```
{ -- FLORIDA HIGH SCHOOLS COMPUTING COMPETITION '87 }
{ -- PASCAL PROGRAM SOLUTIONS }


{1.1}
program One1T87;
{ -- This program will print out the sign of a given number. }
  var
    Num: Real;

begin
  Write ('Enter number: ');
  Readln (Num);
  if Num > 0 then
    Writeln ('POSITIVE')
  else if Num < 0 then
    Writeln ('NEGATIVE')
  else
    Writeln ('ZERO');
end.

{1.2}
program One2T87;
{ -- This program will sum the numbers n, n+1, ... n+20. }
  var
    N, I, Sum: Integer;

begin
  Write ('Enter n: ');  Readln (N);
  Sum := 0;
  for I := 0 to 20 do
    Sum := Sum + N + I;
  Writeln ('SUM = ', Sum);
end.

{1.3}
program One3T87;
{ -- This program will print PROBLEM THREE diagonally. }
uses Crt;
  const
    St = 'PROBLEM THREE';
  var
    Row, Col, I, L: Byte;

begin
  ClrScr;
  L := Length (St);
  Row := (24 - L) div 2;
  Col := (80 - L) div 2;
  for I := 1 to L do begin
    GotoXY (Col+I, Row+I);
    Write (Copy (St,I,1));
  end;
end.
```

```
{1.4}
program One4T87;
{ -- This program displays the numbers on the sides of a die. }
  var
    Top, Front, Right: Byte;

begin
  Write ('Enter number on top: ');    Readln (Top);
  Write ('Enter number on front: ');  Readln (Front);
  Write ('Enter number on right: ');  Readln (Right);

  Writeln ('TOP= ',    Top);
  Writeln ('FRONT= ',  Front);
  Writeln ('RIGHT= ',  Right);
  Writeln ('BOTTOM= ', 7 - Top);
  Writeln ('BACK= ',   7 - Front);
  Writeln ('LEFT= ',   7 - Right);
end.


{1.5}
program One5T87;
{ -- This program will fill the screen with random characters. }
uses Crt;
  var
    Row, Col: Byte;

begin
  Randomize;
  for Row := 1 to 24 do
    for Col := 1 to 80 do
      Write( Chr (Random (96) + 33) );
  repeat until KeyPressed;
  ClrScr;
end.


{1.6}
program One6T87;
{ -- This program will display a rectangular array of periods. }
uses Crt;
  var
    Row1, Col1, Row2, Col2, I, J: Byte;

begin
  Write ('Enter coordinates: ');
  Readln (Row1, Col1, Row2, Col2);
  ClrScr;
  for I := Row1 to Row2 do
    for J := Col1 to Col2 do begin
      GotoXY (J, I);  Write ('.');
    end;
end.
```

```
{1.7}
program One7T87;
{ -- This program will generate 10 random numbers given a seed. }
  var
    Seed, I: Integer;

begin
  Write ('Enter seed: ');  Readln (Seed);
  for I := 1 to 10 do begin
    Seed := (Seed * 421 + 1) mod 100;
    Writeln (Seed);
  end;
end.


{1.8}
program One8T87;
{ -- This program will determine the mass of a fish tank. }
  var
    K, L, W, H, Mass, InchCubed: Real;

begin
  Write ('Enter K, L, W, H: ');  Readln (K, L, W, H);
  InchCubed := 2.54 * 2.54 * 2.54;
  Mass := L * 12 * W * 12 * H * 12 * InchCubed;
  Mass := Mass / 1000 + K;
  Writeln (Mass: 8:2, ' KILOGRAMS');
end.


{1.9}
program One9T87;
{ -- This program will display 21 rows of letters. }
uses Crt;
  var
    Row, I: Integer;
    Ch:     Char;

begin
  ClrScr;
  for Row := 1 to 21 do begin
    Ch := Chr(64 + Row);
    if Row mod 2 = 1 then
      for I := 1 to 31 do
        Write (Ch)
    else begin
      Write (Ch);
      for I := 1 to 10 do
        Write ('  ', Ch);
    end;
    Writeln;
  end;
end.
```

```
{1.10}
program One10T87;
{ -- This program will display the time needed to read a book. }
  const
    Title : Array [1..4] of String[30] =
      ('THE HISTORY OF THE COMPUTER', 'THE RED DOG RUNS',
       'EATING APPLE PIE', 'THE ART OF WINNING');
    Pages : Array [1..4] of Integer = (400, 200, 150, 250);

  var
    BookTitle:   String[30];
    MP, Minutes: Integer;
    BookFound:   Boolean;
    I, Hours:    Integer;

begin
  Write ('Enter book title: ');           Readln (BookTitle);
  Write ('Enter rate (minutes/page): ');  Readln (MP);

  I := 0;  BookFound := False;
  repeat
    Inc(I);
    BookFound := BookTitle = Title[I];
  until (I > 4) or BookFound;

  Minutes := MP * Pages[I];
  Hours := Trunc(Minutes) div 60;
  Minutes := Minutes - Hours * 60;
  Write (Hours, ' HOURS  ', Trunc(Minutes), ' MINUTES');
end.
```

```
{2.1}
program Two1T87;
{ -- This program will rotate a string N times to the left. }
  var
    St:   String[10];
    L, N: Byte;

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

  L := Length(St);
  N := N mod L;
  Write   (Copy (St, N+1, L-N));
  Writeln (Copy (St, 1, N));
end.


{2.2}
program Two2T87;
{ -- This program will determine the number of diskettes bought. }
  var
    Vers, Maxs, Wabs: Integer;

begin
  for Vers := 1 to 98 do
    for Maxs := 1 to 99 - Vers do begin
      Wabs := 100 - Maxs - Vers;
      if (Wabs > 0) and
      (Vers * 225 + Maxs * 297 + Wabs * 120 = 23607) then
        begin
          Writeln (Vers,' VERS  ', Maxs,' MAXS  ', WABS,' Wabs');
          Exit;
        end;
    end;
end.
```

```
{2.3}
program Two3T87;
{ -- This program will display a subset of random numbers. }
uses Crt;
  var
    SetOfNum: Array [1..16] of Integer;
    Nums:     Array [1..5]  of Integer;
    NewNum:   Boolean;
    Ch:       Char;
    Num, I, NumDisplayed, LastIndex:  Integer;

begin
  I := 0;
  repeat
    Inc(I);
    Write ('Enter list item: ');  Readln (SetOfNum[I]);
  until SetofNum[I] < 0;
  LastIndex := I - 1;

  Randomize;
  repeat
    NumDisplayed := 0;
    repeat
      repeat  { -- Get unique random number }
        Num := SetOfNum[ Random(LastIndex) + 1 ];
        NewNum := True;
        for I := 1 to NumDisplayed do
          if Num = Nums[I] then
            NewNum := False;
      until NewNum = True;

      Writeln (Num);
      Inc(NumDisplayed);
      Nums[NumDisplayed] := Num;
    until NumDisplayed = 5;

    Writeln ('PRESS ANY KEY');
    repeat until KeyPressed;
    Ch := ReadKey;
  until Ch = Chr(27);
end.
```

```
{2.4}
program Two4T87;
{ -- This program will display all partitioned sum of a number. }
  var
    Num, I, J: Byte;

begin
  Write ('Enter a number less than 20: ');  Readln (Num);
  for I := Num downto 1 do
    if Num mod I = 0 then begin
      Write (' ': 30 - (Num div I));
      Write (I);
      for J := 2 to Num div I do
        Write ('+', I);
      Writeln;
    end;
end.


{2.5}
program Two5T87;
{ -- This program will calculate the fractional value. }

  var
    St:          String[3];
    A:           Array [1..3] of Integer;
    Num, Den, I: Integer;

begin
  Write ('Enter word: ');  Readln (St);
  for I := 1 to 3 do
    A[I] := Ord(St[I]) - 64;
  Num := A[1] * A[2] + A[2] * A[3] + A[3] * A[1];
  Den := A[1] * A[2] * A[3];
  for I := Den downto 1 do
    if (Num mod I = 0) and (Den mod I = 0) then begin
      Writeln (Num div I, '/', Den div I);  Exit;
    end;
end.
```

```
{2.6}
program Two6T87;
{ -- This program will find a subset of integers. }
  var
    Item:              Array [1..8] of Integer;
    N, S, I, J, Sum, Temp, LastIndex: Integer;

begin
  I := 0;
  repeat
    Inc(I);
    Write ('Enter set item: ');  Readln (Item[I]);
  until Item[I] < 0;
  LastIndex := I - 1;
  Write ('Enter N: ');  Readln (N);
  Write ('Enter S: ');  Readln (S);

  for I := 1 to LastIndex - 1 do
    for J := I + 1 to LastIndex do
      if Item[I] > Item[J] then begin
        Temp := Item[I];  Item[I] := Item[J];  Item[J] := Temp;
      end;

  Sum := 0;
  for I := 1 to N do
    Sum := Sum + Item[I];

  If Sum <= S then
    begin
      Writeln ('YES');
      for I := 1 to N do
        Write (Item[I], '  ')
    end
  else
    Writeln ('NO');
end.
```

```
{2.7}
program Two7T87;
{ -- This program will determine if patterns are legal/illegal. }
  var
    St:       String[20];
    I, BA, A: Byte;
    Legal:    Boolean;

begin
  Write ('Enter pattern: ');  Readln (St);
  Legal := True;
  I := 1;

  if Copy(St, I, 1) <> 'A' then  { -- does not start with A }
    Legal := False
  else begin  { -- starts with A }
    Inc(I);
    while Copy (St, I, 2) = 'BA' do  { -- skip valid BA's }
      I := I + 2;

    A := I;  { -- A = position before finding trailing A's }
    while (I <= Length (St)) and Legal do begin
      if Copy(St, I, 1) <> 'A' then  { -- invalid trailing letter}
        Legal := False;
      Inc(I);
    end;
    if A = I then { -- no trailing A's }
      Legal := False;
  end;

  if not Legal then Write ('IL');
  Writeln ('LEGAL PATTERN');
end.

{2.8}
program Two8T87;
{ -- This program will find integers having F factors. }
  var
    I, J, M, N, F, NumF: Integer;

begin
  Write ('Enter M, N, F: ');  Readln (M, N, F);
  for I := M to N do begin
    NumF := 0;
    for J := 1 to Trunc(Sqrt(I)) do
      if I mod J = 0 then
        NumF := NumF + 2;
    if Sqrt(I) = Trunc(Sqrt(I)) then
      Dec(NumF);
    if NumF = F then
      Writeln (I);
  end;
end.
```

```
{2.9}
program Two9T87;
{ -- This program will alphabetize 5 words according to rules. }
  var
    Word:        Array [1..5]  of String[12];
    Word2:       Array [1..5]  of String[12];
    St:          Array [1..12] of String[1];
    Temp:        String[12];
    I, J, K, L:  Byte;

begin
  for I := 1 to 5 do begin
    Write ('Enter word ', I, ': ');  Readln (Word[I]);
    L := Length( Word[I] );
    for J := 1 to L do
      St[J] := Copy(Word[I], J, 1);
    { -- Alphabetize letters within word and make WORD2. }
    for J := 1 to L - 1 do
      for K := J + 1 to L do
        if St[J] > St[K] then begin
          Temp := St[J];  St[J] := St[K];  St[K] := Temp;
        end;
    Word2[I] := '';
    for J := 1 to L do
      Word2[I] := Word2[I] + St[J];
  end;

  { -- Alphabetize Words according to Word2. }
  for J := 1 to 4 do
    for K := J + 1 to 5 do
      if Word2[J] > Word2[K] then begin
      Temp := Word2[J];  Word2[J] := Word2[K];  Word2[K] := Temp;
      Temp := Word[J];   Word[J]  := Word[K];   Word[K]  := Temp;
    end;

  for I := 1 to 5 do
    Writeln (Word[I]);
end.
```

```
{2.10}
program Two10T87;
{ -- This program will produce a super-duper input routine. }
uses Crt;
  var
    Row, Col, Max, Tipe, InitCol: Byte;
    Ch:       Char;
    ValidCh:  Boolean;
    Entry:    String[20];

begin
  Write ('Enter ROW, COL: ');  Readln (Row, Col);
  Write ('Enter MAX: ');       Readln (Max);
  Write ('Enter TYPE: ');      Readln (Tipe);

  ClrScr;  Entry := '';  InitCol := Col;
  repeat
    GotoXY (Col, Row);
    repeat until KeyPressed;
    Ch := ReadKey;
    if Ch = Chr(8) then begin  { -- Backspace pressed }
      if Length(Entry) > 0 then begin
        Entry := Copy (Entry, 1, Length(Entry)-1);
        Dec(Col);
        GotoXY (Col, Row);  Write (' ');
      end
    end
    else begin
      ValidCh := Length(Entry) < Max;
      If ValidCh then
        Case Tipe of
          1:  if not (Ch in ['A'..'Z', ' ']) then
                ValidCh := False;
          2:  if not (Ch in ['0'..'9', '.']) then
                ValidCh := False;
          3:  begin
                if Col-InitCol in [2, 5] then
                  if Ch <> '-' then ValidCh := False
                  else
                else
                  if not (Ch in ['0'..'9']) then
                    ValidCh := False;
               end;
        end;

      if ValidCh then begin
        Write (Ch);
        Entry := Entry + Ch;
        Inc(Col);
      end;
    end;
  until Ch = Chr(13);
  GotoXY (InitCol, Row+2);  Writeln (Entry);
end.
```

```
{3.1}
program Thr1T87;
{ -- This program will determine if 2 words are closely spelled. }
  type
    String10 = String[10];
  var
    Word1, Word2:    String10;
    Close:           Boolean;
    Len1, Len2, Min: Byte;
    PosDif:          Byte;

function PositionDiffer ({using} Word1, Word2: String10;
                                  Min: Byte): {giving} Byte;
{ -- This function will find the first position that differs. }
  var
    I : Byte;

begin
    for I := 1 to Min do
      if Copy(Word1, I, 1) <> Copy(Word2, I, 1) then begin
        PositionDiffer := I;  Exit;
      end;
    PositionDiffer := Min + 1;
end;  { -- function }

begin
  Write ('Enter word 1: ');  Readln (Word1);
  Write ('Enter word 2: ');  Readln (Word2);
  Len1 := Length(Word1);
  Len2 := Length(Word2);

  Close := False;
  if Word1 = Word2 then  { -- Words are the same }
    Close := True
  else if Abs(Len1 - Len2) < 2 then begin  { -- Could be close }
    { -- Find first character that differs. }
    if Len1 < Len2 then
      Min := Len1
    else
      Min := Len2;
    PosDif := PositionDiffer (Word1, Word2, Min);
    If PosDif > Min then  { -- Close (Same, or differ by add/del)}
      Close := True
    else
      if Len1 = Len2 then  { -- Check if 1 letter changed/trans }
        begin
          if (PosDif < Len1) and
          (Copy(Word1, PosDif+1, 1) = Copy(Word2, PosDif, 1)) and
          (Copy(Word2, PosDif+1, 1) = Copy(Word1, PosDif, 1)) then
               Inc(PosDif);  { -- possible skip over }
          if Copy(Word1, PosDif+1, Len1 - PosDif + 1) =
             Copy(Word2, PosDif+1, Len2 - PosDif + 1) then
            Close := True;
        end
      else  { -- Lengths differ by 1, Check for insertion/delete }
```

```
        if Len2 < Len1 then begin
          if Copy (Word2, PosDif,   Len2 - PosDif + 1) =
             Copy (Word1, PosDif+1, Len1 - PosDif) then
            Close := True
          end
        else
          if Copy (Word1, PosDif,   Len1 - PosDif + 1) =
             Copy (Word2, PosDif+1, Len2 - PosDif) then
            Close := True;
    end;

  if Close then
    WriteLn ('CLOSE')
  else
    WriteLn ('NOT CLOSE');
end.
```

```
{3.2}
program Thr2T87;
{ -- This program will evaluate an NxN determinant for N=2,3,4. }
  var
    I, J, K:     Byte;
    A, B:        Array [1..4, 1..6] of Integer;
    Sum, Tot, N: Integer;
    Power:       Integer;

procedure EvaluateDetWithout ({using} K: Integer);
{ -- This procedure evaluates a 3 x 3 determinant w/o col K }
var
  I, J, S: Byte;

begin
  for I := 1 to 3 do begin
    S := 0;
    for J := 1 to 4 do
      if J <> K then begin  { -- Create an 3 row by 4 col array }
        Inc(S);
        B[I,S]   := A[I,J];
        B[I,S+3] := A[I,J];
      end;
  end;
  Sum := 0;
  for I := 1 to 3 do
    Sum := Sum + B[1,I]   * B[2,I+1] * B[3,I+2]
               - B[1,I+2] * B[2,I+1] * B[3,I];
end;

begin
  Write ('Enter dimension N: ');  Readln (N);
  for I := 1 to N do
    for J := 1 to N do begin
      Write ('Enter row ', I, ', col ', J, ': ');
```

```
      Readln (A[I,J]);
    end;

  if N = 2 then begin  { 2 x 2 determinant }
    Sum := A[1,1] * A[2,2] - A[1,2] * A[2,1];
    Writeln (Sum);
    end
  else if N = 3 then begin  { 3 x 3 determinant }
    EvaluateDetWithout (4);
    Writeln (Sum);
    end
  else begin
    Tot := 0;
    for K := 1 to 4 do begin
      EvaluateDetWithout (K);
      Power := 1;
      for I := 1 to K do
        Power := Power * (-1);
      Tot := Tot + Sum * A[4,K] * Power;
    end;
    WriteLn (Tot);
  end;
end.
```

```
{3.3}
program Thr3T87;
{ -- This program will display the number of word occurrences. }
  type
    String12 = String[12];
  var
    Lines:       String[255];
    Word:        Array [1..20] of String12;
    WordTot:     Array [1..20] of Byte;
    NextWord:    String12;
    NumOfWords:  Byte;
    NewWord:     Boolean;
    Start, I:    Byte;
    WordInd:     Byte;

function GetWord ({using} var Start: Byte): {giving} String12;
{ -- This procedure get the next word in the passage at Start. }
var
  I:         Byte;
  NextWord:  String12;
  Ch:        Char;
  EndOfWord: Boolean;

begin
  I := Start;  EndOfWord := False;  NextWord := '';
  repeat
    Ch := Lines[I];
    if Ch in ['A'..'Z', ''''] then
      NextWord := NextWord + Ch
    else
      EndOfWord := True;
    Inc(I);
  until (I > Length(Lines)) or EndOfWord;
  Start := I;  GetWord := NextWord;
end;

begin
  Write ('Enter text: ');  Readln (Lines);
  Start := 1;  NumOfWords := 0;
  repeat
    NextWord := GetWord(Start);
    if NextWord > '' then
      NewWord := True
    else
      NewWord := False;
    WordInd := 0;
    while (WordInd < NumOfWords) and NewWord do begin
      Inc(WordInd);
      if NextWord = Word[WordInd] then NewWord := False;
    end;
    if NewWord then begin  { -- Add new word to list of words }
      Inc(NumOfWords);
      Word[NumOfWords] := NextWord;
      WordTot[NumOfWords] := 1;
    end
```

```
    else  { -- Increment # of times this word appears }
      Inc( WordTot[WordInd] );
  until Start > Length(Lines);

  for I := 1 to NumOfWords do
    Writeln (WordTot[I], ' ', Word[I]);
end.
```

```
{3.4}
program Thr4T87;
{ -- This program will encrypt a string such that when this
  -- code is entered, the string will be reproduced. }
  var
    St:          String[50];
    I, NumOfCh: Byte;
    Result:      Integer;
    Ch, NextCh: Char;
    AscSt:       String[4];
    Asc:         Array [1..50] of Byte;
    CodeNum:     Byte;

begin
  Write ('Enter text: ');  Readln (St);
  NumOfCh := 0;  I := 1;
  while (I <= Length(St)) do begin
    Ch := St[I];  Inc(NumOfCh);
    if Ch = '\' then
      begin  { -- Either another / or ### follows }
        Inc(I);
        NextCh := St[I];
        if NextCh <> '\' then
          begin { -- Next 3 characters are the ASC code }
            AscSt := Copy (St, I, 3);
            Val (AscSt, Asc[NumOfCh], Result);
            I := I + 2;
          end
        else  { / follows }
          Asc[NumOfCh] := Ord(NextCh);
      end
    else  { -- A regular character }
      Asc[NumOfCh] := Ord(Ch);
    Inc(I);
  end;  { -- while I }

  { -- Encrypt code }
  for I := 1 to NumOfCh do begin
    CodeNum := 255 - Asc[I];
    If CodeNum in [32 .. 92] then begin
      Write (Char(CodeNum));
      if CodeNum = Ord('\') then
        Write ('\');
      end
    else  { -- Non printable }
      begin
        Str (1000 + CodeNum: 4, AscSt);
        Write ('\');  Write(Copy(AscSt, 2, 3));
      end;
  end;
end.
```

```
{3.5}
program Thr5T87;
{ -- This program will unscramble the numbers 5132, 4735, and
  -- 8014153 so that the first times the second equal the
  -- third with a missing digit }
  const
    A : Array [1..4] of Byte = (5, 1, 3, 2);
    B : Array [1..4] of Byte = (4, 7, 3, 5);
    C : Array [1..7] of Byte = (8, 0, 1, 4, 1, 5, 3);
  var
    I, J, K, L, Perm24:       Byte;
    Prod:                     LongInt;
    Result:                   Byte;
    ANum, BNum:               Array [1..24] of LongInt;
    St:                       String[8];
    PCh:                      Array [1..8] of Char;
    Match:                    Boolean;

begin
  { -- Generate 24 permuations of 5132 and 4735 each. }
  Perm24 := 0;
  for I := 1 to 4 do
    for J := 1 to 4 do
      for K := 1 to 4 do begin
        L := 4+3+2+1 -I-J-K;
        if (I=J) or (J=K) or (I=K) then { -- do nothing }
        else begin
          Inc(Perm24);
          ANum[Perm24] := A[I]*1000 + A[J]*100 + A[K]*10 + A[L];
          BNum[Perm24] := B[I]*1000 + B[J]*100 + B[K]*10 + B[L];
        end;
      end;  { -- for K }

  for I := 1 to 24 do
    for J := 1 to 24 do begin
      Prod := ANum[I] * BNum[J];
      if not (Prod < 10E6) then begin { -- has 8 digits }
        Str (Prod, St);
        for K := 1 to 8 do
          PCh[K] := St[K];
        L := 1;
        repeat
          Match := False;  K := 0;
          repeat
            Inc(K);
            if C[L] = Ord(PCh[K]) - Ord('0') then begin
              PCh[K] := ' ';
              Match := True;
            end
          until (K = 8) or Match;
          Inc(L);
        until (L > 7) or not Match;

        if Match then
          Writeln (ANum[I], '  ', BNum[J], '  ', St);
```

```
      end;  { -- if }
    end;  { -- for J }
end.



{3.6}
program Thr6T87;
{ -- This program will display the front colors on the Rubik's
  -- Pocket Cube after a move of T or F is performed. }
  const
    A : Array [1..24] of Char =
       ('W', 'W', 'W', 'W', 'Y', 'Y', 'Y', 'Y',
        'O', 'O', 'O', 'O', 'R', 'R', 'R', 'R',
        'G', 'G', 'G', 'G', 'B', 'B', 'B', 'B');
  var
    I, J:    Byte;
    Move, X: Char;

begin
  repeat
    Write ('Enter T, F, or Q: ');  Readln (Move);
    if Move = 'T' then
      begin
        X    := A[1];  A[1] := A[3];  A[3] := A[4];
        A[4] := A[2];  A[2] := X;
        X    := A[5];  A[5] := A[9];  A[9] := A[13];
        A[13]:= A[17]; A[17]:= X;
        X    := A[6];  A[6] := A[10]; A[10]:= A[14];
        A[14]:= A[18]; A[18]:= X;
      end
    else if Move = 'F' then
      begin
        X    := A[5];  A[5] := A[7];  A[7] := A[8];
        A[8] := A[6];  A[6] := X;
        X    := A[3];  A[3] := A[20]; A[20]:= A[22];
        A[22]:= A[9];  A[9] := X;
        X    := A[4];  A[4] := A[18]; A[18]:= A[21];
        A[21]:= A[11]; A[11]:= X;
      end;
    if Move <> 'Q' then begin
      Writeln (A[5], '  ', A[6]);
      Writeln (A[7], '  ', A[8]);
    end
  until Move = 'Q';
end.
```

```
{3.7}
program Thr7T87;
{ -- This program will simulate a drill of Adding Roman Numerals.}
uses Crt;
  const
    RN:  Array[1..7] of Char= ('M', 'D', 'C', 'L', 'X', 'V', 'I');
    RNV: Array[1..7] of Integer = (1000, 500, 100, 50, 10, 5, 1);
  var
    Option:       Byte;
    Name, Dayte:  String[8];

procedure Do3Problems;
{ -- This procedure will allow the user to do 3 addition problems}
  var
    I, J, K:      Byte;
    Right, Wrong: Byte;
    Prob, XX:     Byte;
    Num:          Array [1..3] of Byte;
    RNum:         Array [1..3] of String[12];
    Ans:          String[12];
    X:            Real;
    Miss:         Byte;
    L1, L2, Col:  Byte;
    Arabic:       Byte;
    Ri, Wr:       Array [1..3] of String[12];
    RiA:          Array [1..3] of Byte;

begin
  Right := 0;  Wrong := 0;
  for Prob := 1 to 3 do begin
    ClrScr;
    Randomize;
    Num[1] := Random(19) + 1;  Num[2] := Random(19) + 1;
    Num[3] := Num[1] + Num[2];  Arabic := Num[3];
    for K := 1 to 3 do
      RNum[K] := '';
    for K := 1 to 3 do
      for I := 1 to 7 do begin
        X := Num[K] / RNV[I];
        if (X < 2) and (X >= 9/5) and ((I=2) or (I=4) or (I=6))
        then { null }
        else begin
          XX := Trunc(X);
          If XX = 9 then
            RNum[K] := RNum[K] + RN[I] + RN[I-2]
          else if XX = 4 then
            RNum[K] := RNum[K] + RN[I] + RN[I-1]
          else if XX > 0 then
            for J := 1 to XX do
              RNum[K] := RNum[K] + RN[I];
          Num[K] := Num[K] - RNV[I] * XX;
        end;
      end; { -- for I }
```

```
    { -- Display Problem }
    GotoXY (15, 10);  Write (RNum[1]);
    L1 := Length(RNum[1]);  L2 := Length(RNum[2]);
    Col := 15 + (L1 - L2) - 2;
    GotoXY (Col, 11);  Write ('+ ', RNum[2]);
    GotoXY (Col, 12);
    for I := 1 to 2 + L2 do Write ('-');
    Miss := 0;
    repeat
      GotoXY (Col, 13);  Readln (Ans);

      { -- Evaluate Answer }
      if Ans = RNum[3] then begin
        Inc(Right);  Miss := 0;  end
      else  { -- Incorrect answer }
        if Miss > 0 then begin  { -- Second Miss }
          Miss := 0;  Sound (400); Delay (200);  NoSound;
          Inc(Wrong);  Wr[Wrong] := Ans;
          Ri[Wrong] := RNum[3];  RiA[Wrong] := Arabic;
          end
        else begin  { -- First Miss }
          Miss := 1;  Sound (400);  Delay (200);  NoSound;
          GotoXY (Col, 16);  Write (Arabic);
          GotoXY (Col, 13);  ClrEol;
        end;
    until Miss = 0;
  end;  { -- for Prob }

  { -- Progress Report }
  ClrScr;  GotoXY (11,1);  Writeln ('PROGRESS REPORT');
  Writeln ('DATE: ', Dayte);
  Writeln ('NAME: ', Name);
  Writeln ('NUMBER CORRECT: ', Right);
  Writeln ('NUMBER OF EXERCISES: 3');
  Writeln ('PERCENT CORRECT: ', Round(RIGHT / 3 * 100), '%');
  Writeln;
  if Wrong > 0 then begin
    GotoXY (1, 15);
    Writeln ('WRONG ANSWER   CORRECT ANSWER  ARABIC');
    for I := 1 to Wrong do begin
      GotoXY (1, 16+I);  Write (Wr[I]);
      GotoXY (16, 16+I); Write (Ri[I]);
      GotoXY (32, 16+I); Write (RiA[I]);
    end;
    GotoXY (1, 23);  Writeln ('PRESS ANY KEY TO RETURN TO MENU.');
    repeat until KeyPressed;
  end;
end;

begin
  Write ('Enter name: ');  Readln (Name);
  Write ('Enter date: ');  Readln (Dayte);

  repeat
    ClrScr;
```

```
    Writeln ('1. INSTRUCTION PAGE');
    Writeln ('2. PRACTICE 3 PROBLEMS');
    Writeln ('3. QUIT');
    Readln (Option);
    if Option = 1 then  { -- Display instructions }
      begin
        ClrScr;
        Writeln ('YOU WILL BE GIVEN 3 PROBLEMS TO');
        Writeln ('WORK. A PROBLEM WILL CONSIST OF');
        Writeln ('ADDING TWO RANDOMLY GENERATED');
        Writeln ('ROMAN NUMERALS LESS THAN 20.');
        Writeln ('YOU WILL TYPE YOUR ANSWER IN');
        Writeln ('ROMAN NUMERALS AND PRESS ''RETURN.''');
        Writeln ('(PRESS ANY KEY TO RETURN TO MENU.)');
        repeat until KeyPressed;
      end
    else if Option = 2 then  { -- Practice 3 problems }
      Do3Problems;
  until Option = 3;
end.
```

```
{3.8}
program Thr8T87;
{ -- This program will determine the area shared w/2 rectangles. }
  var
    A, B, X, Y:       Array [1..4] of Integer;
    AB, XY:           Array [0..20, 0..20] of Integer;
    I, J, Width,
    Width2, Height:  Integer;

begin
  for I := 1 to 4 do begin
    Write ('Enter X,Y: ');  Readln (X[I], Y[I]);
    X[I] := Abs(X[I]);  Y[I] := Abs(Y[I]);
  end;
  Writeln;
  for I := 1 to 4 do begin
    Write ('Enter A,B: ');  Readln (A[I], B[I]);
    A[I] := Abs(A[I]);  B[I] := Abs(B[I]);
  end;

  { -- Initialize AB and XY arrays }
  for I := 0 to 20 do
    for J := 0 to 20 do begin
      AB[I,J] := 0;  XY[I,J] := 0;  end;

  { -- Store a 1 in each occupied square }
  for I := A[1] to A[2] do
    for J := B[4] to B[1] do
      AB[I, J] := 1;

  { -- Determine area in common (height-1 x Width-1) }
  Width := 0;  Height := 0;
  for I := X[1] to X[2] do begin
    for J := Y[4] to Y[1] do
      if (AB[I, J] = 1) then
        Inc(Width);
    if Width > 0 then begin
      Inc(Height);  Width2 := Width;  Width := 0;
    end;
  end;

  Writeln ((Height -1) * (Width2 -1));
end.
```

```
{3.9}
program Thr9T87;
{ -- This program will divide 2 big numbers w/at most 30 digits. }
  var
    ASt, BSt:            String[30];
    A, B:                Array[1..30] of Integer;
    Ch:                  Char;
    LenA, LenB, Quot:    Byte;
    I:                   Integer;
    LastAPos, LastAInd:  Byte;
    DigitsAdded:         Byte;
    AtLeast1Divide:      Boolean;

function ALessThanB: {giving} Boolean;
{ -- This function returns true if A[..] is less than B[..] }
  var
    I: Byte;

begin
  if LastAInd > LenB then
    ALessThanB := False
  else if LastAInd < LenB then
    ALessThanB := True
  else begin  { -- both A and B are same length }
    I := LenB;
    while (I > 1) and (A[I] = B[I]) do
      Dec(I);
    if A[I] < B[I] then  { -- Found position where A is < B }
      ALessThanB := True
    else
      ALessThanB := False;
  end;
end;

procedure AttachDigitToA;
{ -- This procedure will attach another digit at end of A[..] }
begin
  for I := LastAInd downto 1 do
    A[I+1] := A[I];
  if A[LastAInd+1] > 0 then
    Inc(LastAInd);
  Inc(LastAPos);
  Ch := ASt[LastAPos];
  A[1] := Ord(Ch) - Ord('0');
end;

procedure Sub_B_From_A;
{ -- This procedure will subtract B[..] from A[..] with borrowing}
  var
    Borrow: Byte;

begin
  for I := 1 to LenB do begin
    if B[I] <= A[I] then Borrow := 0
      else begin
```

```
        Borrow := 10;
        Dec(A[I+1]);
      end;
    A[I] := A[I] - B[I] + Borrow;
  end;
  { -- Find first non-zero of A[] for LastAInd }
  while (LastAInd > 1) and (A[LastAInd] = 0) do
    Dec(LastAInd);
end;

procedure DivideAbyB;
{ -- This procedure will divide A[..] by B[..] and display quot. }
begin
  Quot := 1;
  while not ALessThanB and (Quot < 10) do begin
    Sub_B_From_A;  Inc(Quot);
  end;
  Write (Quot - 1);
end;
      { -- Main program routine }
begin
  Write ('Enter first number: ');   Readln (ASt);
  Write ('Enter second number: ');  Readln (BSt);
  LenA := Length (ASt);  LenB := Length (BSt);  { -- LenA > LenB }

  { -- Store B number in Array: 456 becomes B[3]=6,B[2]=5,B[1]=4 }
  for I := LenB downto 1 do begin
    Ch := BSt[I];
    B[LenB-I+1] := Ord(Ch) - Ord('0');
  end;
  { -- Store equal number of digits in A as was in B }
  if LenB <= LenA then LastAPos := LenB
    else LastAPos := Length(ASt);
  for I := LastAPos downto 1 do begin
    Ch := ASt[I];
    A[LastAPos-I+1] := Ord(Ch) - Ord('0');
  end;
  LastAInd := LastAPos;
  if ALessThanB and (LastAPos < LenA) then
    { -- Attach 1 more digit so A > B }
    AttachDigitToA;
  AtLeast1Divide := False;

  { -- Perform systematic division by attaching digits
    -- until no more digits }
  while (LastAPos < LenA) or not ALessThanB do begin
    DigitsAdded := 0;
    while ALessThanB and (LastAPos < LenA) do begin
      AttachDigitToA;  Inc(DigitsAdded);
    end;
    for I := 1 to DigitsAdded-1 do
      { -- Print 0's for each excessive digit }
      Write ('0');
    DivideAbyB;
    AtLeast1Divide := True;
```

```
  end;  { -- while }

  { -- Display Remainder }
  if not AtLeast1Divide then Write ('0');  { -- No quotient, A<B }
  Write (' REMAINDER ');
  for I := LastAInd downto 1 do
    Write (A[I]);
end.



{3.10}
program Thr10T87;
{ -- This program will generate random mazes with 8 x 5 paths. }
uses Crt;
  var
    { -- A = Forbidden segments,  PointUsed = Existing points }
    A:         Array [0..33, 0..33] of  Byte;
    PointUsed: Array [0..33, 0..33] of  Byte;
    L, W, Linc, Winc, Lnum, Wnum:       Byte;
    NumOfLines, D, X, Y, X2, Y2, I, J:  Byte;
    LinesDrawn, NumofTries:             Byte;
    SegmentDrawn:                       Boolean;

begin
  ClrScr;  Randomize;  L := 8;  W := 5;
  NumOfLines := (L-1) * (W-1);
  LinesDrawn := 0;
  Linc := 32 div L;  Winc := 15 div W;
  Lnum := L;  Wnum := W;
  for I := 0 to 33 do
    for J := 0 to 33 do begin
      PointUsed[I, J] := 0;  A[I, J] := 0;
    end;

  { -- Draw perimeter }
  for I := 1 to 33 do begin
    Write ('*');  PointUsed[I-1, 0] := 1;
  end;
  for I := 1 to 14 do begin
    GotoXY (1, I+1);  Write ('*');  PointUsed[0, I] := 1;
    GotoXY (33,I+1);  Write ('*');  PointUsed[L, I] := 1;
  end;  Writeln;
  for I := 1 to 33 do begin
    Write ('*');  PointUsed[I-1, W] := 1;
  end;

  A[0, 0] := 1;  A[Lnum, 0] := 1;  A[Lnum, Wnum] := 1;
  A[0, Wnum] := 1;
  repeat
    { -- Get point that exist but is not forbidden }
    repeat
      X := Random(Lnum*2) - (Lnum div 2);
      Y := Random(Wnum*2) - (Wnum div 2);
```

```
    if X < 0 then X := 0;
    if X > Lnum then X := Lnum;
    if Y < 0 then Y := 0;
    if Y > Wnum then Y := Wnum;
  until (PointUsed[X, Y] = 1) and (A[X, Y] = 0);

  repeat
    D := Random(4);  { -- Random direction }

    SegmentDrawn := False;  NumOfTries := 0;
    repeat
      NumOfTries := NumOfTries + 1;
      Inc(D);  If D > 4 then D := D - 4;
      Case D of
        1: begin  { -- Up }
             if (Y > 0) and not (PointUsed[X, Y-1] = 1) then
             begin
               for J := 0 to Winc - 1 do begin
                 GotoXY (X*Linc+1, Y*Winc-J);  Write ('*');
               end;
               X2 := X;  Y2 := Y - 1;
               SegmentDrawn := True;
             end;
           end;

        2: begin  { -- Right }
             if (X < LNum) and not (PointUsed[X+1, Y] = 1) then
             begin
               for J := 0 to Linc - 1 do begin
                 GotoXY (X*Linc+2+J, Y*Winc+1);  Write ('*');
               end;
               X2 := X + 1;  Y2 := Y;
               SegmentDrawn := True;
             end;
           end;

        3: begin  { -- Down }
             if (Y < Wnum) and not (PointUsed[X, Y+1] = 1) then
             begin
               for J := 0 to Winc - 1 do begin
                 GotoXY (X*Linc+1, Y*Winc+2+J);  Write ('*');
               end;
               X2 := X;  Y2 := Y + 1;
               SegmentDrawn := True;
             end;
           end;

        4: begin  { -- Left }
             if (X > 0) and not (PointUsed[X-1, Y] = 1) then
             begin
               for J := 0 to Linc - 1 do begin
                 GotoXY (X*Linc-J, Y*Winc+1);  Write ('*');
               end;
               X2 := X - 1;  Y2 := Y;
               SegmentDrawn := True;
```

```
              end;
            end;
        end;  { -- case }
      until SegmentDrawn or (NumofTries = 4);

      if SegmentDrawn then begin
        PointUsed[X2, Y2] := 1;
        Inc(LinesDrawn);
        X := X2;  Y := Y2;
        end
      else  { -- No more segments can be drawn from this point }
        A[X, Y] := 1;
    until (LinesDrawn = NumOfLines) or not SegmentDrawn;
  until (LinesDrawn = NumOfLines);  { -- Get new point of
                                       -- Segment not drawn }
  { -- Open doors }
  X := Random(Wnum) + 1;  Y := Random (Wnum) + 1;
  for J := 0 to Winc - 2 do begin
    GotoXY (1,  X * Winc - J);  Write (' ');
    GotoXY (33, Y * Winc - J);  Write (' ');
  end;
  GotoXY (1, 23);

end.
```