Assembly Language for Intel-Based Computers, 4th Edition

Kip R. Irvine

Chapter 10: Structures and Macros

Chapter Overview

- Structures
- Macros
- Conditional-Assembly Directives
- Defining Repeat Blocks

Structure

- A template or pattern given to a logically related group of variables.
- field - structure member containing data
- Program access to a structure:
  - entire structure as a complete unit
  - individual fields
- Useful way to pass multiple related arguments to a procedure
  - example: file directory information

Using a Structure

Using a structure involves three sequential steps:

1. Define the structure.
2. Declare one or more variables of the structure type, called structure variables.
3. Write runtime instructions that access the structure.

Structure Definition Syntax

```
name STRUCT
    field-declarations
name ENDS
```

- Field-declarations are identical to variable declarations

COORD Structure

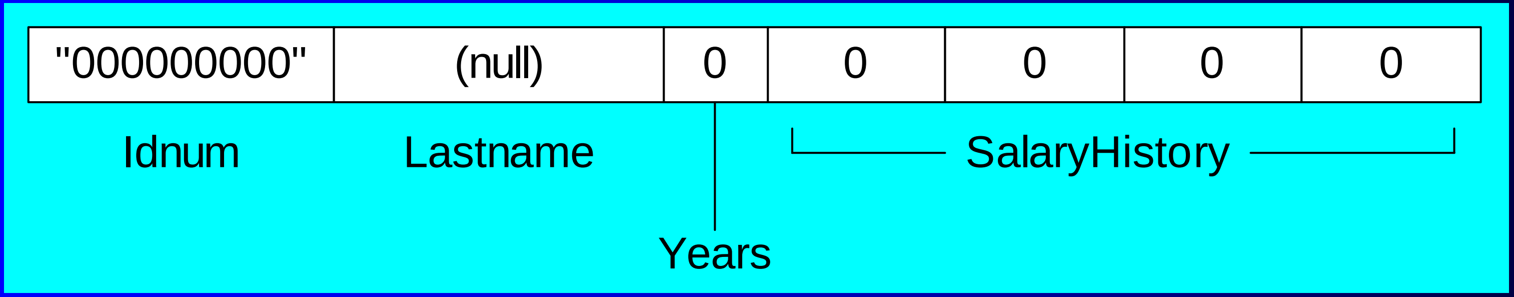
- The COORD structure used by the MS-Windows programming library identifies X and Y screen coordinates

```
COORD STRUCT
    X WORD ?            ; offset 00
    Y WORD ?            ; offset 02
COORD ENDS
```

Employee Structure

A structure is ideal for combining fields of different types:

```
Employee STRUCT
   IdNum BYTE "000000000"
   LastName BYTE 30 DUP(0)
   Years WORD 0
   SalaryHistory DWORD 0,0,0,0
Employee ENDS
```



Declaring Structure Variables

- Structure name is a user-defined type
- Insert replacement initializers between brackets:

  < . . . >
- Empty brackets <> retain the structure's default field initializers
- Examples:

```
.data
point1 COORD <5,10>
point2 COORD <>
worker Employee <>
```

Initializing Array Fields

- Use the DUP operator to initialize one or more elements of an array field:

```
.data
emp Employee <,,,4 DUP(20000)>
```

Array of Structures

- An array of structure objects can be defined using the DUP operator.
- Initializers can be used

```
NumPoints = 3
AllPoints COORD NumPoints DUP(<0,0>)

RD_Dept Employee 20 DUP(<>)

accounting Employee 10 DUP(<,,,4 DUP(20000) >)
```

Referencing Structure Variables

```
Employee STRUCT                         ; bytes
   IdNum BYTE "000000000"               ; 9
   LastName BYTE 30 DUP(0)              ; 30
   Years WORD 0                         ; 2
   SalaryHistory DWORD 0,0,0,0          ; 16
Employee ENDS                           ; 57
```

```
.data
worker Employee <>

mov eax,TYPE Employee                   ; 57
mov eax,SIZEOF Employee                 ; 57
mov eax,SIZEOF worker                   ; 57
mov eax,TYPE Employee.SalaryHistory     ; 4
mov eax,LENGTHOF Employee.SalaryHistory ; 4
mov eax,SIZEOF Employee.SalaryHistory   ; 16
```

. . . continued

```
mov dx,worker.Years
mov worker.SalaryHistory,20000        ; first salary
mov worker.SalaryHistory+4,30000      ; second salary
mov edx,OFFSET worker.LastName

mov esi,OFFSET worker
mov ax,(Employee PTR [esi]).Years

mov ax,[esi].Years    ; invalid operand (ambiguous)
```

Looping Through an Array of Points

Sets the X and Y coordinates of the AllPoints array to sequentially increasing values (1,1), (2,2), ...

```
.data
NumPoints = 3
AllPoints COORD NumPoints DUP(<0,0>)

.code
   mov edi,0                 ; array index
   mov ecx,NumPoints         ; loop counter
   mov ax,1                  ; starting X, Y values
L1:
   mov (COORD PTR AllPoints[edi]).X,ax
   mov (COORD PTR AllPoints[edi]).Y,ax
   add edi,TYPE COORD
   inc ax
   Loop L1
```

Example: Displaying the System Time (1 of 3)

- Retrieves and displays the system time at a selected screen location.
- Uses COORD and SYSTEMTIME structures:

```
SYSTEMTIME STRUCT
   wYear WORD ?
   wMonth WORD ?
   wDayOfWeek WORD ?
   wDay WORD ?
   wHour WORD?
   wMinute WORD ?
   wSecond WORD ?
   wMilliseconds WORD ?
SYSTEMTIME ENDS
```

Example: Displaying the System Time (2 of 3)

- Uses a Windows API call to get the standard console output handle. SetConsoleCursorPosition positions the cursor. GetLocalTime gets the current time of day:

```
.data
sysTime SYSTEMTIME <>
XYPos COORD <10,5>
consoleHandle DWORD ?
colonStr BYTE “:”,0
.code
INVOKE GetStdHandle, STD_OUTPUT_HANDLE
mov consoleHandle,eax

INVOKE SetConsoleCursorPosition,
   consoleHandle, XYPos
INVOKE GetLocalTime, ADDR sysTime
```

Example: Displaying the System Time (3 of 3)

- Display the time using library calls:

```
movzx eax,sysTime.wHour        ; hours
call  WriteDec
mov   edx,offset colonStr      ; ":"
call  WriteString
movzx eax,sysTime.wMinute      ; minutes
call  WriteDec
mov   edx,offset colonStr      ; ":"
call  WriteString
movzx eax,sysTime.wSecond      ; seconds
call  WriteDec
```

Nested Structures (1 of 2)

- Define a structure that contains other structures.
- Used nested braces (or brackets) to initialize each COORD structure.

```
COORD STRUCT
   X WORD ?
   Y WORD ?
COORD ENDS
```

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

.code
rect1 Rectangle { {10,10}, {50,20} }
rect2 Rectangle < <10,10>, <50,20> >
```

Nested Structures (2 of 2)

- Use the dot (.) qualifier to access nested fields.
- Use indirect addressing to access the overall structure or one of its fields

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

// use the OFFSET operator
mov edi,OFFSET rect2.LowerRight
mov (COORD PTR [edi]).X, 50
mov edi,OFFSET rect2.LowerRight.X
mov WORD PTR [edi], 50
```

Example: Drunkard's Walk

- Random-path simulation
- Uses a nested structure to accumulate path data as the simulation is running
- Uses a multiple branch structure to choose the direction

```
WalkMax = 50
DrunkardWalk STRUCT
    path COORD WalkMax DUP(<0,0>)
    pathsUsed WORD 0
DrunkardWalk ENDS
```

View the source code

Declaring and Using Unions

- A union is similar to a structure in that it contains multiple fields
- All of the fields in a union begin at the same offset
  - (differs from a structure)
- Provides alternate ways to access the same data
- Syntax:

unionname UNION

    union-fields

unionname ENDS

Integer Union Example

The Integer union consumes 4 bytes (equal to the largest field)

```
Integer UNION
   D DWORD 0
   W WORD 0
   B BYTE 0
Integer ENDS
```

D, W, and B are often called variant fields.

Integer can be used to define data:

```
.data
val1 Integer <12345678h>
val2 Integer <100h>
val3 Integer <>
```

Union Inside a Structure

An Integer union can be enclosed inside a FileInfo structure:

```
Integer UNION
   D DWORD 0
   W WORD 0
   B BYTE 0
Integer ENDS

FileInfo STRUCT
   FileID Integer <>
   FileName BYTE 64 DUP(?)
FileInfo ENDS

.data
myFile FileInfo <>
.code
mov myFile.FileID.W, ax
```