Structures Containing Structures

Structures can contain instances of other structures. This is called a nested structure.

For example, a Rectangle structure can be defined in terms of its upper-left and lower-right corners, both COORD structures:

```
Rectangle STRUCT
    UpperLeft COORD <>
    LowerRight COORD <>
Rectangle ENDS
```

This means that a **Rectangle structure** will contain **two COORD structures**, one for the upper-left corner and one for the lower-right corner.

Rectangle variables can be declared without overrides or by overriding individual COORD fields. The following examples show how to declare Rectangle variables:

```
rect1 Rectangle < >
rect2 Rectangle { }
rect3 Rectangle { {10,10}, {50,20} }
rect4 Rectangle < <10,10>, <50,20> >
```

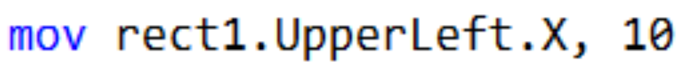
The first two declarations, rect1 and rect2, will create Rectangle variables with the default values for the UpperLeft and LowerRight fields.

The third declaration, rect3, will create a Rectangle variable with the specified values for the UpperLeft and LowerRight fields.

The fourth declaration, rect4, is an alternative way to declare a Rectangle variable with the pecified values for the UpperLeft and LowerRight fields.

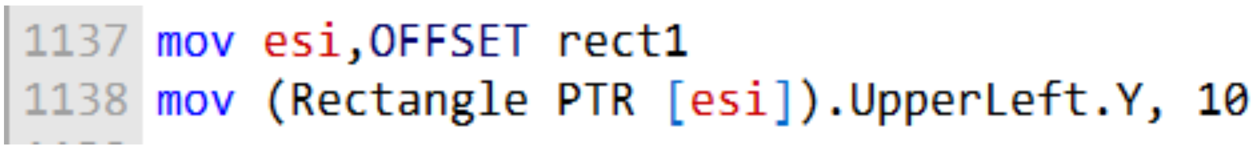
Once a Rectangle variable has been declared, you can access its fields using the dot notation.

For example, the following code moves the value 10 to the X coordinate of the upper-left corner of the rect1 variable:



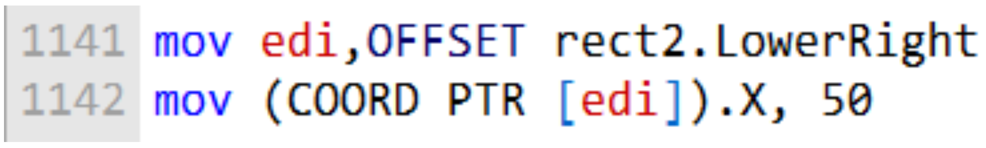
```
mov rect1.UpperLeft.X, 10
```

You can also access a structure field using an indirect operand. For example, the following code moves the value 10 to the Y coordinate of the upper-left corner of the structure pointed to by the esi register:



```
mov esi,OFFSET rect1
mov (Rectangle PTR [esi]).UpperLeft.Y, 10
```

The OFFSET operator can be used to return pointers to individual structure fields, including nested fields. For example, the following code moves the value 50 to the X coordinate of the lower-right corner of the rect2 variable:



```
mov edi,OFFSET rect2.LowerRight
mov (COORD PTR [edi]).X, 50
```

<u>**Example program:**</u>

```
; Drunkard's Walk
; Drunkard's walk program simulates a professor's random path in an imaginary grid.
; The professor starts at coordinates 25, 25 and wanders around the immediate area.

INCLUDE Irvine32.inc

; Constants
WalkMax = 50     ; Maximum number of steps
StartX = 25      ; Starting X-coordinate
StartY = 25      ; Starting Y-coordinate

; Define a structure to store the path and number of steps
DrunkardWalk STRUCT
    path COORD WalkMax DUP(<0,0>) ; Array of COORD objects for the path
    pathsUsed WORD 0               ; Number of steps taken
DrunkardWalk ENDS

; Prototypes
DisplayPosition PROTO currX:WORD, currY:WORD

.data
aWalk DrunkardWalk <> ; Create an instance of the DrunkardWalk structure

.code
main PROC
    mov esi,OFFSET aWalk ; Get the address of the aWalk structure
    call TakeDrunkenWalk ; Simulate the professor's walk
    exit
main ENDP
```

```
;---------------------------------------------------------
TakeDrunkenWalk PROC
LOCAL currX:WORD, currY:WORD

; Takes a walk in random directions (north, south, east, west).
; Receives: ESI points to a DrunkardWalk structure
; Returns: the structure is initialized with random values
;---------------------------------------------------------

pushad ; Preserve registers

; Initialize EDI with the address of the path array
mov edi,esi
add edi,OFFSET DrunkardWalk.path

mov ecx,WalkMax  ; Set loop counter
mov currX,StartX ; Initialize current X-location
mov currY,StartY ; Initialize current Y-location

Again:
; Insert the current location in the array.
mov ax,currX
mov (COORD PTR [edi]).X,ax ; Store X-coordinate
mov ax,currY
mov (COORD PTR [edi]).Y,ax ; Store Y-coordinate

INVOKE DisplayPosition, currX, currY ; Display the current position

mov eax,4 ; Choose a random direction (0-3)
call RandomRange
```

```
.IF eax == 0
    ; North
    dec currY
.ELSEIF eax == 1
    ; South
    inc currY
.ELSEIF eax == 2
    ; West
    dec currX
.ELSE
    ; East (EAX = 3)
    inc currX
.ENDIF

add edi,TYPE COORD ; Move to the next COORD
loop Again

Finish:
mov (DrunkardWalk PTR [esi]).pathsUsed, WalkMax ; Set pathsUsed to WalkMax
popad ; Restore registers
ret
TakeDrunkenWalk ENDP
```

```
;-----------------------------------------------------------
DisplayPosition PROC currX:WORD, currY:WORD
; Display the current X and Y positions.
;-----------------------------------------------------------

.data
commaStr BYTE ",",0 ; Comma string

.code
pushad ; Preserve registers

; Display the current X position
movzx eax,currX
call WriteDec

mov edx,OFFSET commaStr ; Load the comma string
call WriteString

; Display the current Y position
movzx eax,currY
call WriteDec

call Crlf ; Move to the next line

popad ; Restore registers
ret
DisplayPosition ENDP

END main
```

I'll provide an explanation of the "Drunkard's Walk" program, which simulates a professor's random path in an imaginary grid:

Include Directives:
The program begins with an INCLUDE directive to include the Irvine32.inc library, which provides functions for console I/O and other useful features. Constants:

Constants like WalkMax, StartX, and StartY are defined at the beginning. These constants determine the maximum number of steps in the walk and the starting coordinates of the professor.

Structure Definition:

The program defines a structure called DrunkardWalk using the STRUCT and ENDS directives. This structure contains two components: path: An array of COORD objects, which will store the professor's path. pathsUsed: A word to keep track of the number of steps taken.

Prototypes:

The DisplayPosition function's prototype is declared. It's used to display the current position (X and Y coordinates) of the professor. Data Section:

The .data section is used to define data, including a variable aWalk, which is an instance of the DrunkardWalk structure.

Code Section:

The .code section contains the program's main logic. main Procedure: The main procedure is the program's entry point. It initializes esi with the address of the aWalk structure. Then, it calls the TakeDrunkenWalk procedure to simulate the professor's walk.

TakeDrunkenWalk Procedure:

This procedure simulates the professor's walk. It initializes local variables currX and currY to the starting coordinates and uses a loop to simulate the walk. Inside the loop: The current coordinates are inserted into the path array.

The DisplayPosition function is called to display the current position. A random direction (north, south, east, or west) is chosen for the next step. The loop continues until the maximum number of steps (WalkMax) is reached.

DisplayPosition Procedure:

This procedure displays the current X and Y positions with proper formatting. It uses WriteDec to display the coordinates and WriteString to add a comma between X and Y. It also adds a line break with Crlf to format the output.

Program Termination:

After the professor's walk is completed, the program sets the pathsUsed field of the aWalk structure to WalkMax to indicate the number of steps taken.

End of Program:

The END main statement marks the end of the program. In summary, this program uses a structure to simulate a random walk for a professor. It keeps track of the professor's path and displays the X and Y coordinates at each step.

The professor starts at a specified position, and the program randomly selects the next step's direction until the maximum number of steps is reached. The structure helps organize and manage the data related to the walk.