```
{ -- FLORIDA HIGH SCHOOLS COMPUTING COMPETITION '88 }
{ -- PASCAL PROGRAM SOLUTIONS }


{1.1}
program One1T88;
{ -- This program clears the screen and prints a phrase 10 times.}
uses Crt;
  var
    I: Byte;

begin
  ClrScr;
  for I := 1 to 10 do
    Writeln ('THE BEST COMPUTER CONTEST!');
end.


{1.2}
program One2T88;
{ -- This program determines if a given input is integer or real.}
  var
    Num: Real;

begin
  Write ('Enter #: ');  Readln (Num);
  if Trunc(Num) - Num = 0 then
    Writeln ('INTEGER')
  else
    Writeln ('REAL');
end.


{1.3}
program One3T88;
{ -- This program calculates the number of bytes on N diskettes. }
  var
    N, Bytes: LongInt;

begin
  Write ('Enter N: ');  Readln (N);
  Bytes := N * 40 * 8 * 512;
  Writeln (Bytes);
end.
```

```
{1.4}
program One4T88;
{ -- This program prints the computer component missing. }
  const
    Comp: Array[1..5] of String[9] =
          ('CPU', 'PRIMARY', 'SECONDARY', 'INPUT', 'OUTPUT');
  var
    A:          String[9];
    I, J, Sum: Byte;

begin
  Sum := 0;
  for I := 1 to 4 do begin
    Write ('Enter component: ');  Readln (A);
    for J := 1 to 5 do
      if A = Comp[J] then Sum := Sum + J
  end;
  { -- The missing index = (1+2+3+4+5) - Sum }
  Writeln (Comp[15 - Sum]);
end.


{1.5}
program One5T88;
{ -- This program displays 4 rectangles of asterisks with #s. }
uses Crt;
  var
    I: Byte;

begin
  ClrScr;
  for I := 1 to 79 do
    Write ('*');

  for I := 2 to 24 do begin
    GotoXY (1,I);  Write ('*');
    GotoXY (40,I); Write ('*');
    GotoXY (79,I); Write ('*');
  end;

  for I := 1 to 79 do begin
    GotoXY (I,12); Write ('*');
  end;

  for I := 1 to 79 do begin
    GotoXY (I,24); Write ('*');
  end;

  GotoXY (20,6);  Write (1);
  GotoXY (60,6);  Write (2);
  GotoXY (20,18); Write (3);
  GotoXY (60,18); Write (4);
end.
```

```
{1.6}
program One6T88;
{ -- This program displays the acronym for a given set of words. }
  var
    I:  Byte;
    St: String[80];

begin
  Write ('Enter words: ');  Readln (St);
  Write (Copy(St, 1, 1));

  for I := 2 to Length(St) do begin
    if Copy(St, I, 1) = ' ' then
      Write (Copy(St, I+1, 1));
  end;
end.


{1.7}
program One7T88;
{ -- This program will display 3 computer names in order of size.}
  var
    N1, N2, N3, T1, T2, T3: String[10];

begin
  Write ('Enter name: ');  Readln (N1);
  Write ('Enter type: ');  Readln (T1);
  Write ('Enter name: ');  Readln (N2);
  Write ('Enter type: ');  Readln (T2);
  Write ('Enter name: ');  Readln (N3);
  Write ('Enter type: ');  Readln (T3);
  Writeln;

  if T1 = 'MICRO' then
    Writeln (N1)
  else if T2 = 'MICRO' then
    Writeln (N2)
  else
    Writeln (N3);

  if T1 = 'MINI' then
    Writeln (N1)
  else if T2 = 'MINI' then
    Writeln (N2)
  else
    Writeln (N3);

  if T1 = 'MAINFRAME' then
    Writeln (N1)
  else if T2 = 'MAINFRAME' then
    Writeln (N2)
  else
    Writeln (N3);
end.
```

```
{1.8}
program One8T88;
{ -- This program will count the number of cans to be stacked. }
  var
    N, Cans, Sum: Integer;

begin
  Write ('Enter N: ');  Readln (N);
  Cans := N;  Sum := 0;
  while (Cans > 0) do begin
    Sum := Sum + Cans;
    Cans := Cans - 2;
  end;
  Writeln (Sum);
end.


{1.9}
program One9T88;
{ -- This program simulates a queue w/options: ADD, TAKE, QUIT. }
  var
    Min, Max: Integer;
    Command:  String[4];
    A:        Array [1..10] of Integer;

begin
  Min := 0;
  Max := 0;
  repeat
    Write ('Enter command: ');  Readln (Command);
    if Command = 'ADD' then
      begin
        Inc(Max);
        Write ('Enter integer: ');  Readln (A[Max]);
      end
    else if Command = 'TAKE' then
      begin
        Inc(Min);
        Writeln (A[Min]);
      end
  until Command = 'QUIT';
end.
```

```
{1.10}
program One10T88;
{ -- This program determines events of history between dates. }
  type
    Ar = Array [1..7] of String[30];
  const
    Date: Array [1..7] of Integer =
              (1642, 1801, 1830, 1890, 1944, 1946, 1949);
    Per: Ar = ('BLAISE PASCAL', 'JOSEPH JACQUARD',
               'CHARLES BABBAGE', 'HERMAN HOLLERITH',
              'HOWARD AIKEN', 'ECKERT AND MAUCHLY', 'VON NEUMAN');
    Inv: Ar = ('ADDING MACHINE', 'PUNCHCARD AND WEAVING LOOM',
               'DESIGN OF ANALYTIC ENGINE',
               'PUNCHCARD TABULATING MACHINE', 'MARK I',
               'ENIAC', 'EDVAC');
  var
    Y1, Y2, I: Integer;

begin
  Write ('Enter years: ');  Readln (Y1, Y2);
  for I := 1 to 7 do begin
    if (Date[I] >= Y1) and (Date[I] <= Y2) then
      Writeln (Per[I], ' INVENTED ', Inv[I]);
  end;
end.
```

```
{2.1}
program Two1T88;
{ -- This program displays a solid diamond of asterisks. }
uses Crt;
  var
    I, J, N, NumOfSpaces: Integer;

begin
  Write ('Enter N: ');  Readln (N);

  { -- Display top half of diamond. }
  I := 1;
  repeat
    NumOfSpaces := (N - I) div 2 + 1;
    Write (' ': NumOfSpaces);
    for J := 1 to I do
      Write ('*');
    Writeln;
    I := I + 2;
  until I = N;
  I := I + 2;

  { -- Display middle row and bottom half of diamond. }
  repeat
    I := I - 2;
    NumOfSpaces := (N - I) div 2 + 1;
    Write (' ': NumOfSpaces);
    for J := 1 to I do
      Write ('*');
    Writeln;
  until I = 1;
end.
```

```
{2.2}
program Two2T88;
{ -- This program determines the efficiency order of 3 sorts. }
  const
    BS = 'BUBBLE SORT';
    SS = 'SHELL SORT';
    QS = 'QUICK SORT';
  var
    N:       Integer;
    B, S, Q: Real;

begin
  Write ('Enter N: ');  Readln (N);
  B := N * (N - 1) / 2;
  S :=     (Ln(N) / Ln(2));  S := N * S * S;
  Q := N * (Ln(N) / Ln(2));

  if (B < S) and (B < Q) then
    begin
      Writeln (BS);
      if S < Q then
        begin
          Writeln (SS);  Writeln (QS);
        end
      else
        begin
          Writeln (QS);  Writeln (SS);
        end
    end
  else if (S < B) and (S < Q) then
    begin
      Writeln (SS);
      if B < Q then
        begin
          Writeln (BS);  Writeln (QS);
        end
      else
        begin
          Writeln (QS);  Writeln (BS);
        end
    end
  else  { -- Q is less than both S and B }
    begin
      Writeln (QS);
      if B < S then
        begin
          Writeln (BS);  Writeln (SS);
        end
      else
        begin
          Writeln (SS);  Writeln (BS);
        end
    end
end.
```

```
{2.3}
program Two3T88;
{ -- This program determines the number of people in a group. }
  type
    Ar = Array [1..4] of Byte;
  const
    Di: Ar = (2, 3, 5, 7);
    Re: Ar = (1, 2, 1, 2);
  var
    Num, I: Byte;
    Found:  Boolean;

begin
  Num := 1;
  repeat
    Inc(Num);
    Found := True;
    for I := 1 to 4 do
      if (Num mod Di[I]) <> Re[I] then
        Found := False;
  until Found or (Num > 200);
  Writeln (Num);
end.


{2.4}
program Two4T88;
{ -- This program generates 5 random numbers between 0 and 9999. }
  const
    EightDigits = 10E7;
  var
    I, J:       Byte;
    Seed, Prod: LongInt;
    St, SeedSt: String[8];
    Code:       Integer;

begin
  Write ('Enter seed: ');  Readln (Seed);
  for I := 1 to 5 do begin
    Prod := Seed * Seed;
    while (Prod < EightDigits) and (Prod <> 0) do
      Prod := Prod * 10;
    Str (Prod, St);
    SeedSt := Copy (St, 3, 4);
    Val (SeedSt, Seed, Code);
    Writeln (Seed);
  end;
end.
```

```
{2.5}
program Two5T88;
{ -- This program checks to see if data transmitted is Correct. }
  var
    Bit, Par: String[8];
    I, One:   Byte;
    Error:    Boolean;

begin
  Write ('Enter bits: ');  Readln (Bit);
  Write ('Enter parity: ');  Readln (Par);
  if Length(Bit) < 8 then
    Writeln ('ERROR')
  else
    begin
      Error := False;
      One := 0;
      for I := 1 to 8 do begin
        If not (Bit[I] in ['0','1']) then
          Error := True;
        If Bit[I] = '1' then
          Inc(One);
      end;  { -- for }

      if (One mod 2 = 0) and (Par <> 'EVEN') then
        Error := True
      else if ((One mod 2) <> 0) and (Par <> 'ODD') then
        Error := True;
      if Error then
        Writeln ('ERROR')
      else
        WriteLn ('CORRECT');
    end;  { -- else }
end.
```

```
{2.6}
program Two6T88;
{ -- This program will calculate the area of a polygon. }
  var
    I, N: Byte;
    X, Y: Array [1..10] of Integer;
    Sum:  Integer;

begin
  Write ('Enter n: ');  Readln (N);
  for I := 1 to N do begin
    Write ('Enter vertex: ');
    Readln (X[I], Y[I]);
  end;

  Sum    := 0;
  X[N+1] := X[1];
  Y[N+1] := Y[1];
  for I := 1 to N do
    Sum := Sum + X[I] * Y[I+1] - Y[I] * X[I+1];
  Writeln ('AREA = ',Abs(Sum) / 2 : 4:1);
end.
```

```
{2.7}
program Two7T88;
{ -- This program displays the date before/after a given date. }
  const
     Mo: Array [1..12] of Byte =
         (31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31);
  var
    Month, Day, D1, D2, M1, M2, Leap1, Leap2: Byte;
    Year, Y1, Y2:                             Integer;

begin
  Write ('Enter month, day, year: ');  Readln (Month, Day, Year);

  D1 := Day - 1;  D2 := Day + 1;
  M1 := Month;    M2 := Month;
  Y1 := Year;     Y2 := Year;
  Leap1 := 0;  Leap2 := 0;
  if (Y1 mod 4 = 0) and (Y1 mod 100 <> 0) then
    if (M1 = 3) and (D1 = 0) then
      Leap1 := 1
    else
      if (M2 = 2) and (D2 = 29) then
        Leap2 := 1;

  if D1 <= 0 then begin
    Dec(M1);
    if M1 > 0 then
      D1 := Mo[M1] + Leap1
    else
      begin
        M1 := 12;  D1 := Mo[M1];  Dec(Y1);
      end;
  end;  { -- If then }
  if D2 > (Mo[M2] + Leap2) then begin
    Inc(M2);  D2 := 1;
    if M2 > 12 then begin
      M2 := 1;  Inc(Y2);
    end;
  end;  { -- if then }

  Writeln (M1, '-', D1, '-', Y1);
  Writeln (M2, '-', D2, '-', Y2);
end.
```

```
{2.8}
program Two8T88;
{ -- This program displays a student's Cumulative G. P. Ave. }
  var
    Sem, Total, HrsTot:  Byte;
    Gr:                  Char;
    Hrs, Poynts, I:      Byte;
    CumTotal, CumHrs:    Byte;
    GPA, CGPA, LastCGPA: Real;
    Dismissed:           Boolean;

begin
  Sem := 1;  Dismissed := False;  LastCGPA := 0;
  CumHrs :=0; CumTotal :=0;
  while (Sem <= 8) and not Dismissed do begin
    Total := 0;  HrsTot := 0;
    for I := 1 to 4 do begin
      Write ('Enter grade, credits: ');
      Readln (Gr, Hrs);
      if Gr = 'F' then Gr := 'E';
      Poynts := 4 - (Ord(Gr) - 65);  { -- A=4,B=3,C=2,D=1,F=0 }
      Total  := Total  + Poynts * Hrs;
      HrsTot := HrsTot + Hrs;
    end;  { -- for }

    GPA := Total / HrsTot;
    GPA := Int (GPA * 1000 + 0.5) / 1000;
    Writeln (' GPA= ', GPA: 5: 3);
    CumTotal := CumTotal + Total;
    CumHrs   := CumHrs   + HrsTot;
    CGPA := CumTotal / CumHrs;
    CGPA := Int (CGPA * 1000 + 0.5) /1000;
    Writeln ('CGPA= ', CGPA: 5: 3);
    if CGPA < 1 then
      Dismissed := True;
    if (CGPA < 2) and (LastCGPA < 2) and (Sem > 1) then
      Dismissed := True;
    LastCGPA := CGPA;
    Inc(Sem);
  end;  { -- while }
  If Dismissed then
    Writeln ('STUDENT IS DISMISSED');
end.
```

```
{2.9}
program Two9T88;
{ -- This program displays 2 elements that form a battery. }
uses Crt;
  const
    Elem: Array [1..10] of String[8] =
     ('LITHIUM ', 'SODIUM  ', 'ZINC    ', 'IRON    ', 'TIN     ',
      'IODINE  ', 'SILVER  ', 'MERCURY ', 'BROMINE ', 'CHLORINE');
    Pot:  Array [1..10] of Real =
     ( +3.05, +2.71, +0.76, +0.44, +0.14,
       -0.54, -0.80, -0.85, -1.09, -1.36);

  var
    I, J, Count:    Byte;
    Dif, Volt, Tol: Real;
    Displayed:      Boolean;
    Ch:             String[1];

begin

  Write ('Enter Desired Voltage, Tolerance: ');
  Readln (Volt, Tol);

  Displayed := False;  Count := 0;
  for I := 1 to 10 do
    for J := 1 to 10 do begin
      Dif := Pot[I] - Pot[J];
      If (Dif >= Volt - Tol) and (Dif <= Volt + Tol) then begin
        Inc(Count);
        if (Count = 1) and Displayed then begin
          Writeln ('PRESS ANY KEY FOR MORE');
          Ch:= '';  While Ch = '' do Ch := ReadKey;
          Writeln;
        end;
        Writeln (Elem[I], '   ', Elem[J], '   ', Dif: 3: 2);
        Displayed := True;
      end;  { -- if Dif }
      if Count = 8 then begin
        Writeln;
        Count := 0;
      end;
    end;  { -- for J }
  if not Displayed then
    Writeln ('NO BATTERY CAN BE FORMED');
end.
```

```
{2.10}
program Two1088;
{ -- This program will keep score for a double dual race. }
uses Crt;
  var
    Init:                Array [1..21] of Char;
    TeamName:            Array [1..3] of Char;
    I, J, K:             Byte;
    StillUnique:         Boolean;
    UniqueTeams, Pl:     Byte;
    Team1Pos, Team2Pos: Array [1..7] of Byte;
    Team1,    Team2:     Byte;
    Team1Pl,  Team2Pl:   Byte;

begin
  ClrScr;  UniqueTeams := 0;
  for I := 1 to 21 do begin
    Write ('Place ', I: 2, ': ');  Readln (Init[I]);
    J := 0;  StillUnique := True;
    while (J < UniqueTeams) and StillUnique and (I > 1) do begin
      Inc(J);
      if TeamName[J] = Init[I] then
        StillUnique := False;
    end; { -- while }
    if StillUnique then
      begin
        Inc(UniqueTeams);
        TeamName[UniqueTeams] := Init[I];
      end;
  end; { -- for I }
  { -- Assert that Team[1,2,3] = 3 unique team Initials. }

  for I := 1 to 2 do
    for J := I+1 to 3 do begin
      PL := 0;  Team1 := 0;  Team2 := 0;
      Team1Pl := 0;  Team2Pl :=0;
      for K := 1 to 21 do begin
        if Init[K] = TeamName[I] then
        begin
          Inc(Pl);
          Team1 := Team1 + Pl;
          Inc(Team1Pl);
          Team1Pos[Team1Pl] := Pl
        end;
        if Init[K] = TeamName[J] then
        begin
          Inc(Pl);
          Team2 := Team2 + Pl;
          Inc(Team2Pl);
          Team2Pos[Team2Pl] := Pl
        end;
      end;  { -- for K }
      Team1 := Team1 - Team1Pos[6] - Team1Pos[7];
      Team2 := Team2 - Team2Pos[6] - Team2Pos[7];
      Writeln ('TEAM ', TeamName[I], ': ', Team1, ' POINTS');
```

```
      Writeln ('TEAM ', TeamName[J], ': ', Team2, ' POINTS');
      if (Team1 < Team2)
      or ((Team1 = Team2) and (Team1Pos[6] < Team2Pos[6])) then
        Write ('TEAM ', TeamName[I])
      else
        Write ('TEAM ', TeamName[J]);
      Writeln (' WINS!');  Writeln;
    end;  { -- for J }
end.
```

```
{3.1}
program Thr1T88;
{ -- This program puts a set of real numbers in numerical order. }
  const
    Order:    Array [0..9] of Byte = (0,8,1,2,5,4,3,9,7,6);
  var
    I, J, N:  Byte;
    A:        Array [1..10] of String[18];
    B:        Array [1..10] of Real;
    Temp:     Real;
    TempSt,
    Num:      String[18];
    NumVal,
    NumVal2:  Integer;
    Md:       Char;
    NumValSt: String[1];
    Result:   Integer;

begin
  Write ('Enter N: ');  Readln (N);
  for I := 1 to N do begin
    Write ('Enter #: ');  Readln (A[I]);
  end;

  { -- Replace digits in duplicated number }
  for I := 1 to N do begin
    Num := A[I];
    for J := 1 to Length(Num) do begin
      Md := Num[J];
      NumVal := Ord(Md) - Ord('0');
      if (NumVal > 0) or (Md = '0') then begin
        NumVal2 := Order[NumVal];
        Delete (Num, J, 1);
        Str (NumVal2, NumValSt);
        Insert (NumValSt, Num, J);
      end;
    end;  { -- for J }
    Val (Num, B[I], Result);
  end;  { -- for I }

{ -- Sort according to numbers with replaced digits }
  for I := 1 to N - 1 do
    for J := I + 1 to N do
      if B[I] > B[J] then begin
        Temp   := B[I];  B[I] := B[J];  B[J] := Temp;
        TempSt := A[I];  A[I] := A[J];  A[J] := TempSt;
      end;

  for I := 1 to N do
    Writeln (A[I]);
end.
```

```
{3.2}
program Thr2T88;
{ -- This program displays total number of ways to make change. }
  var
    Amount:           Real;
    MaxQ, MaxD, MaxN: Integer;
    Q, D, N, Count:   Integer;

begin
  Write ('Enter AMOUNT: ');  Readln (Amount);
  MaxQ := Trunc(Amount * 4);
  MaxD := Trunc(Amount * 10);
  MaxN := Trunc(Amount * 20);
  Count := 0;
  for Q := 0 to MaxQ do
    for D := 0 to MaxD - Trunc(2.5 * Q) do
      for N := 0 to MaxN - 5*Q - 2*D do
        Inc(Count);
  Writeln (Count);
end.
```

```
{3.3}
program Thr3T88;
{ -- This program determines if a point/box is inside a 2nd box. }

function Min (A: Real;  B: Real): Real;
begin
  if A < B then
    Min := A
  else
    Min := B;
end;

function Max (A: Real;  B: Real): Real;
begin
  if A > B then
    Max := A
  else
    Max := B;
end;

{ -- Start of Main Program }
  var
    PX, PY, PZ,
    C1X1, C1Y1, C1Z1, C1X2, C1Y2, C1Z2,
    C2X1, C2Y1, C2Z1, C2X2, C2Y2, C2Z2,
    C1MinX, C1MinY, C1MinZ, C1MaxX, C1MaxY, C1MaxZ,
    C2MinX, C2MinY, C2MinZ, C2MaxX, C2MaxY, C2MaxZ:  Real;
```

```
begin
  Write ('Enter point: ');  Readln (PX, PY, PZ);
  Write ('Enter cube1 diagonal point1: ');
    Readln (C1X1, C1Y1, C1Z1);
  Write ('Enter cube1 diagonal point2: ');
    Readln (C1X2, C1Y2, C1Z2);
  Write ('Enter cube2 diagonal point1: ');
    Readln (C2X1, C2Y1, C2Z1);
  Write ('Enter cube2 diagonal point2: ');
    Readln (C2X2, C2Y2, C2Z2);

  C1MinX := Min (C1X1, C1X2);
  C1MinY := Min (C1Y1, C1Y2);
  C1MinZ := Min (C1Z1, C1Z2);
  C2MinX := Min (C2X1, C2X2);
  C2MinY := Min (C2Y1, C2Y2);
  C2MinZ := Min (C2Z1, C2Z2);
  C1MaxX := Max (C1X1, C1X2);
  C1MaxY := Max (C1Y1, C1Y2);
  C1MaxZ := Max (C1Z1, C1Z2);
  C2MaxX := Max (C2X1, C2X2);
  C2MaxY := Max (C2Y1, C2Y2);
  C2MaxZ := Max (C2Z1, C2Z2);

  Write ('POINT ');
  If (PX < C2MinX) or (PY < C2MinY) or (PZ < C2MinZ)
  or (PX > C2MaxX) or (PY > C2MaxY) or (PZ > C2MaxZ) then
    Write ('DOES NOT LIE')
  else
    Write ('LIES');
  Writeln (' INSIDE 2ND CUBE');

  Write ('1ST CUBE ');
  If (C1MinX < C2MinX) or (C1MinY < C2MinY) or (C1MinZ < C2MinZ)
  or (C1MaxX > C2MaxX) or (C1MaxY > C2MaxY) or (C1MaxZ > C2MaxZ)
  then
    Write ('DOES NOT LIE')
  else
    Write ('LIES');
  Writeln (' INSIDE 2ND CUBE');
end.
```

```
{3.4}
program Thr4T88;
{ -- This program produces an alphabetical list of permutations. }
  type
    String6  = Array [1..6] of String[1];
    PermType = Array [1..720] of String[6];
  var
    Number, I: Integer;
    Letters:   String[6];
    S:         String6;
    Perm:      PermType;
    Total:     Integer;

procedure Permute ({Using}     N:     Integer;
                   {Giving} var S:     String6;
                            var Perm:  PermType;
                            var Total: Integer);
{ -- This procedure will interchange the elements in Array S. }
  const
    Empty = '';
  var
    Temp: String[1];
    I, J: Integer;

begin
  If N > 1 then
    begin
      Permute (N - 1, S, Perm, Total);
      for I := N - 1 downto 1 do begin
        {Interchange the elements in S[N] and S[I] }
        Temp := S[N];  S[N] := S[I];  S[I] := Temp;
        Permute (N - 1, S, Perm, Total);
        Temp := S[N];  S[N] := S[I];  S[I] := Temp;
      end;  { -- for I }
    end  { -- if then }
  else
    begin
      Inc(Total);
      Perm[Total] := Empty;
      for J := 1 to Number do
        Perm[Total] := Perm[Total] + S[J];
    end;
end;  {procedure}

procedure Alphabetize (var Perm: Permtype;  Total: Integer);
{ -- This procedure alphabetizes permutations w/insertion sort. }
  var
    I, Index: Integer;
    Temp:     String[6];

begin
  for I := 2 to Total do begin
    Index := I;
    while (Perm[Index] < Perm[Index-1]) and (Index > 1) do begin
      Temp := Perm[Index];
```

```
      Perm[Index] := Perm [Index-1];
      Perm[Index-1] := Temp;
      Dec(Index);
    end;
  end;
end;  { -- procedure }

procedure Display (var Perm: PermType;  Total: Integer);
{ -- This procedure displays the unique permutations in the list.}
  var
    Total2, I: Integer;

begin
  Writeln (Perm[1]);
  Total2 := 1;
  for I := 2 to Total do
    if Perm[I] <> Perm[I-1] then begin
      Writeln (Perm[I]);
      Inc(Total2);
    end;
  Writeln ('TOTAL= ', TOTAL2);
end; { -- procedure }

      { -- Main program }
begin
  Write ('Enter letters: ');  Readln (Letters);
  Number := Length(Letters);
  for I := 1 to Number do
    S[I] := Copy(Letters, I, 1);
  Total := 0;
  Permute (Number, S, Perm, Total);
  Alphabetize (Perm, Total);
  Display (Perm, Total);
end.
```

```
{3.5}
program Thr5T88;
{ -- This program will control the movements of a snake. }
uses Crt;
  const
    SnakeLen = 25;
  var
    V, H, X, Y:     Byte;
    I:              Integer;
    VCoord, HCoord: Array [1..SnakeLen] of Byte;
    FrontHV, EndHV: Byte;
    Ch:             Char;
    InvalidKey:     Boolean;
```

```
begin
  ClrScr;
  InvalidKey := False;
  V := 12;  H := 40 - (SnakeLen div 2);  GotoXY (H,V);
  FrontHV := 0;   EndHV := 1;
  { -- Center snake (asterisks) on the screen }
  for I := H to (H + SnakeLen - 1) do begin
    Write ('*');
    Inc(FrontHV);
    VCoord[FrontHV] := V;
    HCoord[FrontHV] := I;
  end;
  repeat until KeyPressed;
  Ch := ReadKey;

  repeat
    H := HCoord[FrontHV];
    V := VCoord[FrontHV];
    for I := 1 to 2000 do
      If KeyPressed then Ch := ReadKey;

    case Upcase(Ch) of
      'I': Dec(V);
      'M': Inc(V);
      'J': Dec(H);
      'K': Inc(H);
    end;

    for I := 1 to SnakeLen do
      if (H = HCoord[I]) and (V = VCoord[I]) then
        InValidKey := True;

    if InValidKey or (V = 0) or (V = 25) or (H = 0) or (H = 80)
      then InvalidKey := True
    else begin
      GotoXY (H,V);  Write ('*');
      Y := HCoord[EndHV];
      X := VCoord[EndHV];
      GotoXY (Y,X);  Write (' ');
      HCoord[EndHV] := H;
      VCoord[EndHV] := V;
      Inc(FrontHV);
      if FrontHV > SnakeLen then
        FrontHV := 1;
      Inc(EndHV);
      If EndHV > SnakeLen then
        EndHV := 1;
    end; { -- else }
  until InvalidKey;
end.
```

```
{3.6}
program Thr6T88;
{ -- This program will solve two linear equations. }
  type
    String15 = String[15];
  var
    E1, E2, Eq:  String15;
    A1, B1, C1,
    A2, B2, C2:  Integer;
    St, Den,
    NumX, NumY:  Integer;

function Vaal ({Using} Eq: String15;
                    var St: Integer): {Giving} Integer;
{ -- This function determines the coefficient for a term. }
  var
    Md:       String[5];
    En, Sygn: Integer;
    Result:   Integer;
    Coef:     Integer;
begin
  { -- Find Starting position ST of value }
  Sygn := 1;   { -- Default to 1 for positive unsigned #s }
  Md := Copy(Eq, St, 1);
  if Md = '=' then begin
    Inc(St);
    Md := Copy(Eq, St, 1);
  end;
  if Md = 'X' then
    begin
      Vaal := 1;  Inc(St);  Exit;
    end
  else if Md = '+' then
    Inc(St)
  else if Md = '-' then
    begin
      Sygn := -1;  Inc(St);
    end;

  { -- Find ending position EN of value }
  En := St;  Vaal := 0;  Md := Copy(Eq, En, 1);
  while (En <= Length(Eq)) and
        (Md <> 'X') and (Md <> 'Y') and (Md <> '=') do
  begin
    Md := Copy(Eq, En, 1);
    Inc(En);
  end;
  Dec(En);
  if (Md = 'X') or (Md = 'Y') or (Md = '=') then
    Dec(En);
  if Md = '=' then
    Sygn := -Sygn;
  if St > En then begin
    Vaal := Sygn;  Inc(St);  Exit;
  end;
```

```
  { -- Determine value }
  Md   := Copy (Eq, St, En - St + 1);
  Val (Md, Coef, St);
  Vaal := Sygn * Coef;
  St   := En + 2;
end;  { -- function }

{ -- Main routine }
begin
  Write ('Enter equation 1: ');  Readln (E1);
  Write ('Enter equation 2: ');  Readln (E2);
  St := 1;
  A1 := Vaal(E1, St);
  B1 := Vaal(E1, St);
  C1 := Vaal(E1, St);
  St := 1;
  A2 := Vaal(E2, St);
  B2 := Vaal(E2, St);
  C2 := Vaal(E2, St);

  Den  := A1*B2 - A2*B1;
  NumX := C1*B2 - C2*B1;
  NumY := A1*C2 - A2*C1;
  if Den = 0 then
    Writeln ('NO UNIQUE SOLUTION EXISTS.')
  else begin
    Write   ('XSOLUTION= ', NumX / Den : 3:1);
    Writeln ('   YSOLUTION= ', NumY / Den : 3:1);
  end;
end.
```

```
{3.7}
program Thr7T88;
{ -- This program display all semi-perfect #s between 2 and 35. }
uses Crt;
  type
    ArrayType = Array [1..20] of Byte;
  var
    Factors:      ArrayType;
    Num, Di, Max: Byte;
    Combo:        Byte;

procedure PrintCombos ({Using} Factors:  ArrayType;
                                 Combo:  Byte;
                                   Len:  Byte;  Num: Byte);
{ -- This procedure displays Combo character combinations of Len }
  var
    A:            ArrayType;
    N, I, Q, Sum: Byte;
```

```
begin
  for I := 1 to Combo do
    A[I] := Combo - I + 1;
  Dec(A[1]);  N := 1;

  while N <= Combo do begin
    Inc(A[N]);
    if A[N] <= Len - N + 1 then begin
      for I := N - 1 downto 0 do
        A[I] := A[I+1] + 1;

      { -- One combination produced, Now Check for Semi-perfect }
      Sum := 0;
      for I := 1 to Combo do
        Sum := Sum + Factors[ A[I] ];
      if Sum = Num then begin
        Write (Num:2, '      ', Factors[ A[Combo] ]);
        for I := Combo - 1 downto 1 do
          Write (' + ', Factors[ A[I] ]);
        Writeln;
      end;      { -- if Sum }
      N := 0;  { -- Keep N at value 1 }
    end;        { -- if A[N] }
    Inc(N);

  end;  { -- while }
end;  { -- procedure }

begin
  ClrScr;
  Writeln ('SEMI #  EXAMPLE(S)');
  for Num := 2 to 34 do begin
    Max :=0;
    for Di := 1 to (Num Div 2) do
      if (Num mod Di) = 0 then begin
        Inc(Max);
        Factors[Max] := Di;
      end;  { -- If }
    for Combo := 2 to Max do
      PrintCombos (Factors, Combo, Max, Num);
  end;  { -- for Num }
end.
```

```
{3.8}
program Thr8T88;
{ -- This program will keep score for a bowler. }
uses Crt;
  var
    I, J, Fr,
    CommaPos, Len: Byte;
    A:             Array [1..10] of String[3];
    Frames:        String [40];
    Md:            Char;
    Look, Sum:     Array [0..10] of Integer;
    AA:            Array [1..10,1..3] of Integer;

begin
  ClrScr;
  Write ('Enter frames: ');  Readln (Frames);
  Frames := Frames + ',';
  for I := 1 to 10 do begin
    CommaPos := Pos (',', Frames);
    A[I] := Copy(Frames, 1, CommaPos - 1);
    Frames := Copy(Frames,CommaPos + 1,Length(Frames) - CommaPos);
  end;

  Writeln;
  Writeln ('-1- -2- -3- -4- -5- -6- -7- -8- -9- -10-');
  Writeln ('---!---!---!---!---!---!---!---!---!---!');
  for I := 1 to 10 do
    Write (A[I]: 3, '!');
  Writeln;

  { -- Assign values to A FRames according to X, /, or pins }
  for Fr := 1 to 10 do begin
    AA[Fr,2] := 0;
    for J := 1 to Length(A[Fr]) do begin
      Md := A[Fr,J];
      if Md = 'X' then
        begin
          AA[Fr,J] := 10;  Look[Fr] := 2;
        end
      else if Md = '/' then
        begin
          AA[Fr,J] := 10 - AA[Fr,J-1];  Look[Fr] := 1;
        end
      else
        if Md = '-' then
          AA[Fr,J] := 0
        else begin
          AA[Fr,J] := Ord(Md) - Ord('0');  Look[Fr] := 0;
        end;
    end;  { -- for J }
  end;  { -- for Fr }

  { -- Assign Frame values with Look ahead }
  Sum[0] := 0;
  for Fr := 1 to 10 do begin
```

```
    Sum[Fr] := Sum[Fr-1] + AA[Fr,1] + AA[Fr,2];
    if Look[Fr] > 0 then
      if Look[Fr] = 1 then  { -- A spare / needs 1 more value }
        if Fr = 10 then
          Sum[Fr] := Sum[Fr] + AA[Fr,3]
        else
          Sum[Fr] := Sum[Fr] + AA[Fr+1,1]
      else   { -- A strike X needs 2 more values }
        if Fr = 10 then
          Sum[Fr] := Sum[Fr] + AA[Fr,3]
        else
          begin
            Sum[Fr] := Sum[Fr] + AA[Fr+1,1] + AA[Fr+1,2];
            if Fr < 9 then
              if AA[Fr+1,1] = 10 then
                Sum[Fr] := Sum[Fr] + AA[Fr+2,1];
          end;

    Len := Trunc (Ln(Sum[Fr]) / Ln(10)) + 1;
    Write (Sum[Fr]: Len, '': 3 - Len, '!');
  end;  { -- for Fr }
  Writeln;
  for I := 1 to 40 do Write ('-');
  Writeln;
end.
```

```
{3.9}
program Thr9T88;
{ -- This program will convert a real from one base to another. }
  const
    Digits = '0123456789ABCDEF';
  var
    M, N, I, J, MdVal: Byte;
    Num:               String[10];
    MDigits, NDigits:  Byte;
    Md:                Char;
    Sum:               Real;
    NumArray:          Array [0..8] of Byte;

function Power({Using} Base: Real;
                  Exponent: Byte): {Giving} Real;
{ -- This function returns Base^Exponent. }
  var
    I: Integer;
    P: Real;
begin
  P := 1;
  for I := 1 to Exponent do
    P := P * Base;
  Power := P;
end;
```

```
begin
  Write ('Enter M, N, #: ');
  Readln (M, N, Num);
  Write (Copy(Num, 2, 2));
  MDigits := Length(Num) - 3;
  Num := Copy(Num, 4, MDigits);

  NDigits := 1;
  while (Power((1/N),NDigits) > Power((1/M),MDigits))
  and (NDigits < 7) do
    Inc(NDigits);

  { -- SUM = Base 10 # of Num }
  Sum := 0;
  for I := 1 to MDigits do begin
    Md := Num[I];
    MdVal := Ord(Md) - Ord('0');
    if MdVal > 9 then
      MdVal := MdVal - 7;
    Sum := Sum + MdVal / Power(M,I);
  end;

  { -- Convert base 10 decimal to Base N fraction }
  for I := 1 to NDigits + 1 do begin
    Sum := Sum * N;
    NumArray[I] := Trunc(Sum);
    Sum := Sum - NumArray[I];
  end;

  { -- Print fraction with last digit rounded at NDigits + 1 }
  for I := 1 to NDigits - 1 do
    Write (Copy(Digits, NumArray[I] + 1, 1));
  if NumArray[NDigits+1] >= (N / 2) then
    Inc( NumArray[NDigits] );
  Writeln (Copy(Digits, NumArray[NDigits] + 1, 1));
end.
```

```
{3.10}
program Thr10T88;
{ -- This program computes the composition of P (Q) and Q (P) }
  type
    ArrayType = Array [0..5] of Integer;
  var
    POrder, QOrder, I: Integer;
    PCo, QCo:          ArrayType;

procedure ComputeComp ({Using} PCo, QCo: ArrayType;
                          POrder, QOrder: Integer);
{ -- Compute composition of P of Q }
  var
    ProdOrder, CompOrder: Integer;
    I, J, K, L, Ind:      Byte;
    PofQ, Prod, Prod2:    Array [0..25] of Integer;

begin
  CompOrder := POrder * QOrder;
  for I := 0 to CompOrder do
    PofQ[I] := 0;
  for I := 0 to POrder do
    if PCo[I] <> 0 then
      if I = 0 then
        PofQ[0] := PCo[0]
      else begin
        for J := 0 to QOrder do
          Prod[J] := QCo[J];
        ProdOrder := QOrder;
        If I > 1 then
          for Ind := 1 to I-1 do begin
            for J := 0 to ProdOrder + QOrder do
              Prod2[J] := 0;
            for J := 0 to ProdOrder do
              for K := 0 to QOrder do
                Prod2[J+K] := Prod2[J+K] + Prod[J]*QCo[K];
            ProdOrder := J + K;
            for L := 0 to ProdOrder do begin
              Prod[L] := Prod2[L];  Prod2[L] := 0;
            end;  { -- for L }
          end;  { -- for Ind }
          for J := 0 to ProdOrder do
            Prod[J] := Prod[J] * PCo[I];
          for J := ProdOrder downto 0 do
            PofQ[J] := PofQ[J] + Prod[J]
      end;  { -- else begin }

  { -- Print composition }
  for I := CompOrder downto 0 do begin
    if I < CompOrder then
      Write (' + ');
    Write (PofQ[I], 'X**', I);
  end;
  Writeln;
end;
```

```
begin
  Write ('Enter the ORDER of p(x): ');  Readln (POrder);
  for I := POrder downto 0 do begin
    Write ('Enter coefficient for x**',I,': ');  Readln (PCo[I]);
  end;
  Write ('Enter the ORDER of q(x): ');  Readln (QOrder);
  for I := QOrder downto 0 do begin
    Write ('Enter coefficient for x**',I,': ');  Readln (QCo[I]);
  end;

  Write ('P(Q(X))= ');
  ComputeComp (PCo, QCo, POrder, QOrder);
  Write ('Q(P(X))= ');
  ComputeComp (QCo, PCo, QOrder, POrder);
end.
```