Pointers and Modular Programming
Chapter 6

Problem Solving & Program Design in C

Eighth Edition

Jeri R. Hanly & Elliot B. Koffman

Chapter Objectives

- To learn about pointers and indirect addressing
- To see how to access external data files in a program and to be able to read from input file and write to output files using file pointers
- To learn how to return function results through a function's arguments
- To understand the differences between call-by-value and call-by-reference

Chapter Objectives

- To understand the distinction between input, inout, and output parameters and when to use each kind

Pointers

- pointer (pointer variable)
  - a memory cell that stores the address of a data item
  - 8 bytes on on server but depends on machine
  - syntax: *type *variable*

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

Pointers

- pointer (pointer variable)
  - a memory cell that stores the address of a data item
  - 8 bytes on on server but depends on machine
  - syntax: *type *variable*

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

& operator (address of)

- Returns the address of a variable

Indirection/indirect reference

accessing the contents of a memory cell through a pointer variable that stores it address

FIGURE 6.1

Referencing a Variable Through a Pointer

TABLE 6.1 References with Pointers

Reference	Cell Referenced	Cell Type (Value)
itemp	gray shaded cell	pointer (1024)
*itemp	cell in color	int (25)

* operator (indirection)

- Follows a pointer to what it points to
- (the thing at the address it stores)

Pointers to Files

- C allows a program to explicitly name a file for input or output.
- Declare file pointers:
  - FILE *inp; /* pointer to input file */
  - FILE *outp; /* pointer to output file */
- Prepare for input or output before permitting access:
  - inp = fopen(“infile.txt”, “r”);
  - outp = fopen(“outfile.txt”, “w”);

Pointers to Files

- fscanf
  - file equivalent of scanf
  - fscanf(inp, “%lf”, &item);
- fprintf
  - file equivalent of printf
  - fprintf(outp, “%.2f\n”, item);
- closing a file when done
  - fclose(inp);
  - fclose(outp);

Segmentation fault

- Runtime error
- Means you tried to access memory that you weren't allowed to access
- Examples of causes:
  - trying to read from a file that wasn't open
  - following a dangling pointer
  - accessing data beyond array bounds

Segmentation fault

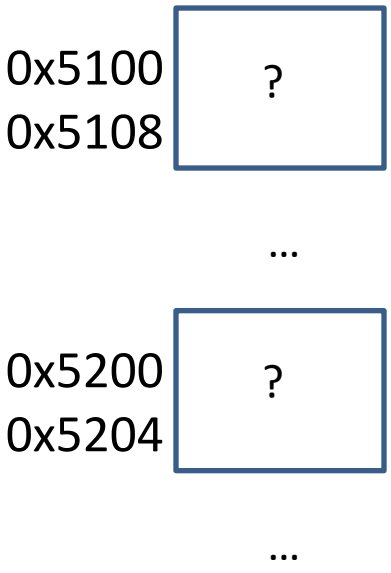
- Runtime error
- Means you tried to access memory that you weren't allowed to access
- Examples of causes:
  - trying to read from a file that wasn't open
  - following a dangling pointer
  - accessing data beyond array bounds

let's introduce a segmentation fault in read.c

Pointers

- Create an integer pointer variable and set it

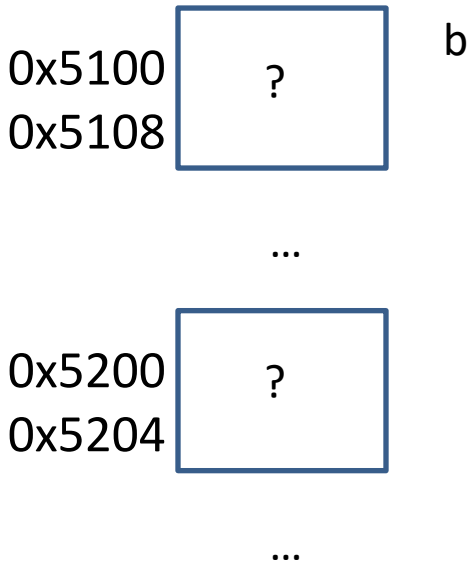
```
int main(void) {
    int *b;
    int n;
    n = 5;
    b = &n;
```



Pointers

- Create an integer pointer variable and set it

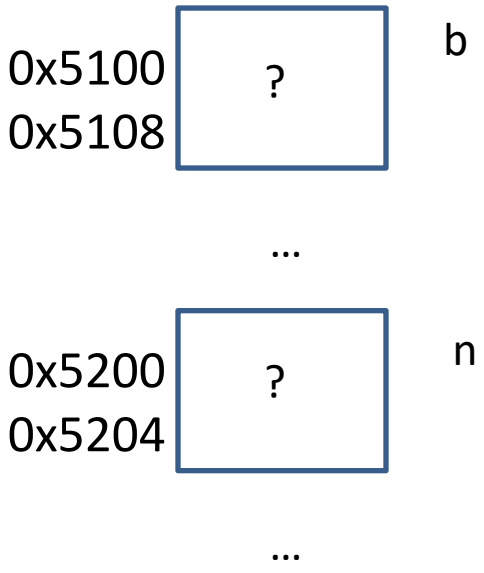
```
int main(void) {
    int *b;
    int n;
    n = 5;
    b = &n;
```



Pointers

- Create an integer pointer variable and set it

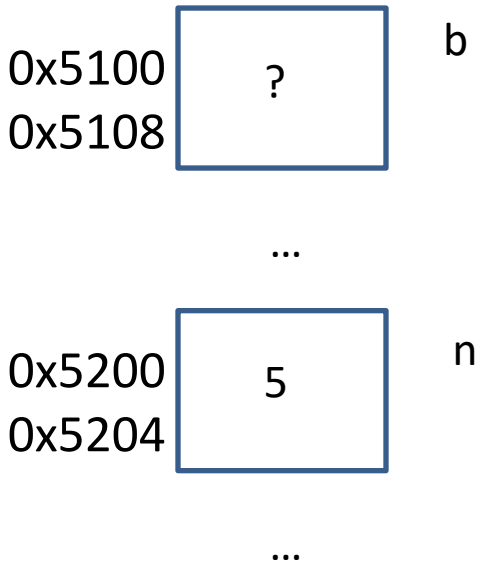
```
int main(void) {
    int *b;
    int n;
    n = 5;
    b = &n;
```



Pointers

- Create an integer pointer variable and set it

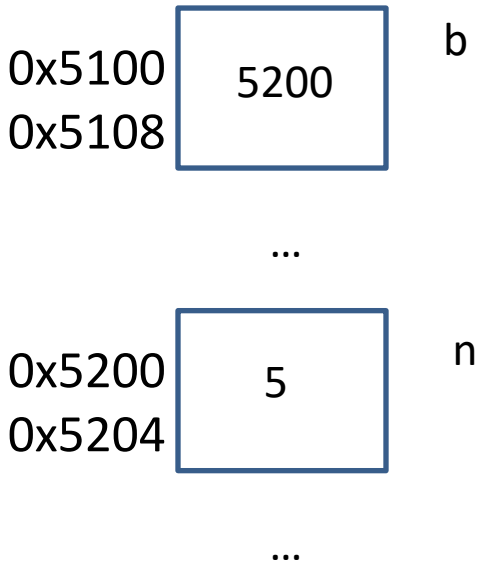
```
int main(void) {
    int *b;
    int n;
    n = 5;
    b = &n;
```



Pointers

- Create an integer pointer variable and set it

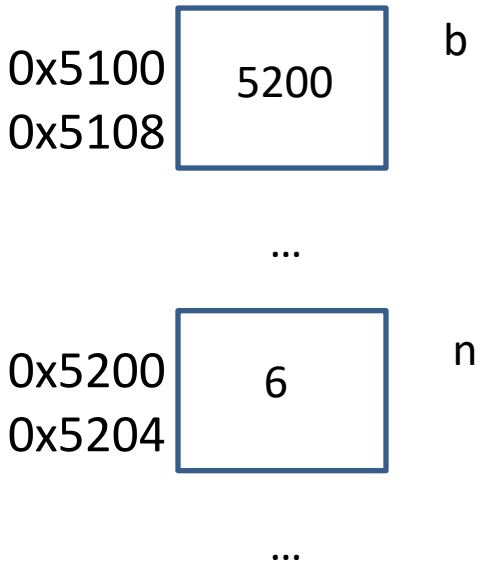
```
int main(void) {
    int *b;
    int n;
    n = 5;
    b = &n;
```



Pointers

- Create an integer pointer variable and set it

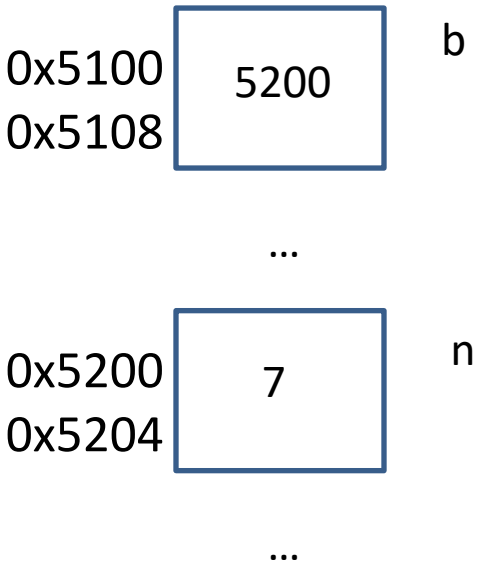
```
int main(void) {
    int *b;
    int n;
    n = 5;
    b = &n;
    n = 6;
    *b += 1;
    *b = 2 * (*b);
```



Pointers

- Create an integer pointer variable and set it

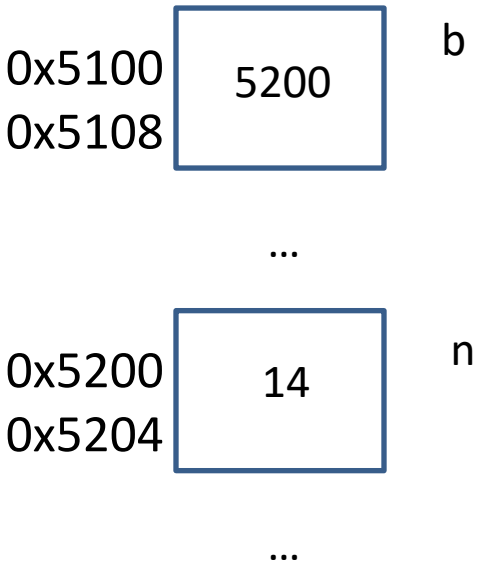
```
int main(void) {
    int *b;
    int n;
    n = 5;
    b = &n;
    n = 6;
    *b += 1;
    *b = 2 * (*b);
```



Pointers

- Create an integer pointer variable and set it

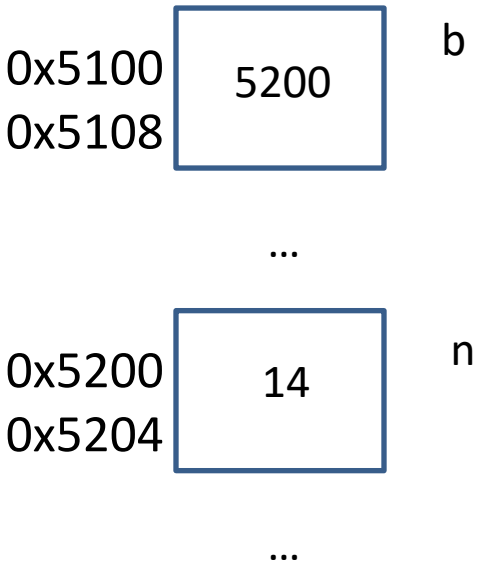
```
int main(void) {
    int *b;
    int n;
    n = 5;
    b = &n;
    n = 6;
    *b += 1;
    *b = 2 * (*b);
```



Pointers

- Create an integer pointer variable and set it

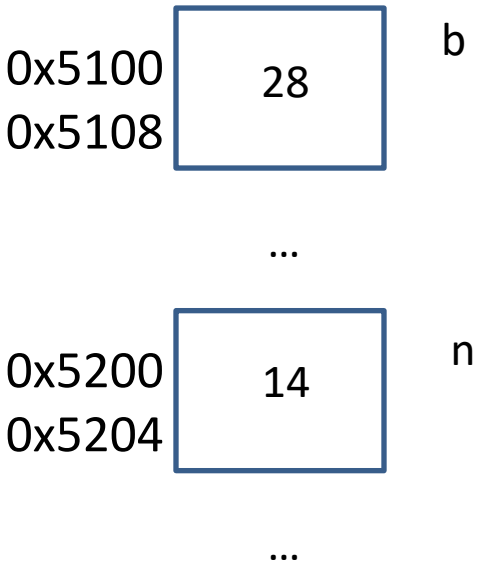
```
int main(void) {
    int *b;
    int n;
    n = 5;
    b = &n;
    n = 6;
    *b += 1;
    *b = 2 * (*b);
    b = 2 * (*b);
```



Pointers

- Create an integer pointer variable and set it

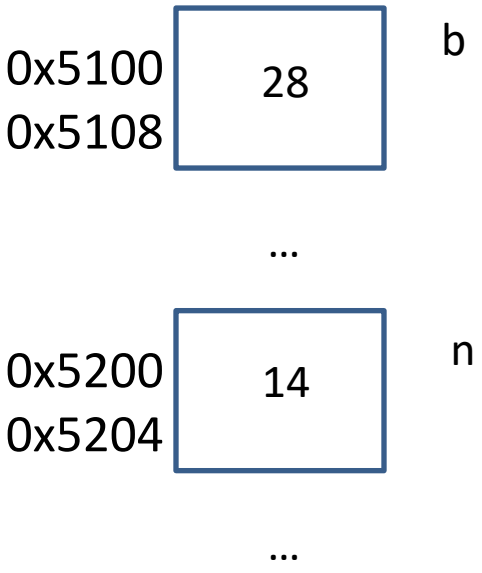
```
int main(void) {
    int *b;
    int n;
    n = 5;
    b = &n;
    n = 6;
    *b += 1;
    *b = 2 * (*b);
    b = 2 * (*b);
```



Pointers

- Create an integer pointer variable and set it

```
int main(void) {
    int *b;
    int n;
    n = 5;
    b = &n;
    n = 6;
    *b += 1;
    *b = 2 * (*b);
    b = 2 * (*b);
```



ptr0.c shows seg fault accessing *b

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

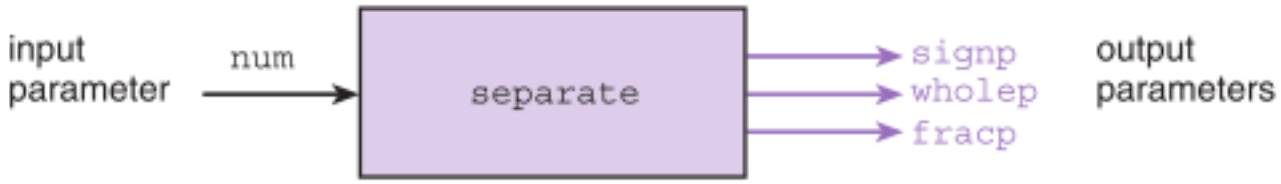


FIGURE 6.6

Parameter Correspondence for separate(value, &sn, &whl, &fr);

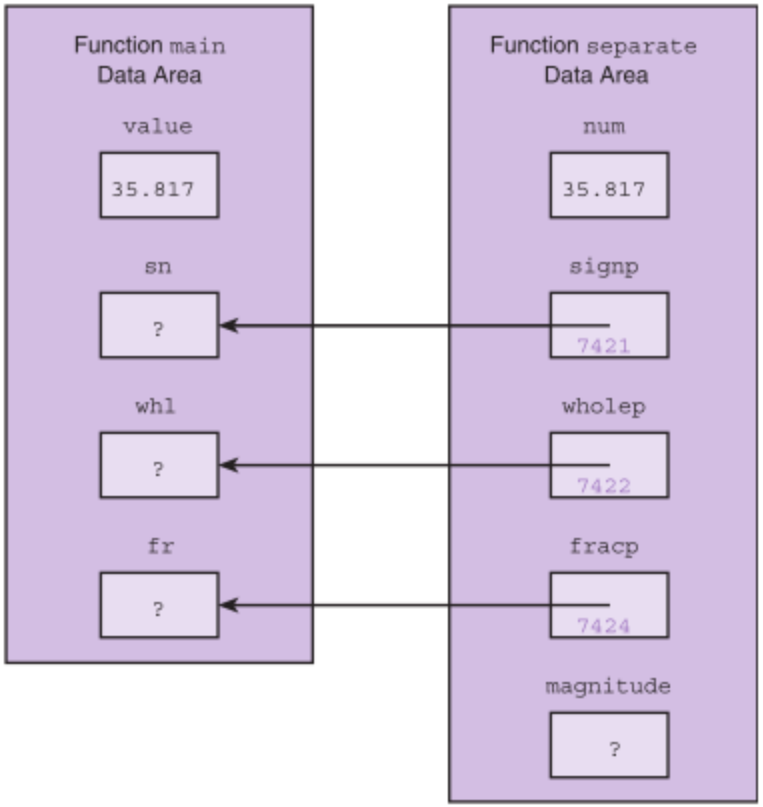


TABLE 6.2 Effect of & Operator on the Data Type of a Reference

Declaration	Data Type of x	Data Type of &x
`char x`	`char`	`char * (pointer to char)`
`int x`	`int`	`int * (pointer to int)`
`double x`	`double`	`double * (pointer to double)`

Meaning of Symbol *

- binary operator for multiplication
- “pointer to” when used when declaring a variable or a function parameters
- unary indirection operator in a function body

Multiple Calls to a Function with Input/Output Parameters

An example of sorting data

TABLE 6.3 Trace of Program to Sort Three Numbers

Statement	num1	num2	num3	Effect
`scanf("...", &num1, &num2, &num3);`	7.5	9.6	5.5	Enters data
`order(&num1, &num2);`				No change
`order(&num1, &num3);`	5.5	9.6	7.5	Switches num1 and num3
`order(&num2, &num3);`	5.5	7.5	9.6	Switches num2 and num3
`printf("...", num1, num2, num3);`				Displays 5.5 7.5 9.6

FIGURE 6.8

Data Areas After `temp = *smp;` During Call `order(&num1, &num3);`