Notations:
LOAD(i) = ADDR(LL ON of i), LOAD

```
%  Assignment 4 Sample Program 1          % 1-1
{                                         % 1-2
   var i, j, k, n, m : integer            % 1-3

   var p, q, r, s, t : boolean            % 1-4
   var A[7] , B[ -100 .. 50 ] : integer   % 1-5
   var C[ -7 .. -3 ], D[ 400 ]    : boolean    % 1-6
   PUSHMT
   SETD 0
   PUSH UNDEFINED
   PUSH main_needed_words
   DUPN
   func f(i : integer, j : integer) : integer    % 1-7
         {  if i > 0 then                        % 1-8
            LOAD(i)
            PUSH(0)
            SWAP
            LT
             PUSH(1-10)
            BF

                  result f( i - 1 , j + 1 )    % 1-9
                  PUSH (1-15) // exit if

                  ADDR LL 0
                  PUSH 3
                  SUB //push the address of the return value, which is the display base address - 3

                  // call function f
                  PUSH UNDEFINED   // return value, to be filled in,
                  PUSH(result_case)
                  ADDR LL 0              // saved display reference

                  // display update
                  PUSHMT
                  SETD LL                // LL of function / procedure

                  ADDR (LL ON of i)
                  LOAD
                  PUSH(1)
```

```
        SUB
        ADDR (LL ON of j)
        LOAD
        PUSH(1)
        ADD
        PUSH(1-7) // function call F
        BR

        result_case:
        // done  evaluating result expression
        STORE

        PUSH num_params + num_local_words
        POPN
        SETD LL                 // LL of function / procedure
        BR

        BR // exit if

else if i < 0 then                  % 1-10
LOAD(i)
PUSH(0)
LT
PUSH(1-13)
BF
        result f( i + 1 , j - 1 )    % 1-11
        PUSH (1-15) // exit if

        ADDR LL 0
        PUSH 3
        SUB //push the address of the return value, which is the display base address - 3

        // call function f
        PUSH UNDEFINED  // return value, to be filled in,
        PUSH(result_case)
        ADDR LL 0               // saved display reference

        // display update
        PUSHMT
        SETD LL                 // LL of function / procedure

        ADDR (LL ON of i)
        LOAD
```

```
                PUSH(1)
                ADD
                ADDR (LL ON of j)
                LOAD
                PUSH(1)
                SUB
                PUSH(1-7)
                BR

                result_case:
                // done  evaluating result expression
                STORE

                PUSH num_params + num_local_words
                POPN
                SETD LL                 // LL of function / procedure
                BR

                BR // skip else after done all the statements in then part
        else                            % 1-12
                result j                        % 1-13

                ADDR LL 0
                PUSH 3
                SUB // push the address of the return value

                ADDR (LL ON of j)
                LOAD

                STORE

                PUSH num_params + num_local_words
                POPN
                SETD LL                 // LL of function / procedure
                BR


        fi fi                           % 1-14
        }                                       % 1-15
n := j * (k-1) / ( n + 2 )              % 1-16
ADDR (LL ON of n)
LOAD(j)
LOAD(k)
```

```
PUSH(1)
SUB //k-1
MUL //j*(k-1)
LOAD(n)
PUSH(2)
ADD //n+2
DIV //j*(k-1)/(n+2)
STORE
r := (not q and p) or (q and not p)   % 1-17
ADDR (LL ON of r)
LOAD(q)
PUSH(1)
SUB
NEG // not q
LOAD(p)
MUL
LOAD(q)
LOAD(p)
PUSH(1)
SUB
NEG
MUL
OR
STORE
p := i < j or  k <= n              % 1-18
ADDR (LL ON of p)
LOAD(i)
LOAD(j)
LT
LOAD(k)
LOAD(n)
SWAP
LT
PUSH(1)
SUB
NEG
OR
STORE
r := j = n and k not = m                  % 1-19
ADDR (LL ON of r)
LOAD(j)
LOAD(n)
EQ
```

```
LOAD(k)
LOAD(m)
EQ
PUSH(1)
SUB
NEG
MUL
STORE
t := ( j > k     ? r = s : i not= j )          % 1-20
ADDR (LL ON of t)
LOAD(j)
LOAD(k)
SWAP
LT
PUSH(addr of false_case)
BF
LOAD(r)
LOAD(s)
EQ
PUSH(sign_t)
BR

false_case:
LOAD(i)
LOAD(j)
EQ
PUSH(1)
SUB
NEG

sign_t:
STORE
A[ i+ f(-4,A[n+3]) ] := 5                    % 1-21
ADDR (LL ON of A)
LOAD(i)

//function call for f(-4,A[n+3])

PUSH UNDEFINED// return value, to be filled in,
PUSH(after_function_call)
ADDR LL 0            // saved display reference

// display update
```

```
PUSHMT
SETD LL                 // LL of function / procedure

PUSH(-4)
ADDR (LL ON of A)
ADDR (LL ON of n)
LOAD
PUSH(3)
ADD
PUSH(lower bound of A)
SUB // calculate the correct offset
ADD
LOAD
PUSH(1-7)
BR

after_function_call:
ADD
PUSH(lower bound of A)
SUB // calculate the correct offset
ADD // add array offset of A
PUSH(5)
STORE
B[B[B[i+1]]] := A[f( 17 , 5 )]              % 1-22
ADDR (LL ON of B)
ADDR (LL ON of B)
ADDR (LL ON of B)
ADDR (LL ON of i)
LOAD
PUSH(1)
ADD // i+1
PUSH(lower bound of B)
SUB // calculate the correct offset
ADD
LOAD // B[i+1]
PUSH(lower bound of B)
SUB // calculate the correct offset
ADD
LOAD // B[B[i+1]]
PUSH(lower bound of B)
SUB // calculate the correct offset
ADD // B[B[B[i+1]]]
```

```
ADDR (LL ON of A)
//function call for f(17,5)
PUSH UNDEFINED// return value, to be filled in,
PUSH(after_function_call)
ADDR LL 0            // saved display reference

// display update
PUSHMT
SETD LL              // LL of function / procedure

PUSH(17)
PUSH(15)
PUSH(1-7)
BR

after_function_call:
PUSH(lower bound of A)
SUB // calculate the correct offset
ADD
LOAD
STORE
C[-4] := p or q or j >= f(k,7)          % 1-23
ADDR (LL ON of C)
PUSH(-4)
PUSH(lower bound of C)
SUB // calculate the correct offset
ADD
ADDR (LL ON of p)
LOAD
ADDR (LL ON of q)
LOAD
OR
ADDR (LL ON of j)
LOAD
//function_call for f(k,7)
PUSH UNDEFINED// return value, to be filled in,
PUSH(after_function_call)
ADDR LL 0            // saved display reference

// display update
PUSHMT
SETD LL              // LL of function / procedure
```

```
ADDR (LL ON of k)
LOAD
PUSH(7)
PUSH(1-7)
BR

after_function_call:
LT
PUSH(1)
SUB
NEG
OR
STORE
{                                       % 1-24
        var E[ 10 , -4 .. 5 ] : integer    % 1-25
        var B[ -2 .. 4 , 7  ] : boolean    % 1-26
        E[ i -1 , A[k+1] ] := B[ m - 2]             % 1-27
         ADDR (LL ON of E)
         ADDR (LL ON of i)
         LOAD
         PUSH(1)
         SUB
        PUSH(1D lower bound of E)
        SUB // calculate the correct offset
         PUSH(10) // length of inner array
         MUL
        ADDR (LL ON of A)
         ADDR (LL ON of k)
         LOAD
         PUSH(1)
        ADD
        PUSH(lower bound of A)
        SUB // calculate the correct offset
         ADD
         LOAD
        PUSH(2D lower bound of E)
        SUB // calculate the correct offset
        ADD // index1*length_inner_array+index2
        ADD // base_array_address+(index1*length_inner_array+index2)
        ADDR (LL ON of B)
        ADDR (LL ON of m)
        LOAD
        PUSH(2)
```

```
        SUB
        PUSH(lower bound of B)
        SUB // calculate the correct offset
        ADD
        LOAD
        STORE
        C[ j - 3 ] := B[f(n,m) , n + m ]            % 1-28
           or f(i,j+1) <= f(j,i+1) and              % 1-29

                f(A[i],E[i-1,k+1]) >= 7             % 1-30
  }                                                 % 1-31
}                                                   % 1-32


        ADDR (LL ON of C)
        ADDR (LL ON of j)
        LOAD
        PUSH(3)
        SUB
        PUSH(lower bound of C)
        SUB // calculate the correct offset
        ADD

        //B[f(n,m),n+m]
        ADDR (LL ON of B)

        //function_call f(n,m)
        PUSH UNDEFINED   // return value, to be filled in,
        PUSH(after_function_call_1)
        ADDR LL 0              // saved display reference

        // display update
        PUSHMT
        SETD LL                // LL of function / procedure

        ADDR (LL ON of n)
        LOAD
        ADDR (LL ON of m)
        LOAD
        PUSH(1-7)
        BR

        after_function_call_1:
```

```
PUSH(1D lower bound of B)
SUB // calculate the correct offset
PUSH(7) // size of inner array
MUL
ADDR (LL ON of n)
LOAD
ADDR (LL ON of m)
LOAD
ADD //n+m
PUSH(2D lower bound of B)
SUB // calculate the correct offset
ADD // f(n,m) * 7 + n + m
ADD // offset of B
LOAD

//f(i,j+1)<=f(j,i+1)
//function_call for f(i,j+1)
PUSH UNDEFINED   // return value, to be filled in,
PUSH(after_function_call_2)
ADDR LL 0              // saved display reference

// display update
PUSHMT
SETD LL                // LL of function / procedure

ADDR (LL ON of i)
LOAD
ADDR (LL ON of j)
LOAD
PUSH(1)
ADD
PUSH(1-7)
BR

after_funciton_call_2:
//function_call for f(i,j+1)
PUSH UNDEFINED   // return value, to be filled in,
PUSH(after_function_call_3)
ADDR LL 0              // saved display reference

// display update
PUSHMT
SETD LL                // LL of function / procedure
```

```
ADDR (LL ON of i)
LOAD
ADDR (LL ON of j)
LOAD
PUSH(1)
ADD
PUSH(1-7)
BR

after_function_call_3:
SWAP
LT
PUSH(1)
SUB
NEG

//f(A[i],E[i-1,k+1])>=7
PUSH UNDEFINED   // return value, to be filled in,
PUSH(after_function_call_4)
ADDR LL 0              // saved display reference

// display update
PUSHMT
SETD LL                // LL of function / procedure

ADDR (LL ON of A)
//A[i]
ADDR (LL ON of i)
LOAD
PUSH(lower bound of A)
SUB // calculate the correct offset
ADD
LOAD
//E[i-1,k+1]
ADDR (LL ON of E)
ADDR (LL ON of i)
LOAD
PUSH(1)
SUB
PUSH(1D lower bound of E)
SUB // calculate the correct offset
PUSH(10)
```

```
MUL
ADDR (LL ON of k)
LOAD
PUSH(1)
ADD // k+1
PUSH(2D lower bound of E)
SUB // calculate the correct offset
ADD // (i-1)*10+k+1
ADD //addr_E+(i-1)*10+k+1
LOAD
PUSH(1-7)
BR
after_function_call_4:
PUSH(7)
LT
PUSH(1)
SUB
NEG

MUL // f(i,j+1) <= f(j,i+1) and f(A[i],E[i-1,k+1]) >= 7

OR // B[f(n,m) , n + m ] or (f(i,j+1) <= f(j,i+1) and f(A[i],E[i-1,k+1]) >= 7)

STORE
```