INTERNATIONAL ATOMIC ENERGY AGENCY
UNITED NATIONS EDUCATIONAL, SCIENTIFIC AND CULTURAL ORGANIZATION

INTERNATIONAL CENTRE FOR THEORETICAL PHYSICS
34100 TRIESTE (ITALY) - P.O.B. 586 - MIRAMARE - STRADA COSTIERA 11 - TELEPHONE: 2240-1
CABLE: CENTRATOM - TELEX 460392 - 1

SECOND SCHOOL ON ADVANCED TECHNIQUES
IN COMPUTATIONAL PHYSICS
(18 January - 12 February 1988)

SMR.282/27

MODULA-2/EXERCISES AND SOLUTIONS

V.B.A.Fack
University of Ghent, Belgium

MODULA–2 EXERCISES and SOLUTIONS

1. Write a program that reads a cardinal number and determines whether it is a prime number or not, by trying to divide it by $2, 3, 5, 7, 9, \ldots$

```
MODULE Exercise1;
FROM InOut IMPORT
  ReadCard, WriteCard, WriteString, WriteLn;
VAR Num, Div : CARDINAL;
BEGIN
  LOOP
    WriteString ("Give a cardinal number : ");
    ReadCard (Num); WriteLn;
    IF Num MOD 2 = 0 THEN
      WriteCard (Num, 5);
      WriteString (" is not a prime"); WriteLn
    ELSE
      Div := 3;
      LOOP
        IF Num MOD Div = 0 THEN
          WriteCard (Num, 5);
          WriteString (" is not a prime"); WriteLn;
          EXIT
        END (* if *);
        INC (Div, 2);
        IF Div * Div > Num THEN
          WriteCard (Num, 5);
          WriteString (" is a prime"); WriteLn;
          EXIT
        END (* if *)
      END (* loop *)
    END (* if *)
  END (* loop *)
END Exercise1.
```

2. A simple coding system is based on the placement of letters and digits in 6×6 square, depending on a keyword. For the keyword `ictp15`, the coding square is:

i	c	t	p	1	5
a	b	d	e	f	g
h	j	k	l	m	n
o	q	r	s	u	v
w	x	y	z	0	2
3	4	6	7	8	9

A given string is coded by splitting it up in consecutive pairs and coding each pair by replacing each of its characters by the character on the same row and on the column of the other one. If the string to be coded has an odd number of characters, the last one remains the same.

For example: `br-ai-nd-am-ag-e` is coded to `dq-ai-kg-fh-ga-e`.

Note that decoding the string again can be done in the same way.

Write a program that reads a keyword, then codes a sequence of strings to be read from the keyboard, outputs the coded strings and decodes them again as a check.

Try to find a *good* representation of the coding square, i.e. one that allows to obtain the position of a given character easily. (It is *NOT* the 6×6 square!)

```
MODULE Exercise2;

FROM InOut IMPORT
  ReadString, WriteCard, Write, WriteString, WriteLn;
FROM Strings IMPORT
  Length;
CONST
  MaxSize = 6;
  MaxLen  = 79;
  EoStr   = 0C;
TYPE
  Positions = RECORD Row, Col : CARDINAL END;
VAR
  CodeSquare    : ARRAY [1..MaxSize], [1..MaxSize] OF CHAR;
  PosList       : ARRAY ['0'..'z'] OF Positions;
  String, Code  : ARRAY [0..79] OF CHAR;
  High          : CARDINAL;
```

```
PROCEDURE Init;
  (* Constructing the coding square
     and the list of positions       *)
  VAR
    key                   : ARRAY [1..MaxSize] OF CHAR;
    ch                    : CHAR;
    i, j, curRow, curCol : CARDINAL;
  BEGIN
    FOR ch := '0' TO 'z' DO
      WITH PosList[ch] DO Row := 0; Col := 0 END
    END (* for *);
    WriteString ("Give a keyword of 6 letters/digits : ");
    ReadString (key); WriteLn;
    WriteLn;
    FOR i := 1 TO MaxSize DO
      CodeSquare[1,i] := key[i];
      WITH PosList[key[i]] DO
        Row := 1; Col := i
      END (* with *)
    END (* for *);
    curRow := 2; curCol := 1;
    FOR ch := 'a' TO 'z' DO
      WITH PosList[ch] DO
        IF Row = 0 THEN
          CodeSquare[curRow,curCol] := ch;
          Row := curRow; Col := curCol;
          IF curCol = MaxSize THEN
            INC (curRow); curCol := 1
          ELSE
            INC (curCol)
          END (* if *)
        END (* if *);
      END (* with *)
    END (* for *);
    FOR ch := '0' TO '9' DO
      WITH PosList[ch] DO
        IF Row = 0 THEN
          CodeSquare[curRow,curCol] := ch;
          Row := curRow; Col := curCol;
          IF curCol = MaxSize THEN
            INC (curRow); curCol := 1
          ELSE
            INC (curCol)
          END (* if *)
```

```
          END (* if *);
        END (* with *)
      END (* for *)
    END Init;

  PROCEDURE EDCode (high : CARDINAL;
                    VAR str, code : ARRAY OF CHAR);
    (* Coding/decoding a string *)
    VAR
      len, i1, i2, row1, row2, col1, col2 : CARDINAL;
    BEGIN
      i1:= 0; i2 := 1;
      WHILE i1 < high DO
        WITH PosList[str[i1]] DO
          row1 := Row; col1 := Col
        END (* with *);
        WITH PosList[str[i2]] DO
          row2 := Row; col2 := Col
        END (* with *);
        code[i1] := CodeSquare[row1,col2];
        code[i2] := CodeSquare[row2,col1];
        INC (i1, 2); INC (i2, 2)
      END (* while *);
      IF i1 = high THEN
        code[i1] := str[i1]; code[i2] := EoStr
      ELSE
        code[i1] := EoStr
      END (* if *)
    END EDCode;

BEGIN
  Init;
  LOOP
    WriteString ("String to code : ");
    ReadString (String); WriteLn;
    High := Length (String) - 1;
    EDCode (High, String, Code);
    WriteString (" --> "); WriteString (Code);
    EDCode (High, Code, String);
    WriteString (" --> "); WriteString (String);
    WriteLn; WriteLn
  END (* loop *)
END Exercise2.
```

3. Write a non-recursive and a recursive function to compute the factorial of a cardinal number.

```
MODULE Exercise3;
FROM InOut IMPORT
  ReadCard, WriteCard, WriteString, WriteLn;
PROCEDURE NRFact (n : CARDINAL) : CARDINAL;
  VAR fact, i : CARDINAL;
  BEGIN
    fact := 1;
    FOR i := 1 TO n DO fact := fact * i END;
    RETURN fact
  END NRFact;
PROCEDURE RFact (n : CARDINAL) : CARDINAL;
  BEGIN
    IF n = 0 THEN
      RETURN 1
    ELSE
      RETURN n * RFact (n-1)
    END (* if *)
  END RFact;
VAR Num : CARDINAL;
BEGIN
  WriteString ("Give a cardinal number : ");
  ReadCard (Num); WriteLn;
  WriteString ("Recursive : n! = ");
  WriteCard (RFact(Num), 6); WriteLn;
  WriteString ("Non-recursive : n! =");
  WriteCard (NRFact (Num), 6); WriteLn;
END Exercise3.
```

4. For given n and m, one can generate all n-tuples $(a_1, a_2, \ldots, a_n)$, $a_i \in \mathbf{N}, \forall i = 1, \ldots, n$ that satisfy the condition

$$a_1 + 2a_2 + 3a_3 + \cdots + na_n = m$$

For example: For $n = m = 5$, one obtains the following 5-tuples:

$$(0,0,0,0,1)$$
$$(1,0,0,1,0)$$
$$(0,1,1,0,0)$$
$$(2,0,1,0,0)$$
$$(1,2,0,0,0)$$
$$(3,1,0,0,0)$$
$$(5,0,0,0,0)$$

One easily sees that:

- $ia_i \leq m, \forall i = 1, \ldots, n$, which gives an upper limit for a_i.
- $a_1 + 2a_2 + \cdots + (n-1)a_{n-1} = m - na_n$, which suggests a recursive solution.

Write a program that reads n and m, where $1 \leq n, m \leq 20$, and displays all possible n-tuples.

```
MODULE Exercise4;

FROM InOut IMPORT
  ReadCard, WriteCard, WriteString, WriteLn;

CONST
  MaxSize = 20;

VAR
  Tuple      : ARRAY [1..MaxSize] OF CARDINAL;
  Size, Sum  : CARDINAL;
```

```
PROCEDURE TryTuples (size, sum : CARDINAL);
  VAR
    try : CARDINAL;
  BEGIN
    IF size = 0 THEN
      IF sum = 0 THEN
        FOR try := 1 TO Size DO
          WriteCard (Tuple[try], 3)
        END (* for *);
        WriteLn
      END (* if *)
    ELSE
      try := 0;
      WHILE sum >= size * try DO
        Tuple[size] := try;
        TryTuples (size-1, sum-size*try);
        INC (try)
      END (* for *)
    END (* if *)
  END TryTuples;

BEGIN
  WriteString ("n, m ? ");
  ReadCard (Size); WriteString ("   ");
  ReadCard (Sum); WriteLn;
  TryTuples (Size, Sum)
END Exercise4.
```

5. Write a DEFINITION MODULE and IMPLEMENTATION MODULE to implement vector arithmetic. This includes:

   - the definition of a vector type,
   - the definition and implementation of basic vector operations such as adding 2 vectors, computing the dot product of 2 vectors, computing the length of a vector.
   - the definition and implementation of vector I/O procedures

   Write also a testing module for this module.

```
DEFINITION MODULE VectArith;

EXPORT QUALIFIED
  MaxSize, Vector,
  AddVect, DotProdVect, ReadVect, WriteVect;

CONST
  MaxSize = 20;

TYPE
  Vector = RECORD
             Size : CARDINAL;
             Comp : ARRAY [1..MaxSize] OF REAL
           END (* record *);

PROCEDURE AddVect (v1, v2 : Vector; VAR res : Vector);

PROCEDURE DotProdVect (v1, v2 : Vector) : REAL;

PROCEDURE ReadVect (size : CARDINAL; VAR v : Vector);

PROCEDURE WriteVect (v : Vector; nrdigits : CARDINAL);

END VectArith.
```

```
IMPLEMENTATION MODULE VectArith;

FROM InOut IMPORT
  WriteLn, WriteString;
FROM RealInOut IMPORT
  ReadReal, WriteReal;
```

```
PROCEDURE AddVect (v1, v2 : Vector; VAR res : Vector);
  VAR
    i : CARDINAL;
  BEGIN
    WITH res DO
      Size := v1.Size;
      FOR i := 1 TO Size DO
        Comp[i] := v1.Comp[i] + v2.Comp[i]
      END (* for *)
    END (* with *)
  END AddVect;

PROCEDURE DotProdVect (v1, v2 : Vector) : REAL;
  VAR
    i   : CARDINAL;
    res : REAL;
  BEGIN
    res := 0.0;
    WITH v1 DO
      FOR i := 1 TO Size DO
        res := res + Comp[i] * v2.Comp[i]
      END (* for *)
    END (* with *);
    RETURN res
  END DotProdVect;

PROCEDURE ReadVect (size : CARDINAL; VAR v : Vector);
  VAR
    i : CARDINAL;
  BEGIN
    WITH v DO
      Size := size;
      WriteString ('( ');
      ReadReal (Comp[1]);
      FOR i := 2 TO size DO
        WriteString (' , '); ReadReal (Comp[i])
      END (* for *);
      WriteString (' )'); WriteLn
    END (* with *)
  END ReadVect;

PROCEDURE WriteVect (v : Vector; nrdigits : CARDINAL);
  VAR
```

```
    i : CARDINAL;
  BEGIN
    WITH v DO
      WriteString ('( ');
      WriteReal (Comp[1], nrdigits);
      FOR i := 2 TO Size DO
        WriteString (' , ');
        WriteReal (Comp[i], nrdigits);
      END (* for *);
      WriteString (' )'); WriteLn
    END (* with *)
  END WriteVect;

BEGIN
END VectArith.
```

```
MODULE VectArithTst;

FROM VectArith IMPORT
  Vector, AddVect, DotProdVect, ReadVect, WriteVect;

FROM InOut IMPORT
  ReadCard, WriteString, WriteLn;
FROM RealInOut IMPORT
  WriteReal;

VAR
  Size : CARDINAL;
  Vect1, Vect2, Res : Vector;

BEGIN
  WriteString ("Size ? "); ReadCard (Size); WriteLn;
  WriteString ("V1 = "); ReadVect (Size, Vect1); WriteLn;
  WriteString ("V2 = "); ReadVect (Size, Vect2); WriteLn;

  WriteString ("V1 + V2 = ");
  AddVect (Vect1, Vect2, Res);
  WriteVect (Res, 10); WriteLn;
  WriteString ("V1 . V2 = ");
  WriteReal(DotProdVect (Vect1, Vect2), 10); WriteLn;
END VectArithTst.
```