

CSE 142 Computer Programming I

Pointer Parameters

or... Mysteries of scanf Revealed!

© 2000 UW CSE

M-1

Overview

Concepts this lecture

Function parameters

Call by value (review)

Pointer parameters - call by reference

Pointer types

& and ***** operators

M-2

Reading

6.1 Output (pointer) Parameters

6.2 Multiple calls to functions with output parameters

6.3 Scope of Names

6.4 Passing Output Parameters to other functions

6.6, 6.7 Debugging and common programming errors

M-3

What Does This Print?

```
/* change x and y */  
void move_one ( int x, int y ) {  
    x = x - 1;  
    y = y + 1;  
}  
  
int main ( void ) {  
    int a, b ;  
    a = 4 ; b = 7 ;  
    move_one(a, b) ;  
    printf("%d %d", a ,b);  
    return 0;  
}
```

Output:

3 8 ?

4 7 ?

M-4

Function Call Review

Remember how function calls are executed:

Allocate space for parameters and local variables

Initialize parameters by *copying* argument values

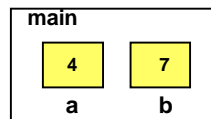
Begin execution of the function body

Trace carefully to get the right answer

M-5

Trace

```
/* change x and y */  
void move_one ( int x, int y ) {  
    x = x - 1;  
    y = y + 1;  
}  
  
int main ( void ) {  
    int a, b ;  
    a = 4 ; b = 7 ;  
    move_one(a, b) ;  
    printf("%d %d", a ,b);  
    return 0;  
}
```



Output: 4 7

M-6

Call By Value is Not Enough

Parameters are initialized with copies of the arguments; there is *no further connection!*

If a function changes its parameters, it affects the local copy *only*.

To change the arguments in the caller, the function needs access to the *locations* of the arguments, not just their *values*.

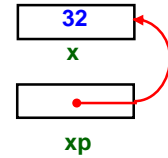
M7

But... why would we ever want to change the parameters?

New Type: Pointer

A **pointer** contains a **reference** to another variable; that is, a pointer contains the memory address of a variable.

the variable xp has type **pointer to int**
(often written: xp has type **int***)



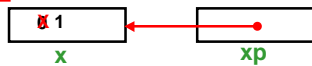
M8

Declaring and Using a Pointer

```
int x;           /* declares an int variable */
int *xp;        /* declares a pointer to int */
```

If the address of x is stored in xp, then:

```
*xp = 0;        /* Assign integer 0 to x */
*xp = *xp + 1; /* Add 1 to x */
```



M9

Pointer Solution to *move one*

```
void move_one ( int * x_ptr, int * y_ptr ) {
```

```
    *x_ptr = *x_ptr - 1;
```

```
    *y_ptr = *y_ptr + 1;
```

```
}
```

```
int main ( void ) {
```

```
    int a, b;
```

```
    a = 4; b = 7;
```

```
    move_one(&a, &b);
```

```
    printf("%d %d", a, b);
```

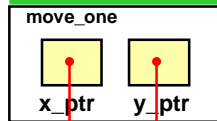
```
    return 0;
```

```
}
```

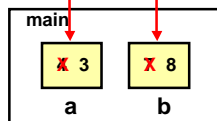
The & operator in front of a variable name creates a pointer to that variable

M10

Trace



```
void move_one (
    int * x_ptr,
    int * y_ptr ) {
    *x_ptr = *x_ptr - 1;
    *y_ptr = *y_ptr + 1;
}
```

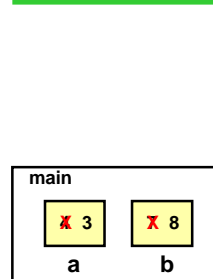


```
a = 4; b = 7;
move_one( &a, &b );
```

Output:

M11

Trace



```
void move_one (
    int * x_ptr,
    int * y_ptr ) {
    *x_ptr = *x_ptr - 1;
    *y_ptr = *y_ptr + 1;
}
```

```
a = 4; b = 7;
move_one( &a, &b );
```

Output: 3 8

M12

Aliases

***x_ptr** and ***y_ptr** act like aliases for the variables **a** and **b** in the function call.

When you change ***x_ptr** and ***y_ptr** you are changing the values of the caller's variables.

To create these aliases you need to use **&a**, **&b** in the call.

M-13

Pointer Types

Three new types:

int * "pointer to int"
double * "pointer to double"
char * "pointer to char"

These are all different:

pointer to a char can't be used where a pointer to an int is needed.

pointer to double can't be used where a double is needed!

M-14

Pointer Operators

Two new (unary) operators:

- &** "address of"
& can be applied to any variable (or param)
- *** "location pointed to by"
* can be applied only to a pointer

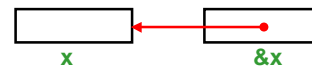
Keep track of the types:

if **x** has type **double**,
&x has type "pointer to double" or "double **"

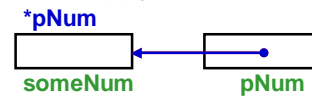
M-15

Warning: I'm not in your course packet!

Follow the Bouncing Arrow:



Apply **&**



Apply ***** or *dereference*

M-16

Vocabulary

Dereferencing or **indirection**:

following a pointer to a memory location

The book calls pointer parameters "**output parameters**":

can be used to provide a value ("input") as usual, **and/or store a changed value** ("output")

Don't confuse with printed output (**printf**)

M-17

Why Use Pointers?

For parameters:

in functions that need to change their actual parameters(such as **move_one**)

in functions that need multiple "return" values (such as **scanf**)

These are the only uses in this course

In advanced programming, pointers are used to create **dynamic** data structures.

M-18

scanf Revisited

Now we can make sense out of the punctuation in *scanf*

```
int x,y,z;
```

```
scanf("%d %d %d", x, y, x+y); NO!  
scanf("%d %d", &x, &y); YES! Why?
```

M19

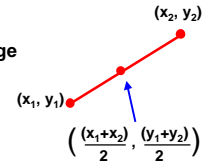
Example: Midpoint Of A Line

Problem: Find the midpoint of a line segment.

Algorithm: find the average of the coordinates of the endpoints:

$$\begin{aligned} x_{mid} &= (x_1+x_2)/2.0; \\ y_{mid} &= (y_1+y_2)/2.0; \end{aligned}$$

Programming approach: We'd like to package this in a function



M20

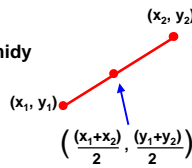
Function Specification

Function specification: given endpoints (x_1, y_1) and (x_2, y_2) of a line segment, store the coordinates of the midpoint in (mid_x, mid_y)

Parameters:

$x_1, y_1, x_2, y_2, mid_x,$ and mid_y

The (mid_x, mid_y) parameters are being altered, so they need to be pointers



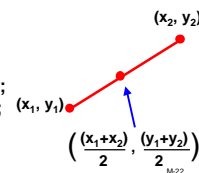
M21

Midpoint Function: Code

```
void set_midpoint( double x1, double y1,  
                 double x2, double y2,  
                 double * midx_p, double * midy_p )
```

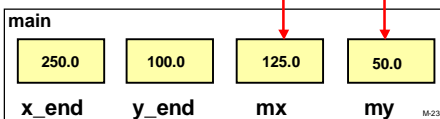
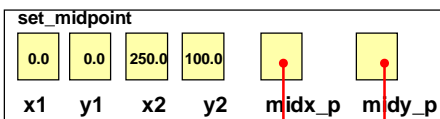
```
{  
    *midx_p = (x1 + x2) / 2.0;  
    *midy_p = (y1 + y2) / 2.0;  
}
```

```
double x_end, y_end, mx, my;  
x_end = 250.0; y_end = 100.0;  
set_midpoint(0.0, 0.0,  
             x_end, y_end,  
             &mx, &my);
```



M22

Trace



M23

Example: Gameboard Coordinates

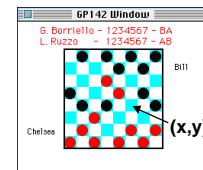
Board Coordinates

row, column (used by players)

Screen Coordinates

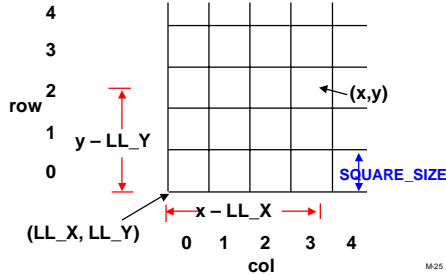
x, y (used by graphics package)

Problem: convert (x, y) to (row, col)



M24

Coordinate Conversion: Analysis



M25

Coordinate Conversion: Code

```
#define LL_X      40
#define LL_Y      20
#define SQUARE_SIZE 10

void screen_to_board (
    int screenx, int screeny, /* coords on screen */
    int * row_p, int * col_p) /* position on board */
{
    *row_p = (screeny - LL_Y) / SQUARE_SIZE;
    *col_p = (screenx - LL_X) / SQUARE_SIZE;
}

screen_to_board (x, y, &row, &col);
```

M26

Problem: Reorder

Suppose we want a function to arrange its two parameters in reverse numeric order.

Example:

-1, 5 need to be reordered as 5, -1
12, 3 is already in order (no change needed)

Parameter analysis: since we might change the parameter values, they have to be pointers

This example is a small version of a very important problem in computer science, called "sorting" ^{M27}

Code for Reorder

```
/* ensure *p1 >= *p2,
interchanging values if needed
*/
```

```
void reorder(int *p1, int *p2) {
    int tmp;
    if (*p1 < *p2) {
        tmp = *p1;
        *p1 = *p2;
        *p2 = tmp;
    }
}
```

These 3 lines can be said to "swap" two values

M28

swap as a Function

```
/* interchange *p and *q */
void swap ( int * p, int * q) {
    int temp ;
    temp = *p ;
    *p = *q ;
    *q = temp ;
}
```

```
int a, b ;
a = 4; b = 7;
...
swap (&a, &b) ;
```

M29

Reorder Implemented using swap

```
/* ensure *p1 >= *p2, interchanging values if
needed */
void reorder(int *p1, int *p2) {
    if (*p1 < *p2)
        swap( _____ , _____ );
}
```

What goes in the blanks?

M30

Pointer Parameters (Wrong!)

Normally, if a pointer is expected, we create one using &:

```
/* ensure *p1 >= *p2, interchanging values if
needed */
void reorder(int *p1, int *p2) {
    if (*p1 < *p2)
        swap( &p1 , &p2);
}
```

But that can't be right - p1 and p2 are already pointers!
What are the types of expressions &p1 and &p2?

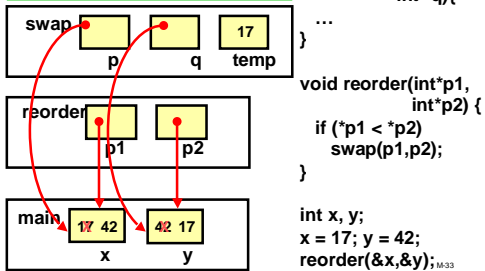
Pointer Parameters (Right!)

Right answer: if the types match (int *), we use the pointers directly

```
/* ensure *p1 >= *p2, interchanging values if
needed */
void reorder(int *p1, int *p2) {
    if (*p1 < *p2)
        swap( p1 , p2 );
}
```

M32

Trace



Pointers and scanf Once More

Problem: User is supposed to enter 'y' or 'n', and no other answer is acceptable. Read until user enters 'y' or 'n' and give character to caller

```
void Read_y_or_n(char *chp) {
    ...
}

int main(void) {
    char ch;
    Read_y_or_n(&ch);
    ...
}
```

M34

Pointers and scanf Once More

/* read until user enters 'y' or 'n', give char to caller */

```
void Read_y_or_n(char *chp) {
    printf("Enter an 'y' or a 'n'.\n");
    scanf("%c", chp);
    while ( *chp != 'y' && *chp != 'n' ) {
        printf ("Sorry, try again\n");
        scanf("%c", chp);
    }
}

int main(void) {
    char ch;
    Read_y_or_n(&ch);
    ...
}
```

No '&'!

M35

Wrapping Up

Pointers are needed when the parameter value may be changed
& creates a pointer
* dereferences the value pointed to

This completes the technical discussion of functions in C for this course

Learning how to design and use functions will be a continuing concern in the course

M36

QOTD: When an Irresistible Force Meets an Immutable Object

Does the function call to swap inside the function below change the value of px?

-If so, to what?

-If not, is it possible for any function call to change px's value? How?

```
void tryIt(int * px, int * py)
{
    swap(px, py);
}
```

M37

WARNING: This may be harder than it seems!