Math 155A – C/C++ Basic Concepts needed for the course.

Outline:

1. Data types, variable declarations, assignments.
2. If … Else … and other conditionals.
3. For loops, while loops
4. Functions
5. Arrays

DATA TYPES, VARIABLE DECLARATIONS, ASSIGNMENTS

C++ has three basic types we use in the course:

1. Integers: `int`
2. Single precision floating point: `float`
3. Booleans (true/false): `bool`

Less used: `double`, `char`, `char*`

Declarations may be done anywhere in the code, and are in effect for the remainder of the routine in which they are defined.

Sample code:

```
main() {

       int i;        // Declares i to be integer. Value not initialized
       i = 7;        // Sets i equal to 7.
       int j = 2;    // Declares j and sets its value.

       int k = i*i - j;     // k now equal to 47
       k++;                 // k now equal to 48
       k--;                 // k now equal to 47
       k += i;              // k now equal to 54

       float x = 3.14159f;
       float y = sqrtf(x);  // Use #include “math.h” to have this function available

       // It is tricky (usually not a good idea) to mix int's and floats,
       //   but here is an example.
       j = (int)x;           // j is now equal to 3 (rounded down)

       // Watch out for rounding down with integer division (A common programming bug!)
       float z1 = 2.0*(1/3);      // This is probably a bug: 1/3 evaluates to zero
       float z2 = 2.0*(1.0/3.0);  // This is probably what you meant. Sets z2 = 0.666667

       // bools use 0 for false, anything non-zero for true
       bool u = (x > 4);           // Sets u equal to false (with x the value above)
       bool v = (x<4 && j==2);     // Set v equal to true -- note the "==" for comparison
```

```
      u = !v;                     // u is set to the negation of v
      u = (j != 2);               // Use != for "not equal to"
}
```

CONDITIONALS – IF.. ELSE.. etc

`if` tests allow code to be executed only when an expression (test) is true. The optional `else` clause is executed if the expression is false.

More advanced conditionals are the ternary conditional operator `?:` and the `switch` `case` operators.

Sample code:

```
if (i == 0) {
       j = 5;
}
if (i == 1) {
       j = 6;
}
else {
       j = -20;
}

if ((j % 2) == 1) {  // "%" is the "mod" operator
       j--;
}
j = (j >> 1) << 1;   // Does the same as the if. (Advanced topic!)

// Advanced features - do the above if-else with a single line.
j = (i == 1) ? 6 : -20;

switch (i)
{
case 0:
       j = 17;       // Do this when i=0
       break;        // Skip to end of switch (dangerous to omit this)
case 3:
       j = 12;
       break;
default:
       j = 16;       // Do this in all other cases.
}
```

FOR LOOPS, WHILE LOOPS

`for` loops allow code to be iterated while incrementing values (or more sophisticated uses). `while` loops give similar functionality, but are simpler.

Sample code:

```
// Sum the squares of the first 10 integers
int i;
int j = 0;
for (i = 1; i<=10; i++) {   // Test i<=10 is done before anything else
       j += i*i;
}
```

```
       // Same functionality
       j = 0;
       for (int i=0; i <= 10; i++) {
              j += i*i;
       }

       // Specify 10 vertices around a circle (in the plane) of radius 2
       for (int i = 0; i<10; i++) {
              float theta = 3.1415926f*(36.0f*(float)i)/180.0f;
              glVertex2f(2.0f*cosf(theta),2.0f*sinf(theta));
       }

       // Same functionality as first 2 loops
       j = 0;
       i = 0;
       while (i <= 10) {
              j += i*i;
              i++;
       }
```

FUNCTIONS

Functions take values as arguments and possibly return a value. A declaration like

```
float myFunc( float a, float b )
```

means that myFunc takes two floating point arguments and returns a floating point value.

There are many convenient C/C++ functions, like sqrtf(), sinf(), cosf(), atan2f(-,-), etc. To use these, you should include the math.h header file, using `#include "math.h"` at the beginning of your program.

You can also define your own functions. It is necessary to declare them (but not define them) before first use.

Sample Code:

```
int multsOfPi(float x);     // Declaration of the function (input and output types)
float myPi = 3.1415926f;    // A global variable
// ...
int main() {
       float x = 100.0f;
       int y = multsOfPi(x);
       return 1;
}

// Definition of the function.
//       Here is a situation where it is necessary to mix int’s and float’s.
int multsOfPi(float x) {
       int m = (int)(x / myPi);
       return m;
}
```

ARRAYS

Arrays can be one-dimensional or higher dimensional. We shall use them mostly for vectors.

Sample code:

```
float vecTwo[2];     // Allocates an array that holds two values
float vecFour[4] = { 1.0, 2.0, 3.0, 5.0 }; // Allocates and sets four values

vecTwo[0] = 1.0;     // Sets the first entry of vecTwo
vecTwo[1] = -1.0;    // Sets the second entry of vecTwo

glVertex2fv(&vecTwo[0]);    // Pass the vector as an argument to glVertex2f.
glVectex2fv(vecTwo);        // Same functionality

// Allocates and sets four values. Identical functionality to "vecFour"
float vecFourAgain[] = { 1.0, 2.0, 3.0, 5.0 }; // Allocates and sets four value

// Loads the same data as "vecFour", but is accessed differently.
//    This is a two-dimensional array.
float vecFourYetAgain[][2] = { {1.0, 2.0},
                               {3.0, 5.0} };
float x = vecFourYetAgain[0][1];    // Sets x equal to 2.0
float y = vecFourYetAgain[1][1];    // Sets x equal to 5.0
```