CSCI 224: Assembly Language Programming

Working with structures in NASM

A structure groups several fields in a single unit. The syntax to define a structure is shown on the following example:

```
STRUC Point
.x: RESD 1
.y: RESD 1
.size:
ENDSTRUC
```

Here `Point` is the structure name and the fields `x` and `y` define the offsets within the structure. Thus, `x = 0` and `y = 4`. The `y` field is initialized with 0. The dots in front of the field names are optional. However, they make the fields name local to the structure and allow to distinguish between similarly named fields in different structures by using the dot notation.

The next example shows how to declare a structure in the `.data` segment:

```
SEGMENT .data
myPoint: ISTRUC Point
AT Point.x, DD 1
AT Point.y, DD 2
IEND
```

The example below shows how to reserve place for structure in .bss segment:

```
SEGMENT .bss
myPoint: RESB Point.size
```

To access the structure fields use the field references, or indirect operands, or the dot notation:

```
SEGMENT .data
p: ISTRUC Point
AT Point.x, DD 1
AT Point.y, DD 2
IEND

SEGMENT.text
mov eax, [p + Point.x]        ; eax = 1 (field reference)
mov esi, p                    ; (indirect operands)
mov eax, [esi + 4]            ; eax = y
mov eax, [esi + Point.y]      ; eax = y
```

The following code expects the user to enter 10 integer numbers and puts then in 5 structures of type `Point` in the first loop. The second loop over the filled structures causes printing of the points coordinates. The last portion of code prints the point declared in the .data segment.

```
%INCLUDE "csci224.inc"

STRUC Point                         ; structure definition
```

```
.x: RESD 1
.y: RESD 1
.size:
ENDSTRUC

SEGMENT .data
prompt: DB "Enter point (x y): ",0

p:      ISTRUC Point                ; declare an instance of point and
AT Point.x, DD 5                    ; initialize its fields
AT Point.y, DD 7
IEND

SEGMENT .bss
pArr:   RESB Point.size*5           ; reserve place for 5 structures
pArr_l: EQU ($ - pArr) / Point.size

SEGMENT .text
main
     mov  ecx, pArr_l               ; reading points coords from keyboard
     mov  esi, pArr
L1:  mov  edx, prompt
     call WriteString               ; print prompt
     call ReadInt                   ; read x
     mov  [esi + Point.x], eax      ; store it in STRUC
     call ReadInt                   ; read y
     mov  [esi + Point.y], eax      ; store it in STRUC
     add  esi, Point.size           ; advance esi to the next point
     loop L1                        ; read all points
     call Crlf

     mov  ecx, pArr_l               ; output points to the screen
     mov  esi, pArr
L2:  mov  eax, [esi + Point.x]      ; get x-coord
     call WriteInt                  ; output it
     mov  al, ' '                   ; space between the x/y values
     call WriteChar
     mov  eax, [esi + Point.y]      ; get y-coord
     call WriteInt                  ; output it
     call Crlf                      ; start new line
     add  esi, Point.size           ; get to the next point
     loop L2                        ; output 'em all
     call Crlf

     mov  eax, [p + Point.x]        ; get x-coord
     call WriteInt                  ; output it
     mov  al, ' '                   ; space between the x/y values
     call WriteChar
     mov  eax, [p + Point.y]        ; get y-coord
     call WriteInt                  ; output it
     call Crlf

     ret
```