,1] := B[6,1];  B[6,1] := Temp;
  { -- Display Magic Number and 6x6 matrix }
  MagicNum := 0;
  for I := 1 to 6 do  Inc(MagicNum, B[I,1]);
  Writeln ('MAGIC NUMBER = ', MagicNum);
  for I := 1 to 6 do begin
```

```
    for J := 1 to 6 do
      Write (B[I,J]: 4);
    Writeln;
  end;
end.
```

```
{3.8}
program Thr8T96;
{ -- This program will display a pie graph. }
uses Crt;
  const
    L:  Array [1..3] of Char = ('A', 'D', 'N');
    PI: Real = 3.1415926;
  var
    A:  Array[1..21, 1..21] of Byte;
    P:  Array[1..3] of Byte;
    I:  Real;
    Ch: Char;
    J, K, R, X, Y, S, Sum, LSum: Integer;

begin
  Write ('Enter 3 percentages: ');  Readln (P[1], P[2], P[3]);
  ClrScr;
  for J := 1 to 21 do
    for K := 1 to 21 do
      A[J, K] := 0;
  { -- Draw Circle }
  I := -PI / 2.0;
  while I < 3 / 2 * PI do begin
    X := Trunc(Cos(I) * 10);  Y := Trunc(Sin(I) * 10);
    GotoXY (11 + X, 11 + Y);  Write ('*');
    A[11 + X, 11 + Y] := 1;  I := I + 0.1;
  end;
  { -- Draw 3 line segments from center }
  Sum := 0;
  for S := 0 to 2 do begin
    Sum := Sum + P[S];
    I := -PI / 2 + 2 * PI * Sum / 100.0;
    for R := 0 to 10 do begin
      X := Trunc(Cos(I) * R);  Y := Trunc(Sin(I) * R);
      GotoXY (11 + X, 11 + Y);  Write ('*');
      A[11 + X, 11 + Y] := 1;
    end;
  end;
  Ch := ReadKey;  Sum := 0;
  { -- fill regions with letters }
  for S := 1 to 3 do begin
    LSum := Sum;  Sum := Sum + P[S];  J := LSum;
    while J < Sum do begin
      I := -PI / 2 + 2 * PI * J / 100.0;
      for R := 1 to 9 do begin
        X := Trunc(Cos(I) * R);  Y := Trunc(Sin(I) * R);
        if A[11 + X, 11 + Y] = 0 then begin
          GotoXY (11 + X, 11 + Y);  Write (L[S]);
        end;
      end;
      Inc(J);
    end;
  end;
end.
```

```
{3.9}
program Thr9T96;
{ -- This program produces a precedence of jobs to run. }
  var
    Num, I, J, K, L, DepLeft, UNum, P, St: Byte;
    Job:                String[3];
    Dep:                String[6];
    U, U2, Jobs, NewU2: String[24];
    A, B:               Array[1..8] of String[3];
    Marked:             Array[1..8] of Boolean;
    NoJob, ValidJob:    Boolean;


begin
  Write ('Enter number of dependencies: ');  Readln (Num);
  U := '';
  for I := 1 to Num do begin
    Write ('Enter dependency: ');  Readln (Dep);
    Dep := Dep + ' ';
    A[I] := Copy(Dep, 1, 3);
    B[I] := Copy(Dep, 4, 3);
    { -- Store unique jobs in string }
    if Pos(A[I], U) = 0 then U := U + A[I];
    if Pos(B[I], U) = 0 then U := U + B[I];
  end;
 { -- Since there is a unique order for all the jobs,
   -- every job will have its successor somewhere in B[].
   -- 1) search all B[] for the only job missing.
   -- 2) exclude all dependencies with this job in it.
   -- 3) search all B[] for the next only job missing.
   -- 4) repeat steps 2 and 3 until the final dependency is left.}
  L := Length(U);  UNum := L div 3;  U2 := U;
  DepLeft := Num;  Jobs := '';
  while DepLeft > 1 do begin
    for I := 1 to Num do  Marked[I] := False;
    for I := 1 to Num do begin
      P := Pos(B[I], U2);
      if P > 0 then Marked[ (P+2) div 3 ] := True;
    end;
    NoJob := True;  I := 0;
    while NoJob and (I < UNum) do begin
      Inc(I);  St := I * 3 - 2;
      Job := Copy(U2, St, 3);
      ValidJob := (Pos(Job, Jobs) = 0) and (Job <> '   ');
      if ValidJob and not Marked[I] then begin
        Jobs := Jobs + Job;
        for K := 1 to Num do
          if A[K] = Job then begin
            A[K] := '*';  B[K] := '*';
            Dec(DepLeft);
          end;
        NewU2 := Copy(U2, 1, St-1) + '   ';
        U2 := NewU2 + Copy(U2, St + 3, L - St - 2);
        NoJob := False;
      end;
```

```
    end;  { -- while }
  end;  { -- while }
  { -- Last dependency is concatenated }
  for I := 1 to Num do
    if A[I] <> '*' then Jobs := Jobs + A[I] + B[I];
  Writeln ('JOBS MUST BE RUN IN THIS ORDER: ', Jobs);
end.
```

```
{3.10}
program Thr10T96;
{ -- This program finds a perfect square with digits 1-9. }
  var
    A, N, Num, Min, NumMin, NumMin2: LongInt;
    I, B, Z, L, Code: Integer;
    Digits:           String[9];
    Good:             Boolean;
    Count:            Byte;

procedure CheckDigits;
{ -- Determine number of swaps made and store in count }
var
  D:          Array[1..9] of Byte;
  I, J, Temp: Byte;

begin
  for I := 1 to 9 do  Val(Digits[I], D[I], Code);
  Count := 0;
  for I := 1 to 9 do
    if D[I] <> I then begin
      J := I + 1;
      While (J < 9) and (D[J] <> I) do  Inc(J);
      Temp := D[I];  D[I] := D[J];  D[J] := Temp;
      Inc(Count);
    end;
end;

{ -- Main program }
begin
  Min := 9;
  for Num := 10001 to Trunc(Sqrt(987654321)) do begin
    A := Num * Num;
    Str(A, Digits);
    Good := True;  L := 1;
    while (L <= 9) and Good do begin
      if Pos(Chr(48+L), Digits) = 0 then Good := False;
      Inc(L);
    end;
    if Good then begin  {-- Found perfect square w/unique digits}
      CheckDigits;
      if Count < Min then begin
        Min := Count;  NumMin := A;  NumMin2 := Num;
      end;
    end;
  end;
  { -- Display the perfect square needing least num of swaps. }
  Str(NumMin, Digits);
  Writeln (Digits, ' IS THE SQUARE OF ', NumMin2);
  Write   ('AND WAS FORMED BY EXCHANGING ', Min);
  Writeln (' PAIRS OF DIGITS');
end.
```