Lecture 1 — Introduction and Background

1.1 Reading Material

The material for today's lecture and Friday's lecture is covered in the following sections of the two books.

Accelerated C++ (Koenig and Moo):

Chapter 0,
Chapter 2 (except strings, which we will review in Lecture 2)

C++ Programming (Malik):

Chapter 2, covering the very basics of C++,
Chapter 3 (pages 96-102 only), covering input and output,
Chapter 4, covering if and switch statements, as well as basic expression logic,
Chapter 5, covering while and for loops,
Chapter 6, covering simple user-defined functions.
Chapter 7 (pages 305-338), covering parameter passiing,
Chapter 9 (pages 423-442), covering arrays

This should all be review from CS 1 or a course at an equivalent level. For those of you who started with Java, most of this will be very familiar. Pay most attention to the differences. Overall, you should use the material in Malik as a reference for more information about the topics covered in class.

1.2 Example 1: Hello World

Here is the standard introductory program...

```
// a small C++ program
#include <iostream>

int main()
{
    std::cout << "Hello, world!" << std::endl;
    return 0;
}
```

Small though it is, it may be used to illustrate a number of important points.

1.3 Basic Syntax

- Comments are indicated using `//` for single line comments and `/*` and `*/` for multi-line comments.
- `#include` asks the compiler for parts of the standard library that we wish to use (e.g. the input/output stream function `std::cout`).
- `int main()` is a necessary component of all C++ programs; it returns a value (integer in this case) **and** it could have parameters (next week).
- `{ }`: the curly braces indicate to C++ to treat *everything* between them as a unit.

1.4 The Standard Library

- The standard library is not a part of the core C++ language. Instead it contains types and functions that are important extensions.
  - In our programming we will use the standard library to such a great extent that it will "feel" like part of the C++ core language.
- streams are the first component of the standard library that we see.
- `std` is a *namespace* that contains the standard library
- `std::cout` and `std::endl` are defined in the standard library (in particular, in the standard library header file `iostream`).

1.5 Expressions

- Each **expression** has a **value** and 0 or more **side effects**.
- An expression followed by a semi-colon is a **statement**. The semi-colon tells the computer to "throw away" the value of the expression.
- The line

  ```
  std::cout << "Hello, world!" << std::endl;
  ```

  is really two expressions and one statement. We will go over this in detail in class.
- `"Hello, world!"` is a *string literal*.

1.6 C++ vs. Java

The following is provided as additional material for students who have learned Java and are now learning C++.

- In Java, everything is an object and everything "inherits" from `java.lang.Object`. In C++, functions can exist outside of classes. In particular, the `main` function is never part of a class.
- Source code file organization in C++ does not need to be related to class organization as it does in Java. On the other hand, creating one C++ class (when we get to classes) per file is the *preferred* organization, with the *main* function in a separate file on its own or with a few helper function.
- Compare the "hello world" example above in C++ to the same example in Java:

```
public class HelloWorld {
  public static void main(String args[]) {
    System.out.println("Hello  World");
  }
}
```

  - The Java object member function declaration

    ```
    public static void main(String  args[])
    ```

    Plays the same role as

    ```
    int main()
    ```

    in the C++ program. A primary difference is that there are no string arguments to the C++ `main` function. Very soon (next week) we will start adding these arguments, although in a somewhat different syntactic form.
  - The statement

    ```
    System.out.println("Hello  World");
    ```

    is analogous to

    ```
    std::cout << "Hello, world!" << std::endl;
    ```

    The `std::endl` is required to end the line of output (and move the output to a new line), whereas the Java `println` does this as part of its job.

1.7 Example 2: Temperature Conversion

Our second introductory example converts a Fahrenheit temperature to a Celsius temperature and decides if the temperature is above the boiling point or below the freezing point:

```
#include <iostream>
using namespace std;   // Eliminates the need for std::

int main()
{
  // Request and input a temperature.
  cout << "Please enter a Fahrenheit temperature:  ";
  float fahrenheit_temp;
  cin >> fahrenheit_temp;

  // Convert it to Celsius and output it.
  float celsius_temp = (fahrenheit_temp - 32) * 5.0 / 9.0;
  cout << "The equivalent Celsuis temperature is " << celsius_temp
       << " degrees.\n";

  // Output a message if the temperature is above boiling or below freezing.
  const int BoilingPointC = 100;
  const int FreezingPointC = 0;
  if ( celsius_temp > BoilingPointC )
    cout << "That is above the boiling point of water.\n";
  else if ( celsius_temp < FreezingPointC )
    cout << "That is below the freezing point of water.\n";

  return 0;
}
```

1.8 Variables and Constants

- A variable is an object with a name (a C++ identifier such as `fahrenheit_temp` or `celsius_temp`).
- An object is computer memory that has a type.
- A type is a structure to memory and a set of operations.
- Too abstract? Think about `float` variables:
  - Each variable has its own 4 bytes of memory, and this memory is formatted according floating point standards for what represents the exponent and mantissa.
  - Many operations are defined on floats, including addition, subtraction, etc.
  - A float is an object
- A constant — such as `BoilingPointC` and `FreezingPointC` — is an object with a name, but a constant object may not be changed once it is defined (and initialized). Any operations on integer types may be applied to a constant int, except operations that change the value.
- Note that the use of capitals and `_` in the identifiers is merely a matter of style.

1.9 Expressions, Assignments and Statements

Consider the *statement*

```
float celsius_temp = (fahrenheit_temp - 32) * 5.0 / 9.0;
```

- The calculation on the right hand side of the `=` is an expression.
  - You should review the definition of C++ arithmetic expressions and operator precedence (Malik, Chapter 2). The rules are pretty much the same in C++ and in Java.
- The value of this expression is assigned — stored in the memory location — of the newly created float variable `celsius_temp`.

1.10 Conditionals and IF statements

Intuitively, the meaning of the code

```
if ( celsius_temp > BoilingPointC )
  cout << "That is above the boiling point of water.\n";
else if ( celsius_temp < FreezingPointC )
  cout << "That is below the freezing point of water.\n";
```

should be pretty clear.

- The general form of an if-else statement is

```
if ( conditional-expression )
   statement;
else
   statement;
```

- Each `statement` may be a single statement, such as the `cout` statement above, a structured statement, or a compound statement delimited by {...}
  - The second `if` is actually a structured statement that is part of the `else` of the first `if`.
- Students should review the rules of logical expressions and conditionals.
- These rules and the meaning of the if - else structure are essentially the same in Java and in C++.

1.11 Example 3: Julian Day

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

const int DaysInMonth[13] = { 0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 };

//  The following function returns true if the given year is a leap year
//  and returns false otherwise.
bool
is_leap_year( int year )
{
  return year % 4 == 0 && ( year % 100 != 0 || year % 400 == 0 );
}

//  Calculate and return the Julian day associated with the given
//  month and day of the year.
int
julian_day( int month, int day, int year )
{
  int jday = 0;
  for ( unsigned int m=1; m<month; ++m )
    {
      jday += DaysInMonth[m];
      if ( m == 2 && is_leap_year(year) ) ++jday;  // February 29th
    }
  jday += day;
  return jday;
}

int
main()
{
  cout << "Please enter three integers (a month, a day and a year): ";
  int month, day, year;
  cin >> month >> day >> year;

  cout << "That is Julian day " << julian_day( month, day, year ) << endl;
  return 0;
}
```

1.12 Arrays and Constant Arrays

- An array is a fixed, consecutive sequence of objects all of the same type.
- The following declares an array of 15 double values:

```
double a[15];
```

- The values are accessed through subscripting operations:

```
int i = 5;
a[i] = 3.14159;
```

  This assigns the value 3.14159 to location `i=5` of the array. Here `i` is the *subscript* or *index*

- C++ array indexing starts at 0.
- Arrays are fixed size, and each array knows NOTHING about its own size. The programmer must write code that keeps track of the size of each array.
  - In the next few weeks we will see a standard library generalization of arrays, called *vectors*, which do not have these restrictions.
- In the statement:

```
const int DaysInMonth[13] = { 0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 };
```

  - `DaysInMonth` is an array of 13 constant integers.
  - The list of values within the braces initializes the 13 values of the array, so that `DaysInMonth[0] == 0`, `DaysInMonth[1]==31`, etc.
  - The array is global, meaning that it is accessible in all functions within the code (after the line in which the array is declared). Global constants such as this array are usually fine, whereas global variables are generally a VERY bad idea.

1.13 Functions and Arguments

- Functions are used to:
  - Break code up into modules for ease of programming and testing, and for ease of reading by other people (never, ever, under-estimate the importance of this!).
  - Create code that is reusable at several places in one program and by several programs.
- Each function has a sequence of parameters and a return type. For example,

```
int julian_day( int month, int day, int year )
```

  has a return type of `int` and three parameters, each of type `int`.

- The order of the parameters in the calling function (the main function in this example) must match the order of the parameters in the function prototype.

1.14 `for` Loops

- The basic form of a `for` loop is

  ```
  for ( expr1; expr2; expr3 )
    statement;
  ```

  where

  - `expr1` is the initial expression executed at the start before the loop iterations begin;
  - `expr2` is the test applied before the beginning of each loop iteration, the loop ends when this expresion evaluates to `false` or `0`;
  - `expr3` is evaluated at the very end of each iteration;
  - `statement` is the "loop body"

- The `for` loop from the `julian_day` function,

  ```
  for ( unsigned int m=1; m<month; ++m )
    {
      jday += DaysInMonth[m];
      if ( m == 2 && is_leap_year(year) ) ++jday;  // February 29th
    }
  ```

  adds the days in the months 1..`month`-1, and adds an extra day for Februarys that are in leap years.

- `for` loops are essentially the same in Java and in C++.

1.15 A More Difficult Logic Example

Consider the code in the function `is_leap_year`:

```
return year % 4 == 0 && ( year % 100 != 0 || year % 400 == 0 );
```

This is important to understand.

- If the year is not divisible by 4, the function immediately returns `false`, without evaluating the right side of the `&&`.
- For a year that is divisible by 4, if the year is not divisible by 100 or is divisible by 400 (and is obviously divisible by 100 as well), the function returns true.
- Otherwise the function returns false.
- The function will not work properly if the parentheses are removed, in part because `&&` has higher precedence than `||`.