```
TITLE      Linear search of integer array      LIN_SRCH.ASM
COMMENT |
        Objective: To implement linear search of an integer
                   array; demonstrates the use of loopne.
            Input: Requests numbers to fill array and a
                   number to be searched for from user.
           Output: Displays the position of the number in
                   the array if found; otherwise, not found
|                  message.
.MODEL SMALL
.STACK 100H
.DATA
MAX_SIZE       EQU 100
array          DW  MAX_SIZE DUP (?)
input_prompt   DB  'Please enter input array: '
               DB  '(negative number terminates input)',0
query_number   DB  'Enter the number to be searched: ',0
out_msg        DB  'The number is at position ',0
not_found_msg  DB  'Number not in the array!',0
query_msg      DB  'Do you want to quit (Y/N): ',0

.CODE
INCLUDE io.mac
```

Selection: 1

```
main    PROC
        .STARTUP
        PutStr  input_prompt ; request input array
        mov     BX,OFFSET array
        mov     CX,MAX_SIZE
array_loop:
        GetInt  AX           ; read an array number
        nwln
        cmp     AX,0         ; negative number?
        jl      exit_loop    ; if so, stop reading numbers
        mov     [BX],AX      ; otherwise, copy into array
        inc     BX           ; increment array address
        inc     BX
        loop    array_loop   ; iterates a maximum of MAX_SIZE
exit_loop:
        mov     DX,BX        ; DX keeps the actual array size
        sub     DX,OFFSET array  ; DX := array size in bytes
        sar     DX,1         ; divide by 2 to get array size
read_input:
        PutStr  query_number ; request number to be searched for
        GetInt  AX           ; read the number
        nwln
```

Selection: 2

```
        push    AX           ; push number, size & array pointer
        push    DX
        push    OFFSET array
        call    linear_search
        ; linear_search returns in AX the position of the number
        ; in the array; if not found, it returns 0.
        cmp     AX,0         ; number found?
        je      not_found    ; if not, display number not found
        PutStr  out_msg      ; else, display number position
        PutInt  AX
        jmp     SHORT user_query
not_found:
        PutStr  not_found_msg
user_query:
        nwln
        PutStr  query_msg    ; query user whether to terminate
        GetCh   AL           ; read response
        nwln
        cmp     AL,'Y'       ; if response is not 'Y'
        jne     read_input   ; repeat the loop
done:                        ; otherwise, terminate program
        .EXIT
main    ENDP
```

Selection: 3

```
;------------------------------------------------------------
; This procedure receives a pointer to an array of integers,
; the array size, and a number to be searched via the stack.
; If found, it returns in AX the position of the number in
; the array; otherwise, returns 0.
; All registers, except AX, are preserved.
;------------------------------------------------------------
linear_search  PROC
       push    BP
       mov     BP,SP
       push    BX             ; save registers
       push    CX
       mov     BX,[BP+4]      ; copy array pointer
       mov     CX,[BP+6]      ; copy array size
       mov     AX,[BP+8]      ; copy number to be searched
       sub     BX,2           ; adjust index to enter loop
search_loop:
       add     BX,2           ; update array index
       cmp     AX,[BX]        ; compare the numbers
       loopne  search_loop
       mov     AX,0           ; set return value to zero
       jne     number_not_found  ; modify it if number found
       mov     AX,[BP+6]      ; copy array size
       sub     AX,CX          ; compute array index of number
number_not_found:
```

Selection: 4

```
        pop     CX            ; restore registers
        pop     BX
        pop     BP
        ret     6
linear_search  ENDP
        END   main
```

```
TITLE    Sorting an array by selection sort    SEL_SORT.ASM
COMMENT |
        Objective: To sort an integer array using selection sort.
            Input: Requests numbers to fill array.
|          Output: Displays sorted array.
.MODEL SMALL
.STACK 100H
.DATA
MAX_SIZE       EQU 100
array          DW  MAX_SIZE DUP (?)
input_prompt   DB  'Please enter input array: '
               DB  '(negative number terminates input)',0
out_msg        DB  'The sorted array is:',0

.CODE
.486
INCLUDE io.mac
main    PROC
        .STARTUP
        PutStr  input_prompt ; request input array
        mov     BX,OFFSET array
        mov     CX,MAX_SIZE
array_loop:
        GetInt  AX           ; read an array number
        nwln
        cmp     AX,0         ; negative number?
        jl      exit_loop    ; if so, stop reading numbers
        mov     [BX],AX      ; otherwise, copy into array
        add     BX,2         ; increment array address
        loop    array_loop   ; iterates a maximum of MAX_SIZE
exit_loop:
        mov     DX,BX        ; DX keeps the actual array size
        sub     DX,OFFSET array  ; DX := array size in bytes
        sar     DX,1         ; divide by 2 to get array size
        push    DX           ; push array size & array pointer
        push    OFFSET array
        call    selection_sort
        PutStr  out_msg      ; display sorted array
        nwln
        mov     CX,DX
        mov     BX,OFFSET array
display_loop:
        PutInt  [BX]
        nwln
        add     BX,2
        loop    display_loop
done:
        .EXIT
main    ENDP
```

```
;-----------------------------------------------------------
; This procedure receives a pointer to an array of integers
; and the array size via the stack. The array is sorted by
; using the selection sort. All registers are preserved.
;-----------------------------------------------------------
SORT_ARRAY  EQU  [BX]
selection_sort PROC
       pusha                  ; save registers
       mov     BP,SP
       mov     BX,[BP+18]     ; copy array pointer
       mov     CX,[BP+20]     ; copy array size
       sub     SI,SI          ; array left of SI is sorted
sort_outer_loop:
       mov     DI,SI
       ; DX is used to maintain the minimum value and AX
       ; stores the pointer to the minimum value
       mov     DX,SORT_ARRAY[SI]  ; min. value is in DX
       mov     AX,SI          ; AX := pointer to min. value
       push    CX
       dec     CX             ; size of array left of SI
sort_inner_loop:
```

```
sort_inner_loop:
       add     DI,2          ; move to next element
       cmp     DX,SORT_ARRAY[DI] ; less than min. value?
       jle     skip1         ; if not, no change to min. value
       mov     DX,SORT_ARRAY[DI] ; else, update min. value (DX)
       mov     AX,DI             ;       & its pointer (AX)
skip1:
       loop    sort_inner_loop
       pop     CX
       cmp     AX,SI         ; AX = SI?
       je      skip2         ; if so, element at SI is its place
       mov     DI,AX         ; otherwise, exchange
       mov     AX,SORT_ARRAY[SI]  ; exchange min. value
       xchg    AX,SORT_ARRAY[DI]  ; & element at SI
       mov     SORT_ARRAY[SI],AX
skip2:
       add     SI,2          ; move SI to next element
       dec     CX
       cmp     CX,1          ; if CX = 1, we are done
       jne     sort_outer_loop
       popa                  ; restore registers
       ret     4
selection_sort ENDP
       END   main
```

Selection: 9

```
TITLE    Sample indirect jump example    IJUMP.ASM
COMMENT |
        Objective: To demonstrate the use of indirect jump.
            Input: Requests a digit character from the user.
                   WARNING: Typing any other character may
                            crash the system!
|          Output: Appropriate class selection message.
.MODEL SMALL
.STACK  100H
.DATA
jump_table  DW  code_for_0   ; indirect jump pointer table
            DW  code_for_1
            DW  code_for_2
            DW  default_code ; default code for digits 3-9
            DW  default_code
            DW  default_code
            DW  default_code
            DW  default_code
            DW  default_code
            DW  default_code

```

Selection: 10

```
prompt_msg  DB  'Type a character (digits ONLY): ',0
msg_0       DB  'Economy class selected.',0
msg_1       DB  'Business class selected.',0
msg_2       DB  'First class selected.',0
msg_default DB  'Not a valid code!',0
.CODE
INCLUDE  io.mac
main PROC
        .STARTUP
read_again:
        PutStr prompt_msg    ; request a digit
        sub    AX,AX         ; AX := 0
        GetCh  AL            ; read input digit and
        nwln
        sub    AL,'0'        ; convert to numeric equivalent
        mov    SI,AX         ; SI is index into jump table
        add    SI,SI         ; SI := SI * 2
        jmp    jump_table[SI] ; indirect jump based on SI
test_termination:
        cmp    AL,2
        ja     done
        jmp    read_again
```

```
code_for_0:
        PutStr msg_0
        nwln
        jmp    test_termination
code_for_1:
        PutStr msg_1
        nwln
        jmp    test_termination
code_for_2:
        PutStr msg_2
        nwln
        jmp    test_termination
default_code:
        PutStr msg_default
        nwln
        jmp    test_termination
done:
        .EXIT
main    ENDP
        END main
```