```
{ -- FLORIDA HIGH SCHOOLS COMPUTING COMPETITION '89 }
{ -- PASCAL PROGRAM SOLUTIONS }


{1.1}
program One1T89;
{ -- This program will print an indented phrase on each line. }
uses Crt;
  const
    Phrase = '1989 COMPUTER CONTEST';
  var
    I: Byte;

begin
  ClrScr;
  for I := 1 to 22 do
    Writeln (' ':I, Phrase);
end.


{1.2}
program One2T89;
{ -- This program will translate gigabytes to megabytes. }
  var
    G: Integer;

begin
  Write ('Enter number of gigabytes: ');  { -- less than 30 }
  Readln (G);
  Writeln (G * 1024, ' MEGABYTES');
end.


{1.3}
program One3T89;
{ -- This program displays a word in a backward-L format. }
  var
    Word:   String[15];
    I, Len: Byte;

begin
  Write ('Enter word: ');  Readln (Word);
  Len := Length(Word);
  for I := 1 to Len-1 do
    Writeln (' ':Len-1, Copy(Word, I, 1));
  Writeln (Word);
end.
```

```
{1.4}
program One4T89;
{ -- This program prints a pattern of numbers in pyramid form. }
  var
    N, I: Byte;

begin
  Write ('Enter N: ');  Readln (N);
  for I := 1 to N do begin
    Write (' ':10-I, I);
    if I > 1 then
      Write (' ':I*2-3, I);
    Writeln;
  end;
end.


{1.5}
program One5T89;
{ --- This program corrects dates with A.D. or B.C. }
  var
    Era:  String[4];
    Dayt: Integer;

begin
  Write ('Enter date: ');           Readln (Dayt);
  Write ('Enter A.D. or B.C.: ');  Readln (Era);
  if Era = 'A.D.' then
    Writeln (Dayt + 4, ' ', Era)
  else if Dayt > 4 then
    Writeln (Dayt - 4, ' ', Era)
  else
    Writeln (5 - Dayt, ' A.D.');
end.
```

```
{1.6}
program One6T89;
{ -- This program will allow a user access with a password. }
  const
    Pass = 'ITSME';
  var
    PassW: String[10];
    I:     Byte;

begin
  Write ('ENTER PASSWORD: ');  Readln (PassW);
  I := 0;
  while (PassW <> Pass) and (I < 2) do begin
    Writeln ('INVALID PASSWORD');
    Write ('ENTER PASSWORD: ');  Readln (PassW);
    Inc(I);
  end;
  if PassW = Pass then
    Writeln ('YOU HAVE ACCESS')
  else
    Writeln ('YOU ARE TRESPASSING');
end.


{1.7}
program One7T89;
{ -- This program will display the best DBMS. }
  var
    N, Max, I, C, E: Byte;
    Name, Best:      String[9];

begin
  Write ('Enter N: ');  Readln (N);  Max := 0;
  for I := 1 to N do begin
    Write ('Enter DBMS name: ');  Readln (Name);
    Write ('Enter convenience, efficiency: ');   Readln (C, E);
    if C + E > Max then begin
       Max := C + E;  Best := Name;
    end;
  end;
  Writeln (Best, ' IS BEST');
end.
```

```
{1.8}
program One8T89;
{ -- This program displays the unique elements of a list. }
  var
    N:      Integer;
    Num, I: Byte;
    List:   Array[1..10] of Integer;

begin
  Write ('Enter #: ');  Readln (N);
  Num := 0;
  while N <> -999 do begin
    I := 1;
    while (I <= Num) and (N <> List[I]) do Inc(I);
    if I > Num then begin
      Num := I;  List[Num] := N;
    end;
    Write ('Enter #: ');  Readln (N);
  end;
  for I := 1 to Num do Write (List[I], ' ');
end.


{1.9}
program One9T89;
{ -- This program determines how many feet deep of dollar coins
  -- over Texas is equivalent to a given probability. }
  const
    TexasArea = 262134.0;
  var
    DolVol, TexasVol, Prob, InchDeep: Real;

begin
  Write ('Enter probability: ');  Readln (Prob);
  DolVol := (1.5) * (1.5) * (3/32);  { -- Volume Dollar takes }
  TexasVol := TexasArea * (5280.0 * 12.0 * 5280.0 * 12.0);
  InchDeep := (Prob / (TexasVol / DolVol));
  Writeln (InchDeep / 12 :5:0, ' FEET DEEP');
end.
```

```
{1.10}
program One10T89;
{ -- This program will map a logical address to the physical. }
  const
    Base: Array[0..4] of Integer = (219, 2300, 90, 1327, 1952);
    Len:  Array[0..4] of Integer = (600, 14, 100, 580, 96);
  var
    Adr, Seg: Integer;

begin
  Write ('Enter Seg#, Address: ');  Readln (Seg, Adr);
  while Seg <= 4 do begin
    if Adr > Len[Seg] then
      Writeln ('ADDRESSING ERROR')
    else
      Writeln (Base[Seg] + Adr);
    Write ('Enter Seg#, Address: ');  Readln (Seg, Adr);
  end;
end.
```

```
{2.1}
program Two1T89;
{ -- This program prints F(x) for a recursive function given x. }
  var
    F:    Array [1..11] of Integer;
    I, X: Byte;

begin
  Write ('Enter x: ');  Readln (X);
  F[1] := 1;  F[2] := 1;  F[3] := 1;
  I := 3;
  while I < X do begin
    F[I+1] := (F[I] * F[I-1] + 2) div F[I-2];
    Inc(I);
  end;
  Writeln ('F(', X, ')=', F[X]);
end.


{2.2}
program Two2T89;
{ -- This program will print the prime factors of a number. }
  var
    I, Num: Integer;

begin
  Write ('Enter #: ');  Readln (Num);
  while Num > 1 do begin
    I := 2;
    while (Num mod I) > 0 do
      Inc(I);
    Write (I);
    Num := Num div I;
    if Num > 1 then Write (' X ');
  end;
end.
```

```
{2.3}
program Two3T89;
{ -- This program will display a word without its vowels. }
  const
    Vow = 'AEIOU';
  var
    Word: String[15];
    Ch:   Char;
    I:    Byte;

begin
  Write ('Enter word: ');  Readln (Word);
  for I := 1 to Length(Word) do begin
    Ch := Word[I];
    if Pos(Ch, Vow) = 0 then
      Write (Ch);
  end;
end.


{2.4}
program Two4T89;
{ -- This program produces the shortest possible identifiers. }
  var
    I, J, K: Byte;
    A:       Array[1..6] of String[10];
    S:       String[10];

begin
  for I := 1 to 6 do begin
    Write ('Enter name: ');  Readln (A[I]);
  end;
  for I := 1 to 6 do begin
    K := 1;  S := Copy(A[I], 1, 1);
    for J := 1 to 6 do
      { -- If S is same as beginning of another var, add letter. }
      while (I <> J)
      and (S = Copy(A[J], 1, K))
      and (K < Length(A[I])) do begin
        Inc(K);  S := S + Copy(A[I], K, 1);
      end;
    Writeln (S);
  end;  { -- for I }
end.
```

```
{2.5}
program Two5T89;
{ -- This program prints the # of distinguishable permutations. }
  var
    Word:   String[15];
    I, Len: Byte;
    LetPos: Integer;
    Let:    Array[1..26] of Byte;
    Num:    LongInt;

begin
  Write ('Enter word: ');  Readln (Word);
  Len := Length(Word);
  for I := 1 to 26 do Let[I] := 0;

  { -- Calculate Len factorial (assuming all different letters) }
  Num := 1;
  for I := 1 to Len do Num := Num * I;

  { -- Divide out of Num the factorials of the same letters }
  for I := 1 to Len do begin
    LetPos := Ord(Word[I]) - 64;
    Let[LetPos] := Let[LetPos] + 1;
    if Let[LetPos] > 1 then
      Num := Num div Let[LetPos];
  end;

  Writeln (Num);
end.
```

```
{2.6}
Program Two6T89;
{ -- This program underlines parts of a sentence between 2 *'s. }
uses Crt;
  const
    Dash = '-';
  var
    Sent:   String[40];
    I, Col: Byte;
    Under:  Boolean;
    Ch:     String[1];

begin
  Write ('Enter Sentence: ');  Readln (Sent);
  ClrScr;  Writeln (Sent);

  Under := False;  Col := 0;
  for I := 1 to Length(Sent) do begin
    Ch := Copy(Sent, I, 1);
    if Ch = '*' then
      { -- Change to Underline mode or un-underline mode. }
      Under := not Under
    else  { -- Display Char and underline if in underline mode. }
      begin
        Inc(Col);
        GotoXY (Col, 3);  Write (Ch);
        if Under then begin
          GotoXY (Col, 4);  Write (Dash);
        end;
      end;
  end;
  Writeln;
end.
```

```
{2.7}
program Two7T89;
{ -- This program will compute an expression containing + - * / }
  var
    St:                    String[10];
    NumSt:                 String[4];
    Num1, Num2, I, Result: Integer;
    Ch, Symbol:            Char;

begin
  Write ('Enter expression: ');  Readln (St);  NumSt := '';
  { -- Parse first number in Num1 and second number in Num2 }
  for I := 1 to Length(St) do begin
    Ch := St[I];
    if Ch in ['+', '-', '*', '/'] then
      begin
        Symbol := Ch;
        Val(NumSt, Num1, Result);  NumSt := '';
      end
    else
      NumSt := NumSt + Ch;
  end;
  Val (NumSt, Num2, Result);

  Case Symbol of
    '+':  Writeln (Num1 + Num2);
    '-':  Writeln (Num1 - Num2);
    '*':  Writeln (Num1 * Num2);
    '/':  Writeln (Num1 div Num2);
  end;
end.
```

```
{2.8}
program Two8T89;
{ -- This program will display the saddle point of a matrix. }
  var
    Rows, Cols, I, J, K:     Byte;
    Mat: Array[1..5,1..5] of Integer;
    Small, Large:            Boolean;

begin
  Write ('Enter # Rows, # Cols: ');  Readln (Rows, Cols);
  for I := 1 to Rows do
    for J := 1 to Cols do begin
      Write ('Enter Row', I, ' Col', J, ': ');
      Readln (Mat[I,J]);
    end;

  for I := 1 to Rows do
    for J := 1 to Cols do begin
      Small := True;
      for K := 1 to Cols do
        if (K <> J) and (Mat[I,J] >= Mat[I,K]) then
          Small := False;
      if Small then begin
        Large := True;
        for K := 1 to Rows do
          if (K <> I) and (Mat[I,J] <= Mat[K,J]) then
            Large := False;
        if Large then begin
          Write ('SADDLE POINT = ');
          Writeln (Mat[I,J], ' AT ROW ', I, ' COL ', J);
        end;
      end;  { -- if Small }
    end;  { -- for J }
end.
```

```
{2.9}
program Two9T89;
{ -- This program will sort a set of dates in increasing order. }
  const
    Mo: Array[1..12] of String[9] = ('JANUARY', 'FEBRUARY',
      'MARCH', 'APRIL', 'MAY', 'JUNE', 'JULY', 'AUGUST',
      'SEPTEMBER', 'OCTOBER', 'NOVEMBER', 'DECEMBER');
  var
    I, J, N, Temp: Integer;
    M:     Array[1..10] of String[9];
    D, Y:  Array[1..10] of Integer;
    Sort:  Array[1..10] of LongInt;
    Index: Array[1..10] of Integer;

begin
  Write ('Enter # of dates: ');  Readln (N);
  for I := 1 to N do begin
    Write ('Enter month: ');  Readln (M[I]);
    Write ('Enter day:   ');  Readln (D[I]);
    Write ('Enter year:  ');  Readln (Y[I]);
    Writeln;

    { -- Combine Year, Month, Day (in that order) for sorting. }
    J := 1;
    while (J < 13) and (M[I] <> Mo[J]) do Inc(J);
    Sort[I] := ((Y[I] * 100) + J) * 100 + D[I];
    Index[I] := I;
  end;

  { -- Sort dates according to values in Sort[] and swap Index. }
  for I := 1 to N - 1 do
    for J := I + 1 to N do
      if Sort[Index[I]] > Sort[Index[J]] then begin
        Temp := Index[I]; Index[I] := Index[J]; Index[J] := Temp;
      end;

  for I := 1 to N do
    Writeln (M[Index[I]], ' ', D[Index[I]], ' ', Y[Index[I]]);
end.


{2.10}
program Two10T89;
{ -- This program displays class grades and the averages. }
uses Crt;
  const
    Name: Array[1..5] of String[8] =
     ('D. WOOLY', 'M. SMITH', 'C. BROWN', 'R. GREEN', 'T. STONE');
    Quiz: Array[1..5,1..4] of Byte =
     ((100, 92, 90, 90),  (55, 75, 70, 65),  (94, 70, 62, 70),
       (90, 74, 80, 85),  (85, 98, 100, 70));
  var
    I, J, Scr:  Byte;
    Sum, Total: Real;
```

```
begin
  for Scr := 1 to 2 do begin
    ClrScr;
    if Scr = 2 then begin
      Writeln ('           MS. HEINDEL''S MUSIC CLASS');
      Writeln ('                  FINAL GRADES');
      Writeln ('                  SPRING 1989');
      Writeln;
    end;
    Write ('  NAME       Q1     Q2     Q3     Q4');
    if Scr = 2 then
      Writeln ('    AVERAGE')
    else
      Writeln;
    Writeln;

    for I := 1 to 5 do begin
      Write (Name[I]);  Sum := 0;
      for J := 1 to 4 do begin
        Write (Quiz[I,J]:7);  Sum := Sum + Quiz[I,J];
      end;
      if Scr = 2 then
        Writeln (' ':4, Sum / 4: 5:2)
      else
        Writeln;
    end;
    Writeln;
    if Scr = 1 then begin
     Write  ('Enter 5 grades for quiz 4: ');
     Readln(Quiz[1,4], Quiz[2,4], Quiz[3,4], Quiz[4,4],Quiz[5,4]);
    end;
  end;  { -- for Scr }

  { -- Display Column averages and Class average. }
  Write ('AVERAGE:');  Total := 0;
  for I := 1 to 4 do begin
    Sum := 0;
    for J := 1 to 5 do
      Sum := Sum + Quiz[J,I];
    Write ('  ', Sum / 5: 5:2);
    Total := Total + Sum;
  end;
  Writeln;  Writeln;
  Writeln ('CLASS AVERAGE: ', Total / 20: 5:2);
end.
```

```
{3.1}
program Thr1T89;
{ -- This program will determine if a word is correctly spelled. }
  var
    St, Part: String[12];
    Correct:  Boolean;
    I, Len:   Byte;

begin
  Write ('Enter word: ');  Readln (St);
  Len := Length(St);   Correct := True;

  { -- Check for E before suffixes ING, IBLE, ABLE }
  if Len >= 4 then begin
    Part := Copy(St, Len-2, 3);
    if (Part = 'ING') and (Copy(St, Len-3, 1) = 'E') then
      Correct := False;
  end;
  if Len >= 5 then begin
    Part := Copy(St, Len-3, 4);
    if ((Part = 'IBLE') or (Part = 'ABLE')) and
    (Copy(St, Len-4, 1) = 'E')
      then Correct := False;
  end;

  { -- Check if IE after C. }
  Part := St;  I := Pos('IE', Part);
  while (I > 0) and Correct do begin
    Dec(I);
    if I >= 1 then
      if Copy(Part, I, 1) = 'C' then Correct := False;
    Part := Copy (Part, I+3, Length(Part) - (I+2));
    I := Pos('IE', Part);
  end;

  { -- Check if EI not after C. }
  Part := St;  I := Pos('EI', Part);
  while (I > 0) and Correct do begin
    Correct := False;
    if I >= 2 then
      if Copy(Part, I-1, 1) = 'C' then Correct := True;
    Part := Copy (Part, I+3, Length(Part) - (I+2));
    I := Pos('EI', Part);
  end;

  { -- Check for 3 consecutive same letters. }
  I := 1;
  while (I <= Len-2) and Correct do begin
    if (Copy(St,I,1) = Copy(St,I+1,1))
    and (Copy(St,I,1) = Copy(St,I+2,1))
      then Correct := False;
    Inc(I);
  end;
  if Correct then
    Writeln ('CORRECT')
```

```
  else
    Writeln ('MISSPELLED');
end.



{3.2}
program Thr2T89;
{ -- This program finds the positive root of V for an equation. }
  const
    P: Array[1..5] of Real = (0.05, 0.7, 10.0, 70.0, 30.0);
  var
    I:              Byte;
    ZeroFound:      Boolean;
    Neg, Pos, T,
    V, VTry, NextV: Real;

function FNA(V: Real): Real;
{ -- This function computes the value of P for the equation. }
begin
  FNA := P[I]*V*V*V*14.14 - P[I]*V*9062.599 - 23511.9*V*V +
         988686.1*V - 400943.0;
end;

begin
  for I := 1 TO 5 do begin
    if I = 5 then begin  { -- Allow for 1 input value }
       Writeln;
       Write ('Enter value for P: ');  Readln (P[5]);
    end;
    VTry := 0;  ZeroFound := False;
    repeat
      NextV := VTry + 1;
      if (FNA(VTry) * FNA(NextV) <= 0) and (FNA(NextV) <> 0) then
      { -- Sign change has occurred }
      begin
        Neg := VTry;  Pos := NextV;
        if FNA(Neg) > FNA(Pos) then begin
          T := Neg;  Neg := Pos;  Pos := T;
        end;
        repeat
          V := (Neg + Pos) / 2.0;
          if FNA(V) < 0 then Neg := V else Pos := V;
        until ABS(Neg - Pos) <= 0.00005;
        Writeln ('P = ', P[I]:5:2, '  V = ', V:6:4);
        ZeroFound := True;
      end;
      VTry := VTry + 1;
    until ZeroFound or (VTry > 2);
  end;  { -- next I }
end.
```

```
{3.3}
program Thr3T89;
{ -- This program will magnify an input positive integer. }
uses Crt;
  const
    Num: Array[0..9] of String[7] = ('123567', '36', '13457',
         '13467', '2346', '12467', '124567', '136',
         '1234567', '12346');
  var
    NSt: String[4];
    Col, Part, I, J, K, N, Magn: Integer;

procedure DisplayPart (Part: Integer);
{ -- This procedure displays a vertical or horizontal line seg. }
begin
  Case Part of
    1: begin
         GotoXY (Col, 1);  for K := 1 to Magn do Write ('****');
         Writeln;
       end;
    2: begin
         for K := 1 to Magn*2+1 do begin
           GotoXY(Col, K);  Write('*');  end;
       end;
    3: begin
         for K := 1 to Magn*2+1 do begin
           GotoXY(Col+Magn*4-1, K);  Write('*');  end;
       end;
    4: begin
         GotoXY(Col, Magn*2+1);
         for K := 1 to Magn do Write ('****');  Writeln;
       end;
    5: begin
         for K := Magn*2+1 to Magn*4+1 do begin
           GotoXY(Col, K);  Write('*');  end;
       end;
    6: begin
         for K := Magn*2+1 to Magn*4+1 do begin
           GotoXY(Col+Magn*4-1, K);  Write('*');  end;
       end;
    7: begin
        GotoXY(Col, Magn*4+1);
        for K := 1 to Magn do Write ('****');  Writeln;
      end;
  end;
end;

begin
  Write ('Enter number: ');  Readln (NSt);
  Write ('Enter magnification: ');  Readln (Magn);
  ClrScr;
  for I := 1 to Length(NSt) do begin
    N := Ord(NSt[I]) - 48;
    Col := (I-1) * Magn * 6 + 1;
    for J := 1 to Length(Num[N]) do begin
```

```
      Part := Ord(Num[N,J]) - 48;
      DisplayPart(Part);
    end;
  end;
end.


{3.4}
program Thr4T89;
{ -- This program produces a calendar for a given month/year. }
{ -- January 1, 1901 is a Tuesday. }
uses Crt;
  const
    Mo: Array[1..12] of String[9] = ('JANUARY', 'FEBRUARY',
      'MARCH', 'APRIL', 'MAY', 'JUNE', 'JULY', 'AUGUST',
      'SEPTEMBER', 'OCTOBER', 'NOVEMBER', 'DECEMBER');
    DaysInMo: Array[1..12] of Byte =
      (31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31);
  var
    Year, Days:                    Integer;
    Month, Day, Col, Leap, I, Mid: Byte;

begin
  Write ('Enter month, year: ');  Readln (Month, Year);
  ClrScr;
  Mid := 2 + (26 - (Length(Mo[Month]) + 5)) div 2;
  Writeln (' ':Mid, Mo[Month], ' ', Year);
  Writeln ('   S   M   T   W   T   F   S');
  Writeln ('  --------------------------');

  { -- # of days from 1/1/1901 to the last day of prior month. }
  Days := (Year - 1901) * 365 + ((Year - 1901) div 4);
  for I := 1 to Month - 1 do
    Days := Days + DaysInMo[I];
  if (Month > 2) and (Year mod 4 = 0) then
    Inc(Days);

  { -- Determine first day of month. }
  Day := (Days + 1) mod 7;
  { -- Day =0 (Mon), =1 (Tue), =5 (Sat), =6 (Sun) }
  Col := (Day + 1) mod 7;
  { -- Day = 0,1,2,3,4,5,6 Sun,Mon,Tue..Sat }
  Leap := 0;
  if (Month = 2) and (Year mod 4 = 0) then  { -- Leap year month }
    Leap := 1;

  { -- Display Month Calendar }
  if Col > 0 then Write (' ':Col*4);
  for I := 1 to DaysInMo[Month] + Leap do begin
    Write (I:4);
    Col := (Col + 1) mod 7;
    if Col = 0 then Writeln;
  end;
end.
```

```
{3.5}
program Thr5T89;
{ -- This program positions 5 queens on the board so none attack.}
uses Crt;
  const
    Dimen = 5;
  type
    Board = Array [1..8] of Byte;
  var
    I, Row, Col:   Byte;
    Configuration: Board;

function IsSafe (Configuration: Board; Row, Col: Byte): Boolean;
{ -- This function returns True if no queen can attack another. }
  var
    I:      Byte;
    Safety: Boolean;
  begin
    Safety := True;
    for I := 1 to Col-1 do
      if ((Configuration[I] + I) = (Row + Col))
      or ((Configuration[I] - I) = (Row - Col))
      or  (Configuration[I] = Row) then
        Safety := False;
      IsSafe := Safety
  end;

begin
  ClrScr;
  Writeln ('ROWS = 1 2 3 4 5');
  Writeln ('----------------');
  Writeln ('COLUMNS');
  Col := 1;  Row := 1;
  repeat
    while (Row <= Dimen) and (Col <= Dimen) do
      if IsSafe(Configuration, Row, Col) then
        { -- Advance the Column }
        begin
          Configuration[Col] := Row;  Inc(Col);  Row := 1
        end
      else
        Inc(Row);

    if (Row = Dimen + 1) then begin  { -- Retreat the Column }
      Dec(Col);
      Row := Configuration[Col] + 1
    end;

    if (Col = Dimen + 1) then begin
      { -- Display Solution and retreat column. }
      Write (' ':7);
      for I := 1 to Dimen do
        Write (Configuration[I], ' ');
      Writeln;
      Dec(Col);
```

```
      Row := Configuration[Col] + 1
    end;
  until (Col = 1) and (Row = Dimen + 1);
end.
```

```
{3.6}
program Thr6T89;
{ -- This program prints the product of 2 large integers in Base.}
  var
    AStr, BStr:           String[31];
    LenA, LenB:           Byte;
    A, B, Prod:           Array[1..61] of Byte;
    I, J, S, Carry, Base: Byte;
    Sign:                 -1..1;

begin
  Write ('Enter base: ');  Readln (Base);
  Write ('Enter first integer:  ');  Readln (AStr);
  Write ('Enter second integer: ');  Readln (BStr);

  { -- Determine if signs are positive or negative, display sign.}
  Sign := 1;
  if Copy(AStr, 1, 1) = '-' then begin
    AStr := Copy(AStr, 2, Length(AStr)-1);   Sign := -1;
  end;
  if Copy(BStr, 1, 1) = '-' then begin
    BStr := Copy(BStr, 2, Length(BStr)-1);  Sign := Sign * -1;
  end;
  if Sign < 0 then Write ('-');

  { -- Store String digits into numerical arrays. }
  LenA := Length(AStr);  LenB := Length(BStr);
  for I := LenA downto 1 do
    A[LenA - I + 1] := Ord(AStr[I]) - 48;
  for I := LenB downto 1 do
    B[LenB - I + 1] := Ord(BStr[I]) - 48;
  for I := 1 to 61 do Prod[I] := 0;

  { -- Multiply 2 numbers as a person would with carries. }
  for I := 1 to LenB do begin
    Carry := 0;
    for J := 1 to LenA do begin
      S := I + J - 1;
      Prod[S] := Prod[S] + B[I] * A[J] + Carry;
      Carry := Prod[S] div Base;;
      Prod[S] := Prod[S] - Carry * Base;
    end;
    If Carry > 0 then Prod[S+1] := Carry;
  end;

  { -- Display product }
  if Carry > 0 then Write (Prod[S+1]);
  for I := S downto 1 do
    Write (Prod[I]);

end.
```

```
{3.7}
program Thr7T89;
{ -- This program computes most efficient change without a coin. }
  const
    Coin: Array[1..4] of String[7] =
       ('QUARTER', 'DIME', 'NICKEL', 'PENNY');
    CVal: Array[1..4] of Byte = (25, 10, 5, 1);
  var
    CoinM:         String[7];
    Num:           Array[1..4] of Byte;
    Cost, Amount: Real;
    Change, I, C: Byte;

procedure MakeChange (X, St, En: Integer);
{ -- Gives most efficient change of X using CoinValues[St..En] }
  var I: Integer;
begin
  for I := St to En do begin
    Num[I] := X div CVal[I];
    X := X - Num[I] * CVal[I];
  end;
end;

procedure DoMissingCoin (C: Byte);
{ -- Make up change for missing coin (if it was used in solution)}
begin
  if C = 1 then  { -- NO Quarters }
    { -- Determine most efficient way withoug quarters }
    MakeChange (Change, 2, 4)
  else if C = 2 then  { -- NO Dimes }
    { -- Add 2 nickels for every dime }
    Num[3] := Num[3] + Num[2] * 2
  else if C = 3 then  { -- NO Nickels }
    { -- IF a nickel then IF at least 1 quarter then
                              Make 3 dimes and 1 less quarter
                            ELSE make 5 more pennies with 1 nickel }
   if Num[3] = 1 then
    if Num[1] > 0 then begin
      Num[2] := Num[2] + 3;  Num[1] := Num[1] - 1; end
    else
      Num[4] := Num[4] + 5;
end;

begin
  Write ('Enter cost, amount: ');  Readln (Cost, Amount);
  Write ('Enter missing coin: ');  Readln (CoinM);
  Change := Trunc((Amount - Cost) * 100 + 0.01);
  MakeChange (Change, 1, 4);  { -- Calculate denominations }

  C := 1;
  while (C < 5) and (CoinM <> Coin[C]) do  Inc(C);
  DoMissingCoin(C);

  { -- Display number of coins of each coin that was used. }
  for I := 4 downto 1 do begin
```

```
    if I <> C then begin
      Write (Num[I],'  ');
      if (I = 4) and (Num[I] <> 1) then Writeln ('PENNIES')
      else begin
        Write (Coin[I]);
        if Num[I] <> 1 then Write ('S');
        Writeln;
      end;
    end;
  end;
  Write ('TOTAL CHANGE RETURNED = ',Change,' CENT');
  if Change <> 1 then Write ('S');
  Writeln;
end.
```

```
{3.8}
program Thr8T89;
{ -- This program displays the coordinates of binary rectangles. }
  var
    A:                              Array [1..6,1..7] of 0..1;
    I, J, K, Num, Den:              Byte;
    RowLen, ColLen, RowSt, ColSt: Byte;
    Rect:                           Boolean;

begin
  { -- Convert 6 numbers to binary representation. }
  for I := 1 to 6 do begin
    Write ('Enter number: ');  Readln (Num);
    Den := 128;
    for J := 6 downto 0 do begin
      Den := Den div 2;  { -- Den = 2^J }
      A[I,7-J] := Num div Den;
      Num := Num - A[I,7-J] * Den;
    end;
  end;
  Writeln;

  { -- Display the 6 row X 7 col grid of 0s and 1s. }
  for I := 1 to 6 do begin
    for J := 1 to 7 do
      Write (A[I,J]);
    Writeln;
  end;
  Writeln;

  { -- Find largest solid rectangles of 1s. }
  for RowLen := 6 downto 2 do
    for ColLen := 7 downto 2 do
      for RowSt := 1 to 7 - RowLen do
        for ColSt := 1 to 8 - ColLen do begin
          Rect := True;
          for I := RowSt to RowSt + RowLen - 1 do begin
            J := ColSt;
            while (J <= ColSt + ColLen - 1) and Rect do begin
              if A[I,J] = 0 then Rect := False;
              J := J + 1;
            end;
          end;  { -- for I }
          if Rect then begin  { -- Display rectangle coordinates }
            Write ('(', RowSt, ',', ColSt, ')');
            Write ('(', RowSt + RowLen - 1, ',');
            Writeln (ColSt + ColLen - 1, ')');
            { -- Clear rectangle 1s to 0s }
            for I := RowSt to RowSt + RowLen - 1 do
              for J := ColSt to ColSt + ColLen - 1 do
                A[I,J] := 0;
          end;
        end;  { -- for ColSt }
end.
```

```
{3.9}
program Thr9T89;
{ -- This program determines the 5 word combination for BINGO. }
  type
    String5 =    String[5];
    OfWord=      Array [1..5] of String[1];
    Array3=      Array [1..5,1..2] of Byte;
    ArrayWord3 = Array [1..5,1..2] of String5;

  const
    LetterValue: Array[1..26] of Byte =
    (9, 14, 1, 16, 20, 5, 10, 2, 21, 17, 6, 25,
    12, 3, 22, 18, 24, 7, 13, 26, 15, 11, 19, 4, 23, 8);

  var
    I, J, K, Sum, Col, Row, MaxCol, St, En,
    WordNum:  Byte;
    Max:      Integer;
    Word:     String5;
    Letter:   Char;
    Letters:  OfWord;
    Highest:  Array3;
    HighWord: ArrayWord3;
    MaxSum:   Array [1..2] of Integer;

procedure UseWord (Word: String5;  Sum: Integer);
{ -- This procedure replaces a word if the sum of new word is >. }
  const
    Bingo = 'BINGO';
  var
    Row, Col: Byte;

begin
  for Col := 1 to 2 do
    for Row := 1 to 5 do
      if Letters[Col] = Copy(Bingo,Row,1) then
        if Sum > Highest [Row, Col] then begin
          Highest  [Row, Col] := Sum;
          HighWord [Row, Col] := Word;
        end;
end;

procedure DisplayValues;
{ -- This procedure displays the two columns of values on screen.}
begin
  Writeln;
  Max := 0;
  for I := 1 to 2 do
    MaxSum[I] := 0;
  St := 1;  En :=2;
  for Row := 1 to 5 do begin
    for Col := St to En do begin
      Write(HighWord [Row, Col]:5, Highest[Row, Col]:4, ' ':3);
      MaxSum[Col] := MaxSum[Col] + Highest[Row, Col];
    end; {for Row}
```

```
    Writeln;
  end; {for Col}

  { -- Determine maximum column and display *** }
  For Col := St to En do begin
    Write (' ': 3 + Col * 3, MaxSum[Col] :3);
    If MaxSum[Col] > Max then begin
      Max := MaxSum[Col];
      MaxCol := Col;
    end;
  end; {for Col}
  Writeln;
  if MaxCol = 1 then
    Writeln (' ':6, '***')
  else
    Writeln (' ':18, '***');
  Writeln;
end;

begin
  HighWord[1,1] := 'BIBLE';  HighWord[1,2] := 'OBESE';
  HighWord[2,1] := 'IDYLL';  HighWord[2,2] := 'TITHE';
  HighWord[3,1] := 'NOISE';  HighWord[3,2] := 'INLET';
  HighWord[4,1] := 'GULLY';  HighWord[4,2] := 'IGLOO';
  HighWord[5,1] := 'OBESE';  HighWord[5,2] := 'TOWER';

  { -- Determine numerical values for given words. }
  for Col := 1 to 2 do
    for Row := 1 to 5 do begin
      Sum := 0;
      for I := 1 to 5 do begin
        Word := HighWord[Row,Col];
        Letter := Word[I];
        Sum := Sum + LetterValue[Ord(Letter) - 64];
      end; {for I}
      Highest[Row,Col] := Sum;
    end;
  repeat
    DisplayValues;
    { -- Allow new words to be entered and computed. }
    Write ('Enter word: ');  Readln (Word);
    while Length(Word) = 5 do begin
      Sum := 0;
      for I := 1 to 5 do begin
        Letter := Word[I];
        Letters[I] := Letter;
        Sum := Sum + LetterValue[Ord(Letter) - 64];
      end; {for I}
      UseWord (Word, Sum);
      Write ('Enter word: ');  Readln (Word);
    end; {while}
  until Word = 'QUIT';
end.
```

```
{3.10}
program Thr10T89;
{ -- This program displays the number of distinguishable
  -- permutations for a cube w/sides input as color symbols. }

  const
    Side: Array[1..6] of String[6] =
    ('TOP', 'FRONT', 'BOTTOM', 'BACK', 'RIGHT', 'LEFT');
  type
    CubeArray = Array[1..6] of Char;
  var
    I, J, K,
    Rot, Num: Byte;
    Cube, C:  CubeArray;
    Unique:   Array[1..24, 1..6] of Char;
    Valid:    Boolean;

procedure Permute (var C: CubeArray;  Rot: Byte);
{ -- Swaps the colors on the squares of the Cube. }
  var
    Temp:   Char;
    Square: Byte;

begin
  if Rot mod 4 > 0 then
    { -- Rotate cube clock-wise about vertical axis }
    begin
      Temp := C[ 2];
      C[2] := C[5];  C[5] := C[4];
      C[4] := C[6];  C[6] := Temp;
    end
  else
  { -- Place a new square ((Rot div 4) + 1) on the top position. }
  begin
    Square := (Rot div 4) + 1;
    C[1] := Cube[Square];
    Case Square of
      1: begin
          for I := 2 to 6 do
            C[I] := Cube[I]
         end;
      2: begin
           C[2] := Cube[3];  C[3] := Cube[4];
           C[4] := Cube[1];  C[5] := Cube[5];  C[6] := Cube[6];
         end;
      3: begin
           C[2] := Cube[4];  C[3] := Cube[1];
           C[4] := Cube[2];  C[5] := Cube[5];  C[6] := Cube[6];
         end;
      4: begin
           C[2] := Cube[1];  C[3] := Cube[2];
           C[4] := Cube[3];  C[5] := Cube[5];  C[6] := Cube[6];
         end;
      5: begin
           C[2] := Cube[2];  C[3] := Cube[6];
```

```
            C[4] := Cube[4];  C[5] := Cube[3];  C[6] := Cube[1];
          end;
        6: begin
            C[2] := Cube[2];  C[3] := Cube[5];
            C[4] := Cube[4];  C[5] := Cube[1];  C[6] := Cube[3];
          end;
    end; { -- case }
  end; { -- if }
end;  { -- Procedure }


begin
  { -- Assign colors to original 4 cubes. }
  { -- [.,#] # is  1= Top, 2= Front, 3= Bot, 4= Bk, 5= Rt, 6= Lt }
  for I := 1 to 6 do begin
    Write ('Enter ', Side[I], ' side: ');  Readln (Cube[I]);
  end;
  Num := 0;

  { -- Rotate cubes and check if it is unique. }
  for Rot := 0 to 23 do begin
    Permute (C, Rot);
    if Rot = 0 then
      Valid := True
    else
      begin
      { -- Check if permuted cube is identical to previous cubes.}
        J := 1;
        repeat
          Valid := False;
          for K := 1 to 6 do
            If C[K] <> Unique[J,K] then Valid := True;
          Inc(J);
        until (J > Num) or not Valid;
      end;

    If Valid then begin  { -- Add new cube to unique cube list }
      Inc(Num);
      for I := 1 to 6 do
        Unique[Num, I] := C[I];
    end;
  end; { Rot }
  Writeln ('NUMBER OF DISTINGUISHABLE CUBES = ', Num);
end.
```