Before the lecture begins …

- Finish Homework 3! Send us questions – we have answers!

Intro to C++

Lecture 7

Pointers II, File I/O

Pointers Review

- The smallest addressable unit of RAM is the ***byte*** (8 bits). A ***pointer*** is a 32-bit value that holds a memory address.
- For instance, the following initializes a pointer to point to some memory address which has unknown contents (hence the void).

```
void *videoBuffer = 0xA0000000;
```

Pointers Review

- We can assign a new memory address in a pointer simply by using the assignment operator:

```
videoBuffer = 0xA0000000;
```

- If we know what type of data is stored at the pointer's address, we can initialize the pointer using the data type, allowing us to conveniently access the data:

```
char *response;
```

Pointers Review

- We can obtain the address of a variable by using the ampersand (&), and we can access the ***contents*** of a memory address by using an asterisk (*):

```
char response = 'y';          // set response to be yes
char *ptrResponse = NULL;
ptrResponse = &response;      // ptrResponse "points to" response
*ptrResponse = 'n';           // now response == 'n' too!
```

- Do you see how the last line modifies the contents of 'response' too?

Pointers Review

- Pointer arithmetic can also be used if the data type the pointer references is specified. Note that adding one to the pointer advances by the number of bytes contained in the data type:

```
long myVal[3] = {100, 200, 300};
long *ptrLong = &myVal[0];          // or just myVal
long newVal = *(ptrLong + 1);       // = 200
```

Dynamic Memory Allocation

- We typically perform ***static*** memory allocation, reserving space for variables when the program begins instead of while it runs.
- However, in some cases it is unknown how much memory we should allocate beforehand, prompting us to ***dynamically*** allocate it during run-time.

new

- We can ***dynamically*** allocate memory using the new operator:

```
char *yesOrNo = NULL;
yesOrNo = new char;
```

- The new operator allocates enough memory to store the data type mentioned, then it returns a ***pointer*** to the newly allocated memory block.

new Arrays

- Why is this useful? When the number of elements to be allocated is unknown prior to run-time:

```
int *bowlSize;
int numBowls;

cout << "How many bowls do you need?" << endl;
cin << numBowls;

bowlSize = new int[numBowls];      // store only enough sizes as necessary

// initialize all sizes to zero
for (int i=0; i<numBowls; i++)
{
   bowlSize[i] = 0;
}
```

delete

- ***Statically*** allocated memory is automatically freed back to the OS when a program terminates.
- ***Dynamically*** allocated memory must also be dynamically freed, or else RAM allocated by your program could be reserved until the user reboots! (This is called a *memory leak*.)

delete

- To free memory when you are finished using it, use the delete operator:

```
delete yesOrNo;
delete [] bowlSize;
```

- Note that you should use [] *before* the variable name to deallocate arrays (the number of elements does not have to be specified).

Stanford's Pointers Video

- http://www.cs.stanford.edu/cslibrary/PointerFunCppBig.avi

Example

- Let's say that we want to allocate memory for an uncompressed image where each pixel is 32 bits. This could perhaps be used as a background or as a *double buffer*:

```
int imgWidth = 640;
int imgHeight = 480;
int pixelBytes = 4;
unsigned long *imgBuffer = NULL;

imgBuffer = new long[imgWidth*imgHeight*pixelBytes];
```

File I/O

- To read from and write to files in C++, we'll use a header file called fstream (file stream):

```
#include <fstream>
using namespace std;
```

- There are two types of ***streams*** that we can create to a file – *output streams* (ofstream) to a file and *input streams* (ifstream) from a file.

Writing to a File

```
ofstream outFile("sample_file.txt");
outFile << "My first file!";
outFile.close();
```

- This opens the file "sample_file.txt" as ***write-only***. Using '<<', you can write to the outFile stream that was created any string or variable:

```
int eggsInDozen = 12;
outFile << "My first file, and there are " << eggsInDozen << " eggs in
   a dozen!" << endl;
```

Reading From a File

```
ifstream inFile(“sample_file.txt”);
int hitPoints;
inFile >> hitPoints;
inFile.close();
```

- This opens the file “sample_file.txt” as ***read-only***. Using ‘>>‘, you can read from the inFile stream. In this case, we are reading in a single variable that is stored in ASCII format.

For More on Files …

- There is a LOT more about how to handle files than I can cover here. In fact, we encourage you to go abroad onto the Internet on these topics discussed to find more information yourselves. Here is a great starting place for learning about C++ File I/O to get you started:

- http://www.cpp-home.com/loobian/tutorials/file_io/index.php

Example

- Writing player stats:

```
int playerHealth, playerStamina, playerMagic;

ofstream outFile(“sample_file.txt”);
outFile << playerHealth << “ “ << playerStamina << “ “ << playerMagic;
outFile.close();
```

- Reading player stats:

```
int playerHealth, playerStamina, playerMagic;
ifstream inFile(“sample_file.txt”);

inFile >> playerHealth >> playerStamina >> playerMagic;
inFile.close();
```

TODO

- **Do your best to finish and submit ITCC_HW3.zip ASAP! Contact us if you have questions!**
- Download ITCC_HW5.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 28.
- Look over the slides for the eighth lecture.
- If you finish with the homework, experiment!