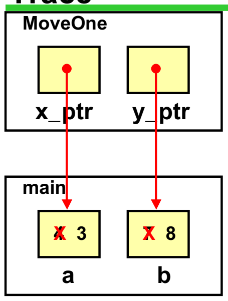
CSE 142
Computer Programming I

Pointer Parameters

Trace



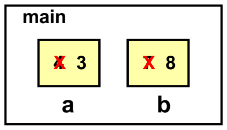
```
void MoveOne (
      int  * x_ptr,
      int  * y_ptr ) {
   *x_ptr = *x_ptr - 1;
   *y_ptr = *y_ptr + 1;
}
```

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

Output:

Trace

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



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

Output: 3 8

Pointer Types

Three new types:

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

These are all different - a pointer to a char can't be used if the function parameter is supposed to be a pointer to an int, for example.

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 *"

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)

Why Use Pointers?

For parameters:

- in functions that need to change their actual parameters (such as MoveOne)
- in functions that need multiple "return" values (such as scanf)

These are the only uses in this course

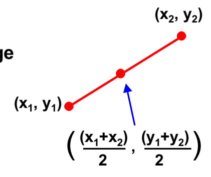
In advanced programming, pointers are used to create dynamic data structures.

Example: Midpoint Of A Line

Problem: Find the midpoint of a line segment.

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

```
xmid = (x1+x2)/2.0;
ymid = (y1+y2)/2.0;
```



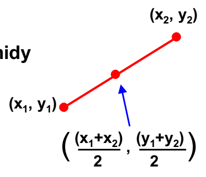
Programming approach: We'd like to package this in a function

Function Specification

Function specification: given endpoints (x1,y1) and (x2,y2) of a line segment, store the coordinates of the midpoint in (midx, midy)

Parameters:
x1, y1, x2, y2, midx, and midy

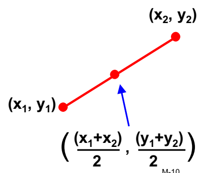
The (midx,midy) parameters are being altered, so they need to be pointers



Midpoint Function: Code

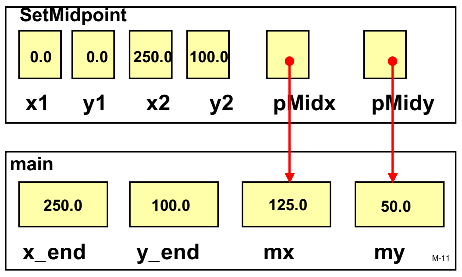
```
void SetMidpoint( double x1, double y1,
                  double x2, double y2,
                  double * pMidx, double * pMidy ) {

    *pMidx = (x1 + x2) / 2.0;
    *pMidy = (y1 + y2) / 2.0;
}
```

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



Trace

```
SetMidpoint(0.0, 0.0,
            x_end, y_end,
            &mx, &my);
```



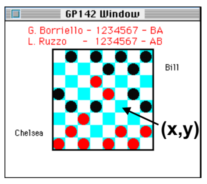
Example II: Gameboard Coordinates

Board Coordinates

row, column (used by players)

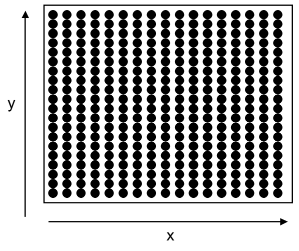
Screen Coordinates

x, y (used by graphics package)



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

Screen Coordinates



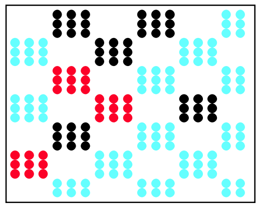
The screen is composed of *pixels* arranged in a grid.

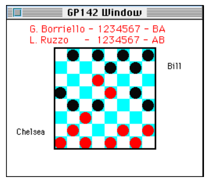
A *screen coordinate* is an (x,y) position naming a pixel.

(*Screen resolution* is the number of pixels your monitor can display. E.g., "800 x 600" or "1280 x 1024")

M-13

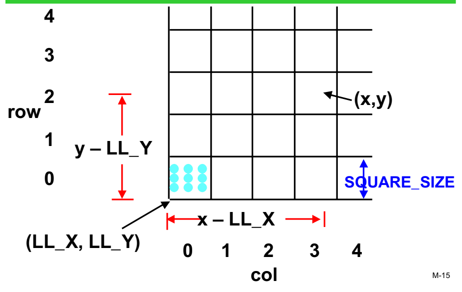
Pixels ⇒Images





M-14

Coordinate Conversion: Analysis



M-15

Coordinate Conversion: Code

```
int LL_X = 40;
int LL_Y = 20;
int SQUARE_SIZE =10;

void ScreenToBoard (
   int screenx, int screeny,    /* coords on screen */
   int * pRow,  int * pCol)     /* position on board */
{
   *pRow =  (screeny - LL_Y) / SQUARE_SIZE;
   *pCol  =  (screenx - LL_X) / SQUARE_SIZE;
}
_______________________________________
ScreenToBoard (x, y, &row, &col);
```

M-16

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"

M-17

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

M-18

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) ;
```

M-19

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?

M-20

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?

M-21

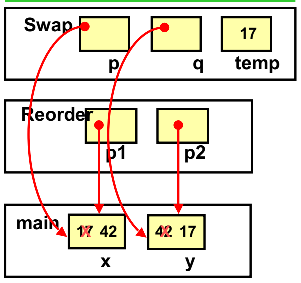
Pointer Parameters (Right!)

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 );
}
```

M-22

Trace



```
void Swap(int *p,
             int *q){
   ...
}

void reorder(int*p1,
               int*p2) {
   if (*p1 < *p2)
      swap(p1,p2);
}

int x, y;
x = 17; y = 42;
reorder(&x,&y);
```

M-23

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

M-24