```
 1:  TITLE      Linear search of integer array      LIN_SRCH.ASM
 2:  COMMENT |
 3:          Objective: To implement linear search of an integer
 4:                     array; demonstrates the use of loopne.
 5:              Input: Requests numbers to fill array and a
 6:                     number to be searched for from user.
 7:             Output: Displays the position of the number in
 8:                     the array if found; otherwise, not found
 9:  |                  message.
10:  .MODEL SMALL
11:  .STACK 100H
12:  .DATA
13:  MAX_SIZE       EQU 100
14:  array          DW  MAX_SIZE DUP (?)
15:  input_prompt   DB  'Please enter input array: '
16:                 DB  '(negative number terminates input)',0
17:  query_number   DB  'Enter the number to be searched: ',0
18:  out_msg        DB  'The number is at position ',0
19:  not_found_msg  DB  'Number not in the array!',0
20:  query_msg      DB  'Do you want to quit (Y/N): ',0
21:
22:  .CODE
23:  INCLUDE io.mac
```

```
24:  main    PROC
25:           .STARTUP
26:           PutStr  input_prompt ; request input array
27:           mov     BX,OFFSET array
28:           mov     CX,MAX_SIZE
29:  array_loop:
30:           GetInt  AX           ; read an array number
31:           nwln
32:           cmp     AX,0         ; negative number?
33:           jl      exit_loop    ; if so, stop reading numbers
34:           mov     [BX],AX      ; otherwise, copy into array
35:           inc     BX           ; increment array address
36:           inc     BX
37:           loop    array_loop   ; iterates a maximum of MAX_SIZE
38:  exit_loop:
39:           mov     DX,BX        ; DX keeps the actual array size
40:           sub     DX,OFFSET array  ; DX := array size in bytes
41:           sar     DX,1         ; divide by 2 to get array size
42:  read_input:
43:           PutStr  query_number ; request number to be searched for
44:           GetInt  AX           ; read the number
45:           nwln
```

Selection: 2

```
46:           push    AX              ; push number, size & array pointer
47:           push    DX
48:           push    OFFSET array
49:           call    linear_search
50:           ; linear_search returns in AX the position of the number
51:           ; in the array; if not found, it returns 0.
52:           cmp     AX,0            ; number found?
53:           je      not_found       ; if not, display number not found
54:           PutStr  out_msg         ; else, display number position
55:           PutInt  AX
56:           jmp     SHORT user_query
57:   not_found:
58:           PutStr  not_found_msg
59:   user_query:
60:           nwln
61:           PutStr  query_msg       ; query user whether to terminate
62:           GetCh   AL              ; read response
63:           nwln
64:           cmp     AL,'Y'          ; if response is not 'Y'
65:           jne     read_input      ; repeat the loop
66:   done:                           ; otherwise, terminate program
67:           .EXIT
68:   main    ENDP
```

Selection: 3

```
70:  ;---------------------------------------------------
71:  ; This procedure receives a pointer to an array of integers,
72:  ; the array size, and a number to be searched via the stack.
73:  ; If found, it returns in AX the position of the number in
74:  ; the array; otherwise, returns 0.
75:  ; All registers, except AX, are preserved.
76:  ;---------------------------------------------------
77:  linear_search  PROC
78:         push    BP
79:         mov     BP,SP
80:         push    BX              ; save registers
81:         push    CX
82:         mov     BX,[BP+4]       ; copy array pointer
83:         mov     CX,[BP+6]       ; copy array size
84:         mov     AX,[BP+8]       ; copy number to be searched
85:         sub     BX,2            ; adjust index to enter loop
86:  search_loop:
87:         add     BX,2            ; update array index
88:         cmp     AX,[BX]         ; compare the numbers
89:         loopne  search_loop
90:         mov     AX,0            ; set return value to zero
91:         jne     number_not_found  ; modify it if number found
92:         mov     AX,[BP+6]       ; copy array size
93:         sub     AX,CX           ; compute array index of number
94:  number_not_found:
```

```
 95:          pop     CX              ; restore registers
 96:          pop     BX
 97:          pop     BP
 98:          ret     6
 99:  linear_search  ENDP
100:          END   main
```

Selection: 5

```
 1:  TITLE    Sorting an array by selection sort    SEL_SORT.ASM
 2:  COMMENT |
 3:          Objective: To sort an integer array using selection sort.
 4:              Input: Requests numbers to fill array.
 5:  |         Output: Displays sorted array.
 6:  .MODEL SMALL
 7:  .STACK 100H
 8:  .DATA
 9:  MAX_SIZE       EQU 100
10:  array          DW  MAX_SIZE DUP (?)
11:  input_prompt   DB  'Please enter input array: '
12:                 DB  '(negative number terminates input)',0
13:  out_msg        DB  'The sorted array is:',0
14:
15:  .CODE
16:  .486
17:  INCLUDE io.mac
18:  main    PROC
19:          .STARTUP
20:          PutStr  input_prompt ; request input array
21:          mov     BX,OFFSET array
22:          mov     CX,MAX_SIZE
23:  array_loop:
```

```
24:          GetInt  AX              ; read an array number
25:          nwln
26:          cmp     AX,0            ; negative number?
27:          jl      exit_loop       ; if so, stop reading numbers
28:          mov     [BX],AX         ; otherwise, copy into array
29:          add     BX,2            ; increment array address
30:          loop    array_loop      ; iterates a maximum of MAX_SIZE
31:  exit_loop:
32:          mov     DX,BX           ; DX keeps the actual array size
33:          sub     DX,OFFSET array  ; DX := array size in bytes
34:          sar     DX,1            ; divide by 2 to get array size
35:          push    DX              ; push array size & array pointer
36:          push    OFFSET array
37:          call    selection_sort
38:          PutStr  out_msg         ; display sorted array
39:          nwln
40:          mov     CX,DX
41:          mov     BX,OFFSET array
42:  display_loop:
43:          PutInt  [BX]
44:          nwln
45:          add     BX,2
46:          loop    display_loop
47:  done:
48:          .EXIT
49:  main    ENDP
```

```
51:  ;-------------------------------------------------------
52:  ; This procedure receives a pointer to an array of integers
53:  ; and the array size via the stack. The array is sorted by
54:  ; using the selection sort. All registers are preserved.
55:  ;-------------------------------------------------------
56:  SORT_ARRAY  EQU  [BX]
57:  selection_sort PROC
58:         pusha                   ; save registers
59:         mov     BP,SP
60:         mov     BX,[BP+18]      ; copy array pointer
61:         mov     CX,[BP+20]      ; copy array size
62:         sub     SI,SI           ; array left of SI is sorted
63:  sort_outer_loop:
64:         mov     DI,SI
65:         ; DX is used to maintain the minimum value and AX
66:         ; stores the pointer to the minimum value
67:         mov     DX,SORT_ARRAY[SI]  ; min. value is in DX
68:         mov     AX,SI           ; AX := pointer to min. value
69:         push    CX
70:         dec     CX              ; size of array left of SI
71:  sort_inner_loop:
```

```
71:  sort_inner_loop:
72:          add     DI,2            ; move to next element
73:          cmp     DX,SORT_ARRAY[DI] ; less than min. value?
74:          jle     skip1           ; if not, no change to min. value
75:          mov     DX,SORT_ARRAY[DI] ; else, update min. value (DX)
76:          mov     AX,DI                ;       & its pointer (AX)
77:  skip1:
78:          loop    sort_inner_loop
79:          pop     CX
80:          cmp     AX,SI           ; AX = SI?
81:          je      skip2           ; if so, element at SI is its place
82:          mov     DI,AX           ; otherwise, exchange
83:          mov     AX,SORT_ARRAY[SI]  ; exchange min. value
84:          xchg    AX,SORT_ARRAY[DI]  ; & element at SI
85:          mov     SORT_ARRAY[SI],AX
86:  skip2:
87:          add     SI,2            ; move SI to next element
88:          dec     CX
89:          cmp     CX,1            ; if CX = 1, we are done
90:          jne     sort_outer_loop
91:          popa                    ; restore registers
92:          ret     4
93:  selection_sort ENDP
94:          END   main
```

```
 1:  TITLE    Sample indirect jump example    IJUMP.ASM
 2:  COMMENT |
 3:          Objective: To demonstrate the use of indirect jump.
 4:              Input: Requests a digit character from the user.
 5:                     WARNING: Typing any other character may
 6:                              crash the system!
 7:  |           Output: Appropriate class selection message.
 8:  .MODEL SMALL
 9:  .STACK  100H
10:  .DATA
11:  jump_table  DW  code_for_0   ; indirect jump pointer table
12:              DW  code_for_1
13:              DW  code_for_2
14:              DW  default_code ; default code for digits 3-9
15:              DW  default_code
16:              DW  default_code
17:              DW  default_code
18:              DW  default_code
19:              DW  default_code
20:              DW  default_code
21:
```

```
22:  prompt_msg  DB  'Type a character (digits ONLY): ',0
23:  msg_0       DB  'Economy class selected.',0
24:  msg_1       DB  'Business class selected.',0
25:  msg_2       DB  'First class selected.',0
26:  msg_default DB  'Not a valid code!',0
28:  .CODE
29:  INCLUDE  io.mac
30:  main PROC
31:          .STARTUP
32:  read_again:
33:          PutStr prompt_msg    ; request a digit
34:          sub    AX,AX         ; AX := 0
35:          GetCh  AL            ; read input digit and
36:          nwln
37:          sub    AL,'0'        ; convert to numeric equivalent
38:          mov    SI,AX         ; SI is index into jump table
39:          add    SI,SI         ; SI := SI * 2
40:          jmp    jump_table[SI] ; indirect jump based on SI
41:  test_termination:
42:          cmp    AL,2
43:          ja     done
44:          jmp    read_again
```

```
45:  code_for_0:
46:           PutStr msg_0
47:           nwln
48:           jmp    test_termination
49:  code_for_1:
50:           PutStr msg_1
51:           nwln
52:           jmp    test_termination
53:  code_for_2:
54:           PutStr msg_2
55:           nwln
56:           jmp    test_termination
57:  default_code:
58:           PutStr msg_default
59:           nwln
60:           jmp    test_termination
61:  done:
62:           .EXIT
63:  main     ENDP
64:           END main
```

Selection: 12