

Chapter 6 Pointers and Modular Programming

A pointer is a new variable type which holds the *address* of another variable.

Pointers are declared using the `*` operator.

Pointers

- pointer (pointer variable)
 - a memory cell that stores the address of a data item
 - syntax: *type *variable*

```
int m = 25;
int *itemp; /* a pointer to an integer */
```

In this example `itemp` is a pointer variable.

We can make `itemp` hold the address of the variable `m` like this:
`itemp = &m;`

Recall that `&` is the address operator that we used in `scanf`

```
#include<stdio.h>
int main()
{
    int m = 25;
    int *mPointer;
    mPointer = &m;
    printf("m = %d\n", m);
    printf("&m = %d\n", &m);
    printf("mPointer = %d\n", mPointer);
}
```

This program prints the following:

`m = 25`

`&m = 19922488`

`mPointer = 19922488`

Press any key to continue . . .

Notice that the *address* of `m` is stored in `mPointer`.

If you run this program again or on another computer the address of *m* may be at a different location – it depends on where the compiler places it in memory.

We can access the data stored in variable *m* by using its name as in:

```
printf("%d\n", m);
```

or,

we can access the data *indirectly* by using the variables address stored in *mPointer*. To do this we again use the *** operator as in:

```
printf("%d\n", *mPointer);
```

An indirect address is the address of an address of the data. When we use **mPointer* as a variable we effectively go to the *mPointer* location, get the address, and go to that address to find the data.

This is somewhat confusing. Note that we now have three different meanings for the *** operator and which meaning is used by the compiler depends on the context – that is, how it is used.

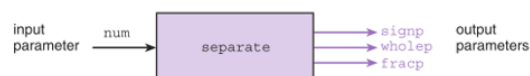
1. If we write *z = x * y*; the *** operator means multiplication
2. If we write *int *x*; the *** operator is used to declare *x* to be a pointer to an int variable.
3. If we use *y = *x*; the *** operator is used for indirect address. We go to the location *x* and find the information there which is the address of the data which gets moved into *y*.

Functions with Output Parameters

- We've used the return statement to send back one result value from a function.
- We can also use output parameters to return multiple results from a function.

FIGURE 6.4

Diagram of
Function *separate*
with Multiple
Results



For example, suppose we want to write a function which will swap two variables which are ints. If we name the function *Swap* and pass it two integers as arguments as in:

```
int x = 5, y = 9;
```

```
Swap(x, y);
```

The function will fail because *x* and *y* are passed by value.

<pre>int main() {int x = 5, y = 9; Swap(x, y); printf("%d %d\n", x, y); return 0; } int Swap(int x, int y) {int tmp; tmp = x; x = y; y = tmp; }</pre>	Swap	main	memory
		x	5
		y	9
	x		5 9
	y		9 5
	tmp		5

We can fix this program by passing the address of *x* and *y* using pointers.

<pre>void Swap(int *xPtr, int *yPtr); int main() {int x = 5, y = 9; Swap(&x, &y); printf("%d %d\n", x, y); return 0; } void Swap(int *xPtr, int *yPtr) {int tmp; tmp = *xPtr; *xPtr = *yPtr; *yPtr = tmp; }</pre>	Swap	main	memory
	*xPtr	x	5 9
	*yPtr	y	9 5
	tmp		5

Syntax for writing functions with output parameters

Prototype:

```
int MyFunction(int x, int *y);  
//x is passed by value, y is passed by reference
```

Calling statement:

```
int z;  
z = MyFunction(x, &y);  
//For reference parameters you must pass the address
```

Function definition

```
int MyFunction(int x, int *y)  
{  
    int z;  
    x = 2;  
    *y = 3;  
    //Reference parameter is used with dereferencing operator  
    z = x + *y;  
    return z;  
}
```

Example

Write a function which will prompt the user for two doubles and return these to the main program.

In the main program:

```
double a, b;  
Getab(&a, &b);
```

In the function

<pre>void Getab(double *a, double *b) { double x, y; printf("Enter a value for a... "); scanf_s("%lf", &x); *a = x; printf("Enter a value for b... "); scanf_s("%lf", &y); *b = y; }</pre>	<pre>void Getab(double *a, double *b) { printf("Enter a value for a... "); scanf_s("%lf", &*a); printf("Enter a value for b... "); scanf_s("%lf", &*b); }</pre>
--	---

Memory map example

Complete the memory map for the following program.

```
#include<stdio.h>
void Fun1(int a, int *b);
void Fun2(int a, int *b);
int main()
{
    int x = 5, y = 2;
    printf("%d, %d\n", x, y);
    Fun1(x, &y);
    printf("%d, %d\n", x, y);
}
void Fun1(int a, int *b)
{
    int c;
    printf("%d, %d\n", a, *b);
    c = *b;
    Fun2(c, &a);
    printf("%d, %d, %d\n", a, *b, c);
}
void Fun2(int a, int *b)
{
    int c;
    printf("%d, %d\n", a, *b);
    c = *b;
    printf("%d, %d, %d\n", a, *b, c);
}
```

Fun2	Fun1	Main	Data

Printed Output

Memory map example SOLUTION

Complete the memory map for the following program.

```
#include<stdio.h>
void Fun1(int a, int *b);
void Fun2(int a, int *b);
int main()
{
    int x = 5, y = 2;
    printf("%d, %d\n", x, y);
    Fun1(x, &y);
    printf("%d, %d\n", x, y);
}
void Fun1(int a, int *b)
{
    int c;
    printf("%d, %d\n", a, *b);
    c = *b;
    Fun2(c, &a);
    printf("%d, %d, %d\n", a, *b, c);
}
void Fun2(int a, int *b)
{
    int c;
    printf("%d, %d\n", a, *b);
    c = *b;
    printf("%d, %d, %d\n", a, *b, c);
}
```

Fun2	Fun1	Main	Data
		x	5
	*b	y	2
*b	a		5
	c		2
a			2
c			5

Printed Output
5, 2
5, 2
2, 5
2, 5, 5
5, 2, 2
5. 2

Pointers and output parameters

Polar coordinates are written in the form of $r\angle\theta$ and rectangular coordinates are expressed as the pair (x, y) . Write a program that prompts the user to enter r and θ (in degrees) and calls a function named *ConvertToXY* which converts the polar coordinates to Cartesian coordinates and returns them to the main program for printing. You will need to pass your function r and *theta* by value and x and y by reference.

To convert polar to Cartesian use:

$$x = r \cos(\theta)$$

$$y = r \sin(\theta)$$

$$\pi = 3.141592653589793$$

Turn in a printed copy of your source file.