# Before the lecture begins …

- Make sure you have Homework 2 turned in.
- Please update your framework at http://pclx.com/itcc for a version that *will* work on your compiler. Dev-C++ (g++) is now supported as well thanks to one of your classmates.

# Intro to C++

## Lecture 5

Functions, Arrays, Bresenham's Line Algorithm

## Functions

- ***Functions*** are groups of commands that can be executed from anywhere in your program. In math, a ***function*** might look like this:

$$f(x) = 5x + 2$$

- Notice that it has an ***input***, *x*, and an ***output***, *f*(*x*). If I write *f*(5), then the ***function*** will ***return*** 27. This is how functions work in C++ too.

## Function Requirements

- In C++, every ***function*** has:
  - *Input*: Variables needed to make the function do what it has to do.
  - *Output*: Variable(s) that are returned from the function once it is finished processing.

- Sometimes, however, functions do not have *inputs* or *outputs*. When this occurs, we say that its input or output section is ***void***.

## Modular Code

- Another way to look at functions is that they help to *modularize* programming – break large programs down into simpler tasks.
- This should always be your goal so that a huge program to write doesn’t overwhelm you.

# Modular Pong

Remember how long the Pong program was and how confusing it seemed? Isn't it a lot easier to understand what's going on now?

```
Initialize();

while (    (itcc->Flip())        // Makes back buffer visible
        && (isRunning))          // While the game is still running
{
      GetInput();
      DrawGraphics();
      MoveBall();
}

Shutdown();
```

## Functions

- Generally, a function looks like this:

  *Output* FuncName(*Inputs*)

- Let's take a look at a function we're used to:

```
void SetPixel (unsigned long x, unsigned long y, unsigned long color);
```

- Here, we see that SetPixel does not have any outputs (void), but that it does have three inputs – *x*, *y*, and *color*. We can call the function by replacing *x*, *y*, and *color* with either values or variables.

**Note**: Technically, a function that does not return anything is called a *subroutine*.

# Function Declarations

- Functions must be ***declared*** and then ***defined*** before they can be used in your program.
- To ***declare*** a function, just write the function ***prototype*** at the top of a source file, terminated with a semicolon. This tells the compiler that this function is valid syntactically and that it will be ***defined*** later in your code:

```
void SayHi(void);
```

# Function Definitions

- A function ***definition*** is the function prototype followed by a pair of braces with code in-between (the *function body*).
- Note that the variables in the *input* can be used within the function body. These *input* variables are called *parameters*, and the variable used as the *parameter* takes on the value that was passed to the function.

```
void SayHi(unsigned long hiColor)
{
	itcc->Text(“Hi!”, 0, 0, hiColor);
}
```

# Returns

- Functions can also have ***return values***. Whenever this occurs, imagine that when you call the function, the entire function is **replaced** by its ***return value***.

```
long AddTwo (long num1, long num2)
{
      long sum = num1 + num2;
      return sum;
}
```

- Note that functions can only return one value (although there are ways around this discussed in future lectures).
- Also note that we use the return statement to return a value of the type listed in the function prototype.

## Example

```
long Power (long base, long exponent);              // Declaration

int main(void)
{
     long numBits = 10, maxNumber = 0;

     maxNumber = Power(2, numBits);                 // Call it!
     …
}


long Power (long base, long exponent)               // Definition
{
     long multNum = 0, result = 1;
     for (multNum=0; multNum<exponent; multNum++)
     {
             result *= base;
     }

     return result;
}
```

## int main(void)

- Wow! Our beloved main is a ***function***! Note that it ***returns*** a value to the operating system telling it whether we terminated correctly or not, and that it does not accept any parameters.

- Note that there is another version of the main function that has parameters taken in from the command line when you first run your program.

## Arrays

• Sometimes we have lots of objects on the screen, and we want to store all of their positions at once.

• One way of doing this would be to make lots of different variables.

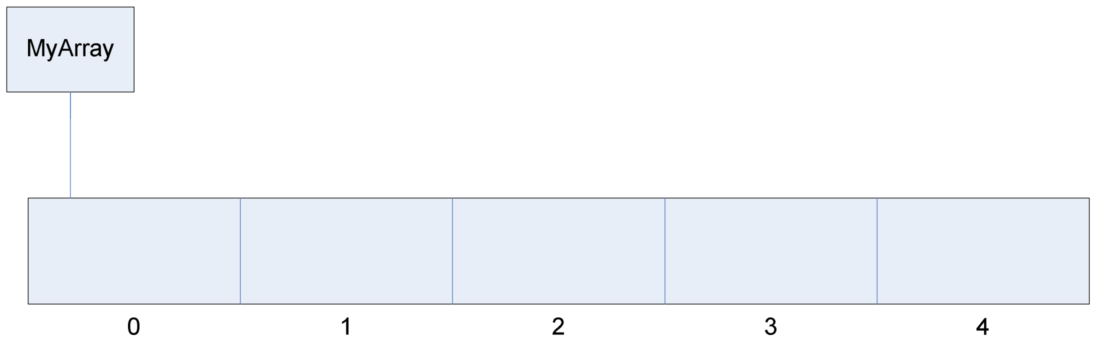
• But then what if we add a new asteroid? We'd have to physically go into our program and add a bunch of variables just to accommodate this! That's why we have ***arrays*** – contiguous blocks of memory that can store variables right after each other that can be reached via an ***index***.

# Arrays

• We ***declare*** an array just like any other variable:

```
long MyArray[5];
```

• This ***declaration*** reserves five contiguous blocks of 32-bits of memory for our use:



## Arrays

• We can access any of these five memory locations by using an ***index*** from the first element:

```
myArray[2] = 5;
```

• Note that here, we set the third element in our array equal to the value ‘5’. We can access this element in the same manner:

```
if (myArray[2] == 5)
{
}
```

## Initializing Arrays

- You can also ***initialize*** an array, giving each array element a value:

```
long xPos[10] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
```

- This can only be done when first initializing the array – otherwise, the elements must be individually set:

```
for (int i=0; i<10; i++)
{
	xPos[i] = 0;
}
```

## Array Uses

• What are arrays useful for? What if you’re making an explosion of pixels and need to keep track of every pixel’s location? What if you have lots of enemies on the screen that you need to keep track of?

• The ability to loop through an array using a for loop is much easier than having to separately declare a new variable for every single item on the screen despite their similarities.

## Passing Arrays to Functions

- Here is how we might pass an array to a function:

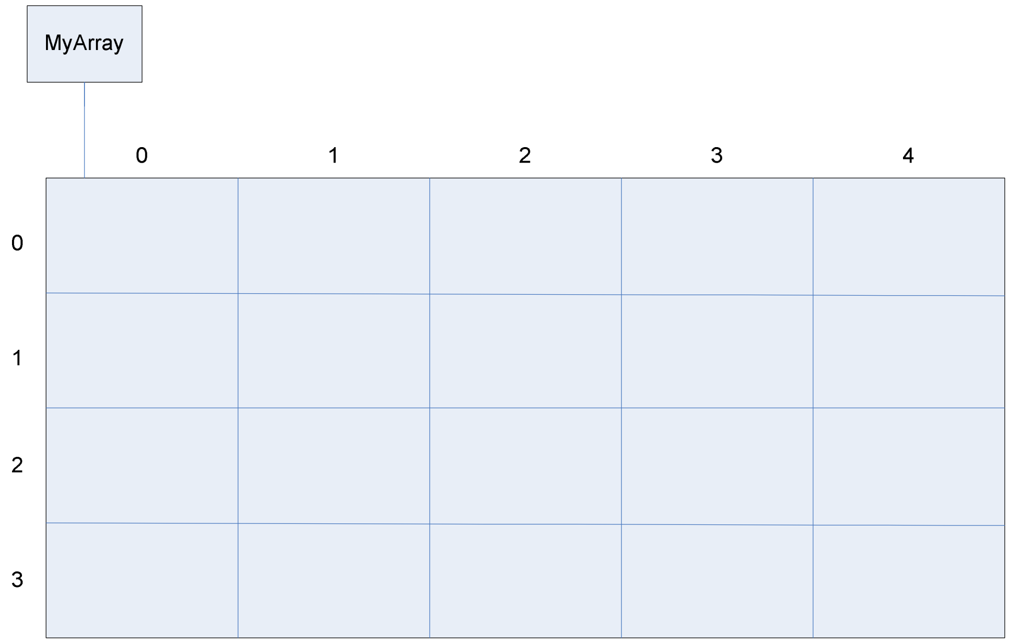
```
void DrawEnemies(long x[], long y[], long numEnemies);
```

- It would be called as follows:

```
long xPos[5], yPos[5];
DrawEnemies(xPos, yPos, 5);
```

# Multidimensional Arrays

- A multidimensional array looks conceptually like the following:



## Two-Dimensional Arrays

- In reality, it is exactly like the linear one-dimensional array. The computer will do the math to figure out where you're referring to.
- Declaration: long myArray[4][5]; allocates the previous memory with four rows and five columns.
- Accessing: myArray[2][1] refers to the square that is the intersection of the third row and the second column.

## Example

```
int polyPoints[3][2] = { {5, 2}, {2, 1}, {5, 3} };

for (int y=0; y<3; y++)
{
   for (int x=0; x<2; x++)
   {
        polyPoints[y][x] = 0;
   }
}
```

# TODO

- **EVERYONE**: PLEASE DOWNLOAD THE LATEST FRAMEWORK VERSION FROM THE SITE AND INSTALL IT. DO NOT USE OLDER FRAMEWORK VERSIONS. Instructions will be posted on installing the new framework after this lecture (they are the same as for the HW3 framework).
- Download ITCC_HW3.zip from the site (will be available soon) http://www.pclx.com/itcc/, and complete the homework exercises, emailing them (FOR THIS WEEK ONLY) to itcc_teachers@pclx.com. Please do not resubmit solutions, even if they are revised. All homework must be submitted by 6:00am PST Wednesday, July 14.
- Some illustrative examples of the topics in this lecture are given in ITCC_HW3.zip.
- Look over the slides for the sixth lecture.
- If you finish with the homework, experiment!