Chapter 10: Structures and Macros

Chapter 10: Structures and Macros

Chapter Overview

- Structures
- Macros

2

Structures - Overview

- Defining Structures
- Declaring Structure Variables
- Referencing Structure Variables
- Nested Structures
- Declaring and Using Unions

3

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

4

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.

5

Structure Definition Syntax

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

- Field-declarations are identical to variable declarations

6

COORD Structure

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

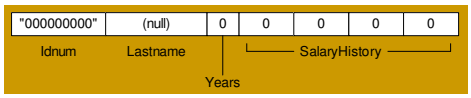
```
COORD STRUCT
   X WORD ?          ; offset 00
   Y WORD ?          ; offset 02
COORD ENDS
```

7

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
```



8

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 <>
```

9

Initializing Array Fields

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

```
.data
emp Employee <,,,2 DUP(20000)>
```

10

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) >)
```

11

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
```

12

. . . 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)

- GetStdHandle gets 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 ?
.code
INVOKE GetStdHandle, STD_OUTPUT_HANDLE
mov consoleHandle,eax
INVOKE SetConsoleCursorPosition, consoleHandle, XYPos
INVOKE GetLocalTime, ADDR sysTime
```

16

Example: Displaying the System Time (3 of 3)

- Display the time using library calls:

```
mov   edx,OFFSET TheTimeIs    ; "The time is "
call  WriteString
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
```

17

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> >
```

18

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
```

19

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
```

20

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

21

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 <>
```

22

Integer Union Example

The variant field name is required when accessing the union:

```
mov val3.B, al
mov ax,val3.W
add val3.D, eax
```

23

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
```

24

Macros

- Introducing Macros
- Defining Macros
- Invoking Macros
- Macro Examples
- Nested Macros

25

Introducing Macros

- A macro is a named block of assembly language statements.
- Once defined, it can be invoked (called) one or more times.
- During the assembler's preprocessing step, each macro call is expanded into a copy of the macro.
- The expanded code is passed to the assembly step, where it is checked for correctness.

26

Defining Macros

- A macro must be defined before it can be used.
- Parameters are optional.
- Each parameter follows the rules for identifiers. It is a string that is assigned a value when the macro is invoked.
- Syntax:

```
macroname MACRO [parameter-1, parameter-2,...]
    statement-list
ENDM
```

27

mNewLine Macro Example

This is how you define and invoke a simple macro.

```
mNewLine MACRO              ; define the macro
    call Crlf
ENDM
.data

.code
mNewLine                    ; invoke the macro
```

The assembler will substitute "call crlf" for "mNewLine".

28

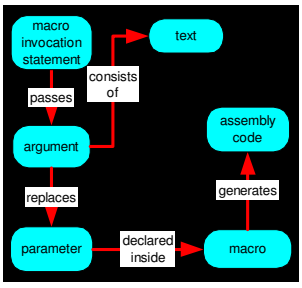
Invoking Macros (1 of 2)

- When you invoke a macro, each argument you pass matches a declared parameter.
- Each parameter is replaced by its corresponding argument when the macro is expanded.
- When a macro expands, it generates assembly language source code.
- Arguments are treated as simple text by the preprocessor.

29

Invoking Macros (2 of 2)

Relationships between macros, arguments, and parameters:



30

mWriteStr Macro (1 of 2)

Provides a convenient way to display a string, by passing the string name as an argument.

```
mWriteStr MACRO buffer
   push edx
   mov  edx,OFFSET buffer
   call WriteString
   pop  edx
ENDM
.data
str1 BYTE "Welcome!",0
.code
mWriteStr str1
```

31

mWriteStr Macro (2 of 2)

The expanded code shows how the str1 argument replaced the parameter named buffer:

```
mWriteStr MACRO buffer
     push edx
     mov  edx,OFFSET buffer
     call WriteString
     pop  edx
ENDM
```

```
1    push edx
1    mov  edx,OFFSET str1
1    call WriteString
1    pop  edx
```

32

mPutChar Macro

Writes a single character to standard output.

Definition:

```
mPutchar MACRO char
   push eax
   mov al,char
   call WriteChar
   pop eax
ENDM
```

Invocation:

```
.code
mPutchar 'A'
```

Expansion:

```
1      push eax
1      mov al,'A'
1      call WriteChar
1      pop eax
```

viewed in the listing file

33

Invalid Argument

- If you pass an invalid argument, the error is caught when the expanded code is assembled.
- Example:

```
.code
mPutchar 1234h
```

```
1       push eax
1       mov al,1234h            ; error!
1       call WriteChar
1       pop eax
```

34

Blank Argument

- If you pass a blank argument, the error is also caught when the expanded code is assembled.
- Example:

```
.code
mPutchar
```

```
1       push eax
1       mov al,     ;Error
1       call WriteChar
1       pop eax
```

35

Macro Examples (pg 320 - 325)

- mReadStr - reads string from standard input
- mGotoXY - locates the cursor on screen
- mDumpMem - dumps a range of memory

36

mReadStr

The mReadStr macro provides a convenient wrapper around ReadString procedure calls.

```
mReadStr MACRO varName
    push ecx
    push edx
    mov edx,OFFSET varName
    mov ecx,(SIZEOF varName) - 1
    call ReadString
    pop edx
    pop ecx
ENDM
.data
firstName BYTE 30 DUP(?)
.code
mReadStr firstName
```

37

mGotoXY

The mGotoXY macro ets the console cursor position by calling the Gotoxy library procedure.

```
mGotoxy MACRO X:REQ, Y:REQ
    push edx
    mov  dh,Y
    mov  dl,X
    call Gotoxy
    pop  edx
ENDM
```

The REQ next to X and Y identifies them as required parameters.

38

mDumpMem

The mDumpMem macro streamlines calls to the link library's DumpMem procedure.

```
mDumpMem MACRO address, itemCount, componentSize
    push ebx
    push ecx
    push esi
    mov  esi,address
    mov  ecx,itemCount
    mov  ebx,componentSize
    call DumpMem
    pop  esi
    pop  ecx
    pop  ebx
ENDM
```

39

mDump

The mDump macro displays a variable, using its known attributes. If <useLabel> is nonblank, the name of the variable is displayed.

```
mDump MACRO varName:REQ, useLabel
    IFB <varName>
      EXITM
    ENDIF
    call Crlf
    IFNB <useLabel>
      mWrite "Variable name: &varName"
    ELSE
      mWrite " "
    ENDIF
    mDumpMem OFFSET varName, LENGTHOF varName,
        TYPE varName
ENDM
```

40

mWrite

The mWrite macro writes a string literal to standard output. It is a good example of a macro that contains both code and data.

```
mWrite MACRO text
    LOCAL string
    .data                       ;; data segment
    string BYTE text,0          ;; define local string
    .code                       ;; code segment
    push edx
    mov  edx,OFFSET string
    call Writestring
    pop  edx
ENDM
```

The LOCAL directive prevents string from becoming a global label.

41

Nested Macros

The mWriteLn macro contains a nested macro (a macro invoked by another macro).

```
mWriteLn MACRO text
    mWrite text
    call Crlf
ENDM
```

```
mWriteLn "My Sample Macro Program"
```

```
2   .data
2   ??0002 BYTE "My Sample Macro Program",0
2   .code
2   push edx
2   mov  edx,OFFSET ??0002
2   call Writestring
2   pop  edx
1   call Crlf
```

nesting level

42

Your turn . . .

- Write a nested macro named mAskForString that clears the screen, locates the cursor at a given row and column, prompts the user, and inputs a string. Use any macros shown so far.
- Use the following code and data to test your macro:

```
.data
acctNum BYTE 30 DUP(?)
.code
main proc
   mAskForString 5,10,"Input Account Number: ", \
        acctNum
```

43

. . . Solution

```
mAskForString MACRO row, col, prompt, inbuf
   call Clrscr
   mGotoXY col, row
   mWrite prompt
   mReadStr inbuf
ENDM
```

44

The End

45