

CSE / ENGR 142 Programming I

Pointers and Output Parameters

© 2000 UW CSE

5/7/00 J-1

Chapter 6

- 6.1 Output 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

5/7/00 J-2

Review: Function Terminology

```
0! is 1
1! is 1
2! is 1 * 2
3! is 1 * 2 * 3
...

```

```

int factorial ( int n ) {
    int product, i;
    product = 1;
    for ( i = n; i > 1; i = i - 1 ) {
        product = product * i;
    }
    return (product);
}
    
```

Annotations:

- function name: `factorial`
- [formal] parameter: `int n`
- local variables: `product, i`
- return type & value: `int`

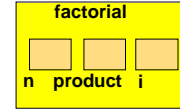
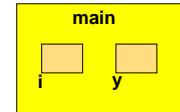
5/7/00 J-3

Review: Local Variables

```

int
main(void)
{
    int i, y;

    i = 3;
    y = factorial ( i + 1 );
    return (0);
}
    
```



5/7/00 J-4

Local Variables: Summary

- Parameters and variables declared in a function are **local** to it:
 - cannot be directly accessed by other functions
- Allocated (created) on function entry.
- De-allocated (destroyed) on function return.
- Formal parameters are initialized by **copying value** of argument in function call.
- Reminder: *no global variables in 142!*

5/7/00 J-5

Call by Value

```

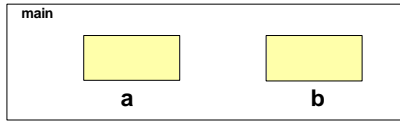
void move_one ( int x, int y ) {
    x = x - 1;
    y = y + 1;
}
    
```

```

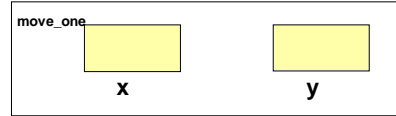
int main ( void ) {
    int a, b;
    a = 4; b = 7;
    move_one(a, b);
    printf(“%d %d”, a, b);
    return (0);
}
    
```

5/7/00 J-6

Trace



5/7/00 J-7

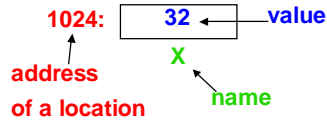


5/7/00 J-8

Values vs. Locations

Problem: for *move_one* (*a,b*) to do what we want, it needs access to the **locations** of *a* and *b* as well as to their **values**.

Recall: variables name memory **locations**, which hold **values**.



5/7/00 J-9

New Type: Pointer

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



5/7/00 J-10

Declaring and Using a Pointer

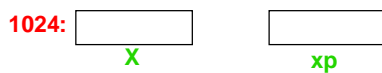
`int x;` /* declares an int variable */

`int *xp;` /* declares a pointer to int */

If somehow, *xp* gets the address of *x*, then:

`*xp = 0;` /* Assign integer 0 to x */

`*xp = *xp + 1;` /* Add 1 to x */



5/7/00 J-11

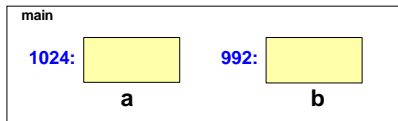
Pointer Solution to *move_one*

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

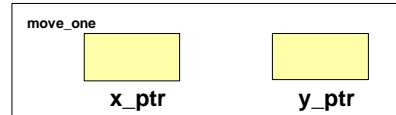
```
int main ( void ) {
    int a, b;
    a = 4; b = 7;
    move_one( &a, &b );
    printf( "%d %d", a, b );
}
```

5/7/00 J-12

Trace



5/7/00 J-13



5/7/00 J-14

Addresses and Pointers

Three new types:

int * “pointer to int”
double * “pointer to double”
char * “pointer to char”

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

5/7/00 J-15

Vocabulary

•Dereferencing or indirection:

–following a pointer to a memory location

•Output parameter:

–a pointer parameter of a function
–can be used to provide a value (“input”) as usual, **and/or store a changed value** (“output”)
–Don’t confuse with printed output (printf)

5/7/00 J-16

scanf Revisited

```
int x,y,z;  
printf(“%d %d %d”, x, y, x+y);
```

What about *scanf*?

```
scanf(“%d %d %d”, x, y, x+y); NO!  
scanf(“%d %d”, &x, &y); YES! (why?)
```

5/7/00 J-17

Why Use Pointers?

•In CSE142, only used for output parameters:

- functions that need to change their actual parameters, e.g., *move_one*
- to get multiple “return” values
 - e.g., *scanf()*

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

5/7/00 J-18

Sort Two Integers

```
/* read in and sort 2 integers */
```

```
int c1, c2, temp ;  
printf ( "Enter 2 integers: " );  
scanf ( "%d%d", &c1, &c2 );  
/* At this point the 2 values may be in either order*/  
if ( c2 < c1 ) { /* swap if out of order */
```

```
    temp = c1 ;  
    c1 = c2 ;  
    c2 = temp ;
```

```
}  
/* At this point c1 <= c2 (guaranteed) */
```

5/7/00 j-19

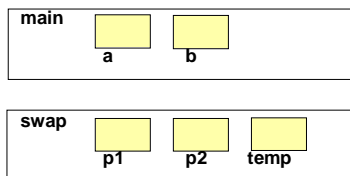
swap as a Function

```
void swap ( int *p1, int *p2 ) {  
    int temp ;  
    temp = *p1 ;  
    *p1 = *p2 ;  
    *p2 = temp ;  
}
```

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

5/7/00 j-20

Trace



5/7/00 j-21

Aliases

A way to think about pointer parameters:

*p1 and *p2 act like **aliases** for the variables in the call of swap.

When you change *p1 and *p2 you are changing the values of the variables in the call.

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

Otherwise, calls are like Xerox copies (except for arrays which also use aliases)

5/7/00 j-22

Sorting

Problem: Sort 3 integers

Three-step algorithm:

1. Read in three integers: x, y, z
2. Put smallest in x:
Swap x, y if necessary; then swap x, z, if necessary.
3. Put second smallest in y:
Swap y, z, if necessary.

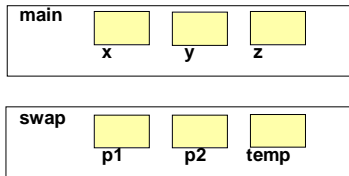
5/7/00 j-23

Sort 3 Integers

```
int main (void) {  
    int x, y, z, scanStatus ;  
    ...  
    scanStatus = scanf ("%d%d%d", &x, &y, &z) ;  
    if scanStatus == 3 {  
        if ( x > y ) swap (&x, &y) ;  
        if ( x > z ) swap (&x, &z) ;  
        if ( y > z ) swap (&y, &z) ;  
    }  
    ...
```

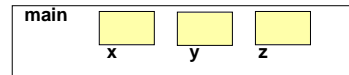
5/7/00 j-24

Trace

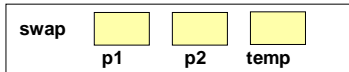


5/7/00 j-25

Trace



5/7/00 j-26



5/7/00 j-27

sort3 as a Function

```
/* interchange values as needed to establish */  
/* *xp <= *yp <= *zp */  
void sort3(int *xp, int *yp, int *zp) {  
    if ( *xp > *yp) swap(xp, yp);  
    if ( *xp > *zp) swap(xp, zp); ← NO &s!  
    if ( *yp > *zp) swap(yp, zp);  
}  
  
int main(void) {  
    int x, y, z;  
    ... /*scanf the values, then: */  
    sort3(&x, &y, &z);  
    ...  
}
```

5/7/00 j-28

Why no & in swap call?

Real reason

xp and yp are **already** pointers that refer to the variables that we want to swap

Alternative explanation using alias idea

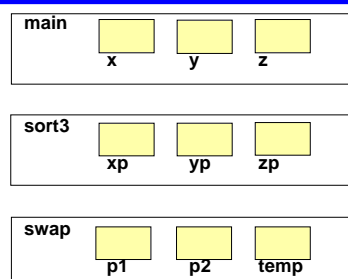
*xp and *yp are aliases for the variables we want to swap

We want to allow swap to use aliases for *xp and *yp so we should use &(*xp) and &(*yp) in the call

BUT xp==&(*xp) and yp==&(*yp) !!!!

5/7/00 j-29

Trace



5/7/00 j-30

C is "strongly typed"

```
int i; int *ip;
double x; double *xp;

...
x = i;      /* no problem */
i = x;      /* not recommended */

ip = 30;    /* No way */
ip = i;     /* Nope */
ip = &i;    /* just fine */
ip = &x;    /* forget it! */
xp = ip;    /* bad */
&i = ip;    /* meaningless */
```

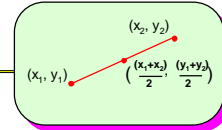
5/7/00 J31

Midpoint Of A Line

/* Given 2 endpoints of a line, "return" coordinates of midpoint */

```
void set_midpoint(
    double x1, double y1,      /* 1st endpoint */
    double x2, double y2,      /* 2nd endpoint */
    double * midx_p, double * midy_p) /* Pointers to midpoint */
{
    *midx_p = (x1 + x2) / 2.0;
    *midy_p = (y1 + y2) / 2.0;
}
```

```
double x_end, y_end, mx, my;
...
set_midpoint(0.0, 0.0, x_end, y_end, &mx, &my);
```



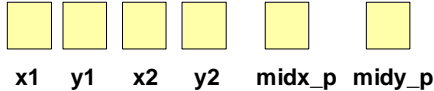
5/7/00 J32

Trace

main



set_midpoint



5/7/00 J33

Example: Coordinates

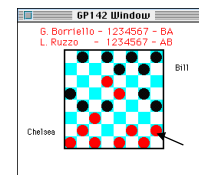
Board Coordinates

row, column

Screen Coordinates

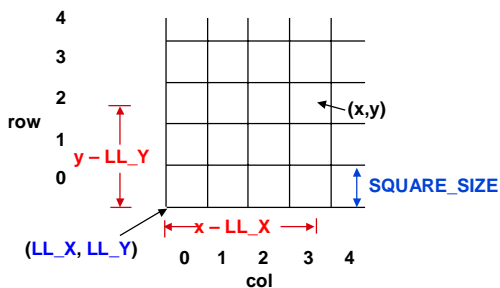
x, y

used by graphics package



5/7/00 J34

Coordinate Conversion



5/7/00 J35

Coordinate Conversion

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

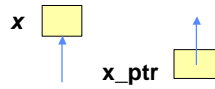
```
void screen_to_board (
    int screenx, int screeny, /* coordinates on screen */
    int * row_p, int * col_p) /* position on board */
{
    *row_p = (screeny - LL_Y) / SQUARE_SIZE;
    *col_p = (screenx - LL_X) / SQUARE_SIZE;
}
```

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

5/7/00 J36

Pointers vs. Values

	in caller	in callee
Declaration:	<code>int x</code>	<code>int * x_ptr</code>
To get the address of x:	<code>&x</code>	<code>x_ptr</code>
To get the value of x:	<code>x</code>	<code>*x_ptr</code>

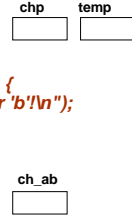


5/7/00 J-37

& in scanf again

```
void Read_a_or_b(char *chp) {
    char temp;
    printf("Enter an 'a' or a 'b'.\n");
    scanf("%c", &temp);
    while (temp != 'a' && temp != 'b') {
        printf("\nNope, it must be 'a' or 'b'!\n");
        scanf("%c", &temp);
    }
    *chp = temp;
}

int main(void) {
    char ch_ab;
    Read_a_or_b(&ch_ab);
    ...
}
```



5/7/00 J-38

Moral:
Wrong rule: "always use &'s in scanf"
Right rule: "always use addresses in scanf"

& in scanf again

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

int main(void) {
    char ch_ab;
    Read_a_or_b(&ch_ab);
    ...
}
```

No '&'!



5/7/00 J-39