Welcome

Object Oriented Programming with C++
CIS 265

Week 2 – Review II

Christopher K. Burns

Syllabus – schedule changes

Week		Content
1	1/11	*Review* Chapter 1 Intro to Computers and C++ Programming Chapter 2 Control Structures Chapter 3 Functions Chapter 4 Arrays
2	1/18	Additional Array and Function Topics Chapter 5 Pointers and Strings Lab 1 – Functions, Arrays, and Strings Homework 1 Assigned
3	1/25	Chapter 6 Classes and Data Abstraction
4	2/1	Chapter 7 Classes: Part I Lab 2 – Classes 1 Homework 1 Due
5	2/8	***MID-TERM EXAMINATION*** *(Chapters 1 through 7*)* **only portions of chapter 7 that were covered in class* *Final Project Assigned*
6	2/15	Chapter 7 Classes: Part II Chapter 8 Operator Overloading Homework 2 Assigned
7	2/22	Chapter 8 Operator Overloading Homework 2 Due Lab 3 – Classes 2
8	3/1	Chapter 9 Inheritance: Part I Homework 3 Assigned
9	3/8	Chapter 9 Inheritance: Part II Chapter 10 Polymorphism Homework 3 Due
10	3/15	Chapter 10 Polymorphism Lab 4 – Inheritance and Polymorphism
11	3/22	***FINAL PROJECTS DUE***

Agenda

Tonight's agenda

- Review part II
  - Arrays and Functions
  - Pointers and Strings
- Start Chapter 6 – C++ Classes
- Lab 1

Object Oriented Programming

Home Work

Homework #1 – Due two weeks from tonight

Lab #1 – Due next week, if not done in class

Read Chapter 6

Agenda

- Review Part II
  - Arrays
  - Strings as arrays of char
  - Pointers
  - Strings as pointers
- Classes Part 1
  - Struct
  - Class
- Lab 1

Review - Arrays

- An array is a simple collection of data
- You can access members of an array through the array's index

Array of 4 numbers:

index	element
0	12
1	24
2	15
3	5

Declaring Arrays

- Declare Array Form 1
  - dataType nameOfArray[numberOfElements];
  - int numbers[4];

Using Arrays

- Using the Array

```
int numbers[4];
numbers[0] = 12;
numbers[1] = 24;
numbers[2] = 15;
numbers[3] = 5;
```

index	element
0	12
1	24
2	15
3	5

Declaring Arrays

- Declare Array Form 2

  You can declare an array and set its elements:

  – dataType nameOfArray[] =
    { element1, element2, … element*n* };

  – int numbers[] = { 12, 24, 15, 5 };

Using Arrays

- Arrays are most useful in loops that are iterating through its elements.

```
for( int i = 0; i < 4; i ++ )
 {
   cout << numbers[ i ] << endl;
 }
```

Using Arrays

- Usually a good idea to use a constant (or #define) to hold the length of the array.
- If you change the size of the array it makes life easier since you don't have to update controls structures and code that is using array size

```
const int ARRAY_SIZE = 4;
```

or

```
#define ARRAY_SIZE 4
```

Example Using Arrays

```
#define ARRAY_SIZE 4

void doubleEm( int nums[], int count )
{
  for( int i = 0; i < count; i++ )
    nums[ i ]*=2;
}

main()
{
  int numbers[ARRAY_SIZE];
  for( int i = 0; i < ARRAY_SIZE; i++ )
  {
    cout << "Enter number to double " << i << ": ";
    cin >> numbers[ i ];
  }
  doubleEm( numbers, ARRAY_SIZE );
  for( int i = 0; i < ARRAY_SIZE; i++ )
  {
    cout << "Element[" << i << "] = " << numbers[ i ] << endl;
  }
}
```

Sorting and Searching Arrays

Often it is useful to organize the collection of elements in the array. There are also times when you would like to search an array to find an element it contains.

There are numerous sorting and searching algorithms for doing this.

As this is a review we will not cover this, but you should be familiar with sections 4.6-4.8 in the book

Strings – Part I

C-Style strings are just an array of char

Chars in C are enclosed in single quotes

```
char aChar = 'S';
```

So a string could be defined:

```
char aString[] = { 'S', 't', 'r', 'i', 'n', 'g' };
```

Strings are just char arrays

Instead of having to define each char, C lets you define a collection of char using double quotes.

```
char aString[] = { 'S', 't', 'r', 'i', 'n', 'g' };
```

Strings is C are enclosed in double quotes

```
char aString[] = "String";
```

Manipulating Strings as arrays

cout and cin handling strings:

```
char aString = “This is a string”;
cout << aString;
cin >> aString;
```

Manipulating Strings as arrays

You can manipulate the string by treating it as an array.

Problem: How do you know when you've reached the end of the string?

Manipulating Strings as arrays

Problem: How do you know when you've reached the end of the string?

NULL Termination

What is NULL Termination?

Traditionally on the PC, all printable characters are stored internally using their ASCII value (see Appendix B (page 1216) in book.) The NULL character is ASCII code 0.

By placing a NULL character (char nullChar = '\0') as the last element of your character array (at the end of the string) you will know when you've reached the end of the string.

NULL Termination Example

```
const int MAX_STRING_LEN = 256;

main()
{
  char aString[ MAX_STRING_LEN ];
  cout << "Enter some text: ";
  cin  >> aString;
  int i = 0;
  while( aString[ i ] != '\0' )
  {
    i++;
  }
  cout << "String is " << i << " characters long" << endl;
}
```

Another String Example

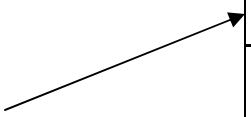
```
void reverseString( char string[] )
{
  int length = 0;
  while( string[ length ] != '\0' )
    length++;
  for( int i = 0; i < ( length ) / 2; i++ )
  {
    char buffer = string[ i ];
    string[ i ] = string[ length - 1 - i ];
    string[ length - i - 1 ] = buffer;
  }
}

main()
{
  char aString[ MAX_STRING_LEN ];
  cout << "Enter some text: ";
  cin  >> aString;
  reverseString( aString );
  cout << "String reversed is " << aString << endl;
}
```

Pointers

The contents of a variable is stored in the computer's memory. A pointer is this memory location. A pointer "points" to a value in memory.

```
int i = 20;
```

i is stored in memory starting at address 0012FED4

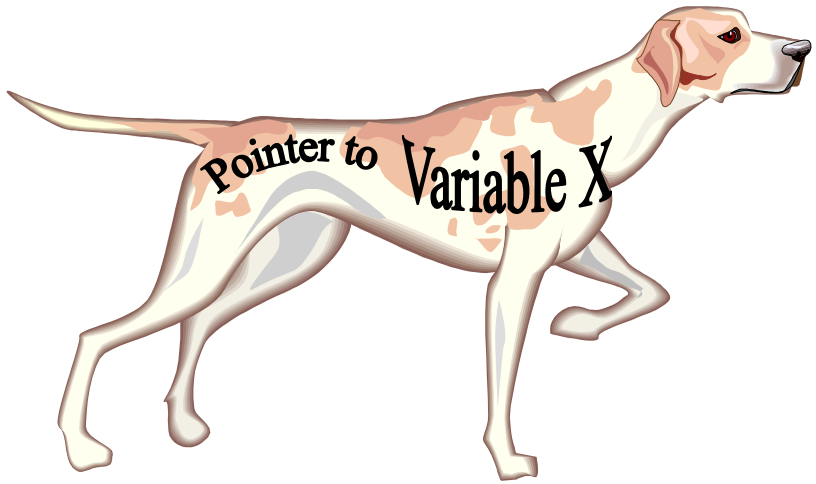


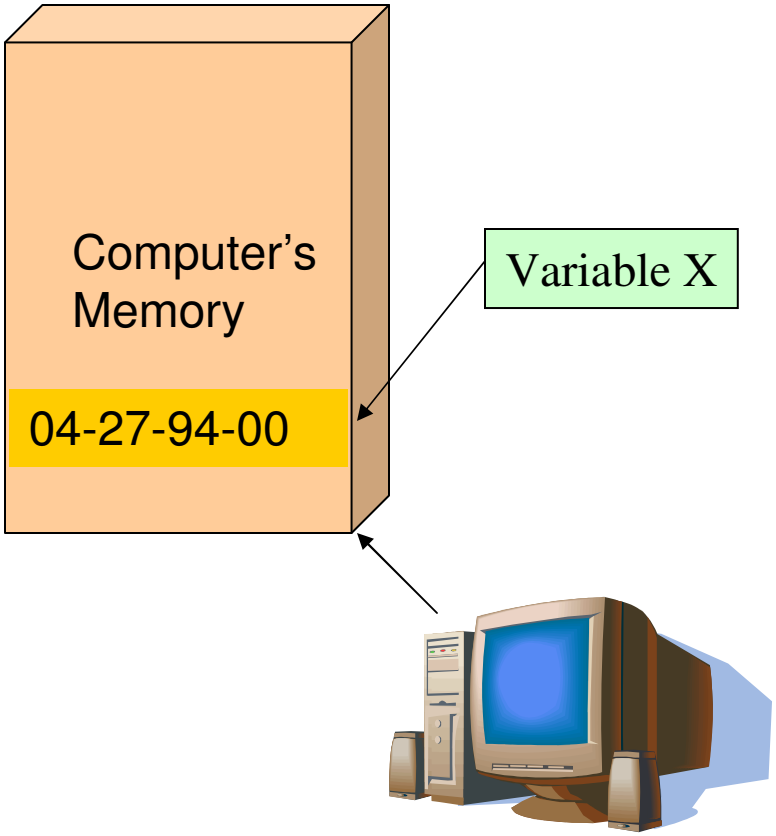
Address	Value
0012FED3	CC
0012FED4	20
0012FED5	0
0012FED6	0
0012FED7	0
0012FED8	CC

Pointers

The contents of a variable are stored in the computer's memory at an address. A pointer contains a memory address.

If the address stored in the pointer is the address where a variable's value is contained, the pointer "points" to a value in memory.





Pointers

The location of the contents in a variable in C can be obtained using the & operator.

```
int i = 20;
cout << "i is stored at: " << &i << endl;
```

When used in this way it is the address operator (not bitwise AND)

Pointers

We can find the address of any variable using the & operator.

The address returned by & in Visual C++ on Pentium class systems is a 4 byte value (long)

The item stored at this address can be of any data type.

How can we represent that this address belongs to a certain data type?

Pointers

* is the value at this memory location

& is the address of this variable

The * is the converse of the & operator

We can declare a pointer variable type. This is a variable that hold a memory address, but has a type associated with it.

int* is a pointer to an int

double* is a pointer to a double

char* is a pointer to a char

These are all essentially the same since they only contain the memory address that corresponds to the start of the variable, but because they have a type (int, double, char…) the compiler knows how to interpret the data they are pointing to.

Pointers

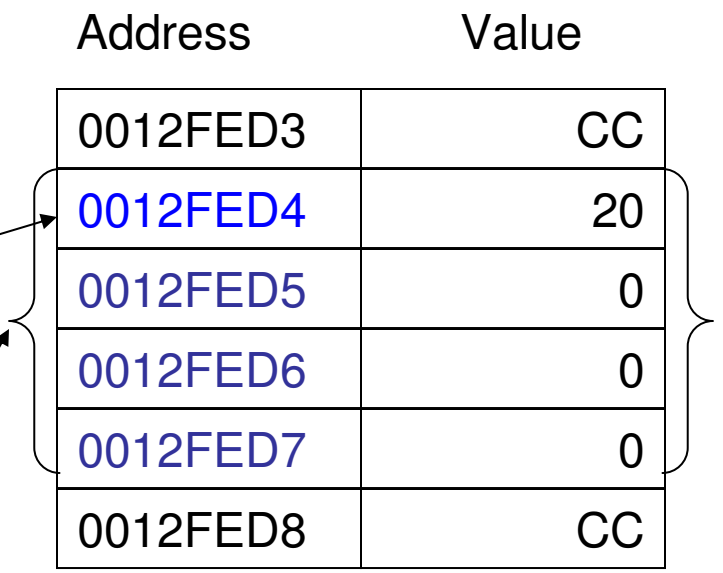
A variable is stored in the computer's memory. A pointer is this memory location. A pointer "points" to a value in memory.

```
int i = 20;
int *ptrI = &i;
```

ptrI == 0x0012FED

*ptrI points to

ptrI is type int so compiler knows how it is stored in memory



Pointers

* is the value at this memory location
& is the address of this variable

The * is the converse of the & operator

```
int  i = 20;
int* iPtr = &i;
cout << "i = " << *iPtr << endl;
```

Pointers and Arrays

You've already used pointers, but they were called arrays!

int array[20];

The variable array is a pointer to the first element of the array.

array == &array[0]

Pointers and Arrays

Thus

```
cout << "array[0] = " << array[0] << endl;
cout << "*array = " << *array << endl;
```

Print out the same result!

Pointers and Arrays and Math

As an array, an index can be used to access (point to) different locations in memory

(remember that the array index selects which value in memory is referenced)

Pointers have the same ability, but it is done with “pointer math”.

Pointers and Arrays and Math

Pointer math is manipulating the location in memory being pointed to by addition/subtraction.

```
int i[] = { 20, 30 };

*i == 20
*(i + 1) == 30
```

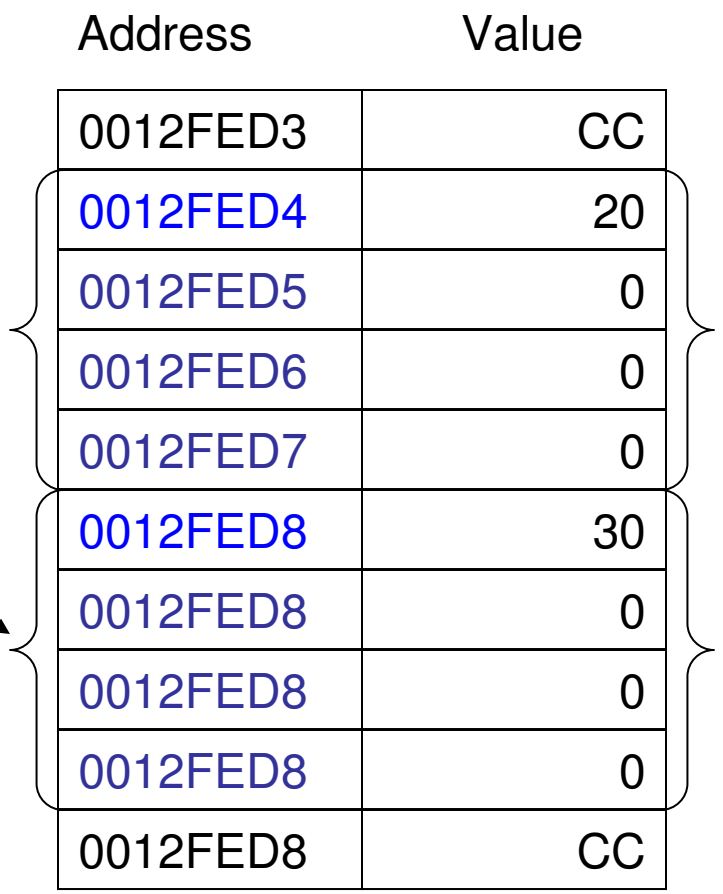
Pointers and Arrays and Math

```
int i[] = { 20, 30 };
```

*i == 20

*(i + 1) == 30

Because the pointer is of type *int*, the pointer math (+1) knows to go to the next int as opposed to the next memory location



Passing Pointers – The Swap

```
void swap( int* a, int* b )
{
  int temp = *a;
  *a = *b;
  *b = temp;
}

main()
{
  int i = 10;
  int j = 20;
  cout << "i = " << i << endl;
  cout << "j = " << j << endl;
  cout << "swapping i and j" << endl;
  swap( &i, &j );
  cout << "i = " << i << endl;
  cout << "j = " << j << endl;
}
```

Pointers and References

Pointers allow for pointer math and the actual address contained in the pointer can be changed.

C++ introduced a different type of pointer called a reference.

A reference is a pointer in spirit but it acts just like the variable it is pointing to.

A reference can be thought of as an alias for a variable.

Pointers and References

```
int i = 10;

int &iRef = i;                  // iRef is a reference to i

int *iPtr = &i;                 // iPtr is a pointer to I

cout << “i = “ << i << “ iRef = “ << iRef << “ *iPtr = “ << *iPtr << endl

i = 10   iRef = 10   *iPtr = 10
```

Passing Pointers – The Swap 2

```
void swapRef( int &a, int &b )
{
  int temp = a;
  a = b;
  b = temp;
}

main()
{
  int i = 10;
  int j = 20;
  cout << "i = " << i << endl;
  cout << "j = " << j << endl;
  cout << "swapping i and j" << endl;
  swapRef( i, j );
  cout << "i = " << i << endl;
  cout << "j = " << j << endl;
}
```

Pointers

Now that you have a thorough understanding of pointers, we can move on… ;)

Pointers are one of the most difficult concepts for programmers new to the C language to master.

(This is one of the reasons James Gosling did not include them in Java)

When using pointers, picture the computer's memory, and that values are just locations in memory and a pointer is just the address of that location.

Strings – Part II

Strings are arrays of char. Arrays are simply pointers. Thus strings can be manipulated as pointers using pointer math. Strings can also be easily passed to functions as either char* or char[].

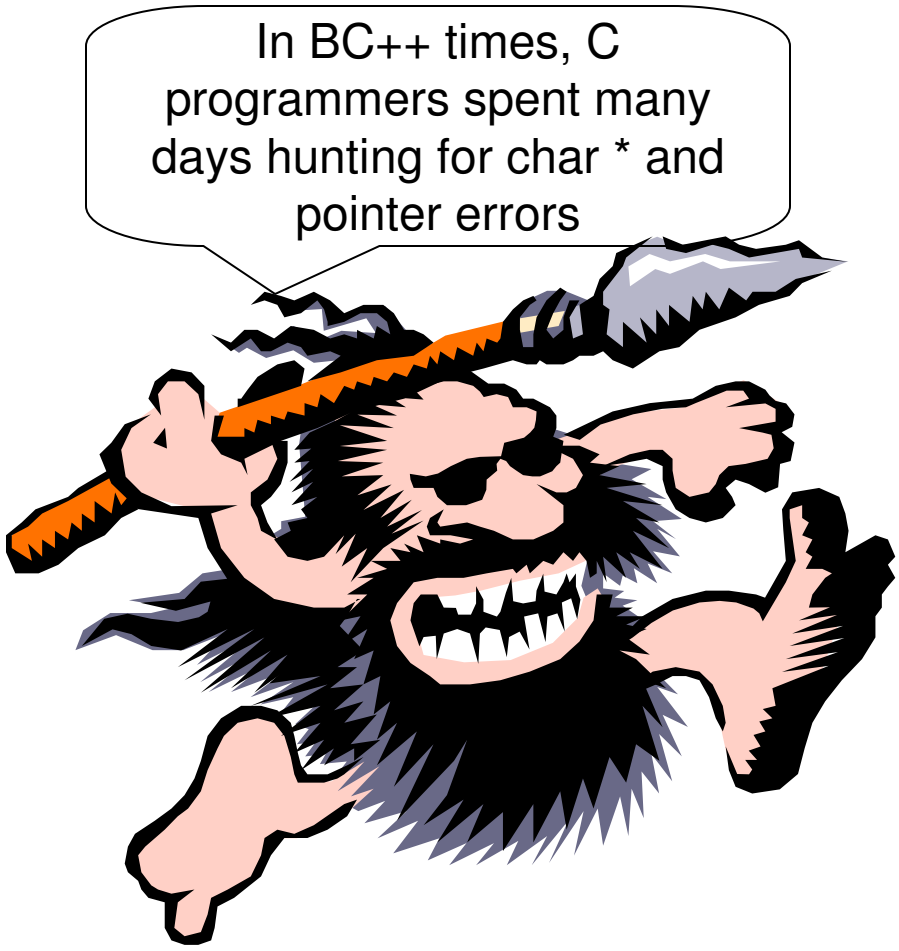
As we'll see this has numerous implications when dealing with string manipulation and passing strings to functions.

Strings – char*

Back in the old C days,

char* was synonymous with string.

Since much of C++ still uses C libraries, we'll stick with the char* convention of string manipulation.



Strings – Standard C Library

There are several functions built into the standard C library for handling null terminated strings. Depending on your development environment there may be many additional functions including some to handle unicode and wide character string formats.

We will look at three categories of string handling functions (there are more string functions available than this list includes):

- String metrics
- Copying strings
- Comparing strings
- Converting strings

Strings – Standard C Library

String metrics

- int strlen(char*) – get the length of a string

Strings – Standard C Library

Copying strings

- strcpy(char* dest, char* source) – copy a string
- strncpy(char* dest, char* source, n) – copy n characters of a string
- strcat(char* dest, char* source) – append source to dest (con*cat*enate)
- strncat(char* dest, char* source) – append n characters of source to dest

Strings – Standard C Library

Comparing strings

- int strcmp(char* string1, char* string2) – returns 0 if strings are lexicographically equal
- int stricmp(char* string1, char* string2) – returns 0 if strings are lexicographically equal. Ignores case.
- char* strstr(char* string, char* subString) – finds first occurrence of subString in string. If found returns pointer to subString
- char* strchr(char* string, char c) – finds first occurrence of c in string. If found returns pointer to c, else returns NULL (0)
- size_t strcspn(char* string, char* charSet) – finds first occurrence of any char in charSet and returns index to that char

Strings – Standard C Library

Convert strings

- int atoi(char*) – converts string into integer. (see also atof, atol)
- itoa(int i, char* output, int radix) – converts integer to string using radix as number base (e.g. radix = 10 for decimal)

Strings – Standard C Library

A complete list of string handling functions for Visual C++ can be found on the msdn website at:

http://msdn.microsoft.com/library/default.asp?url=/library/en-us/vclib/html/_crt_string_manipulation.asp

These functions also require the header file:

```
#include <cstring>
```

to be included

Note: cstring is included by iostream

Strings – Another Example

Example:

sillyString

A function that will return a randomly generated string of a specified length. The string will start with a consonant followed by a vowel followed by a consonant…

sillyString

```
char consonants[] = "bcdfghjklmnpqrstvwxyz";
char vowels[] = "aeiou";
void sillyString( char* string, int length )
{
 srand( (unsigned)time( NULL ) ); // seed random number generator
 for( int i = 0; i < length; i++ )
 {
  if (i % 2) // if i is an odd number
  {
   int randomIndex = (int)
    ( ( (float)rand()/(float)RAND_MAX ) * (float)( strlen( vowels ) - 1 ) );
   string[i] = vowels[randomIndex];
  }
  else // if i is an even number
  {
   int randomIndex = (int)
    ( ( (float)rand()/(float)RAND_MAX ) * (float)( strlen( consonants ) - 1 ) );
      string[i] = consonants[randomIndex];
    }
  }
  string[i] = '\0'; // need to NULL terminate our string
}
```

Ad-hoc Polymorphism

- Function Overloading
  - You can have multiple functions with the same function name.
  - The functions are differentiated by the parameters that are passed to them.
  - The following functions all have the same name, so it is easy for the user of your function. Depending on the parameters passed a different actual version of your function is called.
    - printMe(char*)
    - printMe(int)
    - printMe(float)
    - printMe(char*, char*)

Ad-hoc Polymorphism

- Function Overloading
  - In C++ the name of your function is:

    functionName + parameterNames

    (this is what is referred to a C++ name mangling)
  - The C++ name of the function does not include the return value since it is optional for the caller to use the return value.
  - Thus in C++:

int printMe(char*) has the same name as bool printMe(char*)

Ad-hoc Polymorphism

- Function Overloading
  - The C++ name of the function does not include the return value since it is optional for the caller to use the return value.
  - Thus in C++:

    int printMe(char*)

    has the same name as

    bool printMe(char*)
  - If you tried to compile you would get the error:

```
error C2556: int printMe(char*) overloaded
function differs only by return type from bool
printMe(char*)
```

Function Overloading

- Function Overloading
  - You are writing a function to divide two numbers.
    - If the two numbers are floats, just divide as normal.
    - If the two numbers are integer, divide them but return the result rounded using the floor() function.

Example – divideUs

```
#include <iostream>
#include <cmath>
using namespace std;

const int MAX_STRING_LEN = 256;

float divideUs( float num1, float num2 )
{
  cout << "divideUs - float version called" << endl;
  float returnValue = num1 / num2;
  cout << "         - " << num1 << " / " << num2 << " = " << returnValue << endl;
  return returnValue;
}

int divideUs( int num1, int num2 )
{
  cout << "divideUs - integer version called" << endl;
  int returnValue = (int)floor( (float)num1 / (float)num2 );
  cout << "         - " << num1 << " / " << num2 << " = " << returnValue << endl;
  return returnValue;
}
```

Example – divideUs

Calling divideUs:

```
main()
{
  char aString1[ MAX_STRING_LEN ];
  char aString2[ MAX_STRING_LEN ];

  cout << "Enter two numbers seperated by a space: ";
  cin  >> aString1 >> aString2;

  float floatNum1 = (float)atof( aString1 );
  float floatNum2 = (float)atof( aString2 );
  float floatResult = divideUs( floatNum1, floatNum2 );

  int intNum1 = (int)floatNum1;
  int intNum2 = (int)floatNum2;
  int intResult = divideUs( intNum1, intNum2 );
}
```

Review

The Review is Over

Any Questions?

(If you think of a question later either catch me after class or send me an email)