Section 2 - C/C++ Basics

This section will cover:

- 2.1 Simple C++ Program
- 2.2 Use of Variables
- 2.3 Expressions
- 2.4 Input/Output
- 2.5 Programming Errors
- 2.6 Programming Style

2.1 A Simple C++ Program

```
// marks2.cpp
//
// Program to determine the final mark based on the average of
//    midterm and final exam grades.

/* Note that comments can also be
   placed between
   brackets like this */

#include <iostream>
using namespace std;
int main( )
{
   double midterm = 0.0;
   double finalexam = 0.0;
   double average = 0.0;

   cout << "Enter midterm and final exam grades: " << endl;
   cin >> midterm >> finalexam;
   cout << "You entered " << midterm << " and " << finalexam << endl;

   average = (midterm + finalexam)/2.0;
   cout << "Average of midterm and final exam is"
        << average << endl;

   return 0;
}
```


1) Use of Brackets

- Two bracket types used to document eg.

```
/* This format used in C & C++; can be spread
  over multiple lines; cannot imbed these
  symbols */
```

or

```
// This format used in C++ only; single line
// comments only; you will generally use these
// in 121; can be imbedded in /* … */ brackets
```


2) `#include <iostream>`

- Known as a “compiler directive” (starts with a ‘#’)
- Treats library ‘iostream’ as though it were part of the program
- iostream takes care of the C++ input & output (I/O) streams eg. “streams” of data from keyboard or sent to monitor, etc.
- `#include <iostream.h>` is an old style; not part of the standard

2) `using namespace std;`

- Terms:
  - ‘`using`’ is known as a ‘directive’
  - ‘`namespace`’ is a reserved ‘keyword’
  - ‘`std`’ is a type of namespace (the standard namespace)
- A namespace stores collection of names
- Why? So many names in C++! Ambiguities removed by selecting appropriate namespace

3) `int main()`

- Example of a function: block of computer code set aside from the main code to perform a particular task
- Every C++ program has a main function
- Every function can “return” information
- More on functions and return types later

4) { … }

- Curly brackets enclose functions and other blocks of code to be set apart in the computer code
- Code within curly brackets is normally indented - not required by the compiler, but used to make the code more readable
- See the SD121 Style Guide

5) `double midterm = 0.0;`

- Variable declaration: datatype is "`double`" and variable name is "`midterm`"
- Variables must be declared prior to using them
- Normally declare on separate lines eg. use

```
double midterm;      // declare w/o initializing
double finalexam = 0.0;  // w/ initializing
```

as opposed to (also acceptable to the compiler):

```
double midterm, finalexam;
```

- Declare any time before variable used eg. halfway thru program is ok; most other programming languages do not allow this

6) ;

- Most C/C++ lines of code end in a ";"
- Compilers interpret a ";" as being at the end of a line of code
- Programs made readable by placing these statements on separate lines (compiler really doesn't care, but we do!)

7) `cout << "    " << endl;`

- Output statement: sends data to the monitor
- "string" (a list of characters) is passed to the object "`cout`" (the standard output screen)
- An arrow "<< " (the insertion operator) indicates direction of the stream of data ie. string being sent to the monitor
- '`endl`' moves cursor to next line

8) `cin >> midterm >> finalexam;`

- Input statement: keyboard entry (the "standard input" represented by "object" `cin`) & passed first to `midterm` and then to `finalexam`
- Any "white space" (tab, return key, space) can be used to separate the data entry
- ">>" is the extraction operator ie. extracts data from the stream
- `cout` & `cin` are coded in the iostream library ie. you must include iostream to use them

9) `average = (midterm + finalexam)/2.0;`

- Example of a calculation
- '=' has a different meaning than that found in mathematics
  - Math: lhs is equal to rhs
  - Computers: rhs is "assigned to" lhs
- '=' known as the "assignment operator"
- Only one variable represented on lhs

2.2 Variables

- Programs store data using variables eg.

  some_value = 5;

- In general:
  - → variables can store different data types (e.g. numbers, text)
  - → variables must be declared (memory must be reserved)
  - → each variable has its own memory address
  - → values of variables can be changed (usually!)
  - → variable names should be meaningful (code is then more readable), e.g. use 'velocity' instead of v

2.2 Variables (cont.)

C++ Rules for Variable Names:

- Letters, digits, and underscore allowed in name (no spaces)
- Must start with a letter or underscore symbol (_)
- C++ is case-sensitive, e.g. `myname MyName MYNAME` are all different variables. The naming convention for variables is lowercase
- Allowable or interpretable lengths vary from compiler to compiler
- Cannot use reserved words, e.g. `main` (see Appendix 1 of Savitch for list)

2.2 Variables (cont.)

- Each variable has unique data type associated with it.

Name	Memory	Range
short int	2 bytes	± 32,767
int	4 bytes	±2,147,483,647
long	4 bytes	± 2,147,483,647
unsigned int	4 bytes	0 – 4,294,967,294
float	4 bytes	10^{-38} to 10^{38} (7 digits precision)
double	8 bytes	10^{-308} to 10^{308} (15 digits precision)
long double	10 bytes	10^{-4932} to 10^{4932} (19 digits precision)

- Generally use (for SD121) `int` and `double`

2.2 Variables (cont.)

- A note about ranges and precision …
  - – Integers are exact!
  - – Floating point numbers are approximations
  - – For example, the two variables DO NOT store the same value (although it might appear so!)

```
int integer_val = 3; // stores 3 exactly
double double_val = 3.0; // stores approximation
```


2.2 Variables (cont.)

- 'int' used when variable only requires integer values eg. counters
- 'double' used when real numbers required
- Use scientific notation (no commas!) eg.

  3.14

  7.71e3 is equivalent to 7710 or 7.71×10^3
  - – 7.71 is the "mantissa" (double or int) and 3 is the exponent (int)

2.2 Variables (cont.)

- Characters: letters, digits, and punctuation

  `char initial = 'b';`
  - – Learn how to represent words later
- Boolean data type: stores binary data

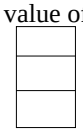
  `bool flag = true;`

  `bool decision = false;`

2.3 Expressions

- Recall that the assignment operator is used to change the value of a variable:

```
int total;
total = 2;
total = 7;
```



- Often assign variables when created eg.

```
int total = 0;
```


2.3 Expressions - Type Compatibility

- Be careful when mixing data types in the same expression eg.

```
int some_value = -54; // ok!
int num = 2.7; // do not use
```

- Integers and doubles can be assigned to a double eg.

```
int num = 2;
double value = num;
value = 3.9;  // all acceptable
```

- Although admissible, not advisable to assign an integer to a char or a char to an integer

2.3 Expressions - Operations

- +, -, *, /, { } can be used in the usual manner

Algebraic	Computer Code
$\frac{A}{B} \bullet C$	
$\frac{A}{B \bullet C} + D$	
$\frac{A - E}{B \bullet C} + D$	

2.3 Expressions - Precedence Rules

- Precedence rules refer to the order of operations; rules are similar to algebra
- Partial list:
  1) ( )
  2) *, /
  3) +, -

  Equivalent operations generally performed left to right.

2.3 Expressions - Determining Expression Type

1) Operations of two ints yields an int

```
int val = 7/4;          // result is _____?
double value = 7/4;     // result is _____?
int num = 17%5;         // result is 2 (the remainder);
                        // '%' is modulo operator
```

2) Operations using a double generates a double

```
aa = 2 * y + x;       // if x or y double, aa
                      // receives a double
double val = 6.0/4;   // val becomes _____?
```

3) Minus sign does not change type

```
int num = 6;
int num = -num;       // num becomes -6
```


2.3 Expressions - Examples

```
int num;

num = 6 * 2 / 4 - 3%2 + 1;     // num is _____?

num = 6 * (2 / 4) - 3%2 + 1;   // num is _____?
```

2.3 Expressions - Shorthand Ops

- Shorthand operators are permissible in C/C++
- These improve readability (once you use them a couple of times) and improve compiler completion times (important with huge programs)
- Use shorthand operators whenever possible

2.3 Expressions - Shorthand (cont.)

longhand	shorthand
`a = a + b`	`a += b`
`a = a - b`	`a -= b`
`a = a * b`	`a *= b`
`a = a / b`	`a /= b`
`a = a * (b + c)`	`a *= (b + c)`
`a = a + 2`	`a += 2`

A common operation is to increment (decrement) by 1. This can be performed using the following format:

```
j = j + l;  or      j++;
j = j – 1;  or      j--;
```

Called 'C++' because it is a version built on 'C'!!!

2.3 Expressions - Constants

- Some variables remain constant in a program, e.g. pi, speed of light, gravity, etc.
- Use the modifier `const` to create a variable that cannot change.

  Syntax: `const Type_name Variable_name = value;`

```
const double PI = 3.14159;
const double SPEED_OF_LIGHT  = 2.99792458e8;
```

- Convention to use upper case letters
- You may have used compiler direction `'#define'` (which performs a direct substitution); convention today is to use `const` since it is more versatile ie. place anywhere in code instead of only at beginning

2.4 Input / Output

- Effectively equivalent statements:

```
cout << "Programming is fun" << endl;
cout << "Programming is fun\n";
```

- `\n` (newline character) must be inserted into the quotes; called a control character; preferable to use `endl`
- Variables can be used in output statements eg.

```
int num = 12;
cout << "Number is: " << num << endl;
```


2.4 I/O - Output Formatting

- Formatted output enabled using I/O manipulation library eg. `#include <iomanip>`
- Some of the syntax may seem odd, but you will get used to it
- Use the following steps:

2.4 I/O - Output Formatting (cont.)

1) Set output to fixed point or scientific notation using either:

```
cout.setf(ios::fixed);
cout.setf(ios::scientific);
```

– "::" indicates the "scope resolution operator"; more on this operator later

2.4 I/O - Output Formatting (cont.)

2) Set the number of decimal places (for *fixed*) or significant digits (for *scientific*)

```
x = 1.1987654;                  OUTPUT
cout.precision(2);              // default is scientific
cout << "x=  " << x << endl;    // x = 1.2 (rounded)
cout.precision(5);
cout << "x=  " << x << endl;    // x = 1.1988 (rounded)
cout.setf(ios::fixed);
cout.precision(2);
cout << "x=  " << x << endl;    // x = 1.20 (rounded)
cout.precision(5);
cout << "x=  " << x << endl;    // x = 1.19877 (rounded)
```


2.4 I/O - Output Formatting (cont.)

3) Right justify text to help line up columns

```
cout.setf(ios::fixed);
cout.precision(3);
x = 1.1;
cout << "x= " << setw(8) << x << " percent"
     << endl;
x = 1.234321;
cout << "x= " << setw(6) << x << " percent"
     << endl;
```

Produces the following output ('#' is a blank):

x = ###1.100 percent
x = #1.234 percent

Q: What happens if precision setting > setw setting?

2.4 I/O - Input

- Recall

```
cin >> var1 >> var2;
```

which is equivalent to:

```
cin >> var1;
cin >> var2;
```

- You should echo keyboard input eg.

```
cout << "Enter a value: ";
cin >> value;
cout << "Value entered is: "
     << value << endl;
```


2.5 Programming Errors

- Three general types of programming errors exist:
  1) Syntax error
  2) Run-time error
  3) Logic error

2.5 Programming Errors (cont.)

1) Syntax error: mistakes in 'grammar'
   – compiler will detect
   – usually easy to remove
   – error message may be cryptic
2) Run-time error: error noticed when program run
   – error with input data or performing a calculation
   – message output to user eg. divide by zero

2.5 Programming Errors (cont.)

3) Logic error: errors in algorithm or its implementation
   – can be difficult to isolate
   – time consuming, especially with large programs
   – professional programmers spend most time on logic errors
   – how to avoid? Plan before you code!

2.6 Programming Style

- Style Guide should be consulted for how to present your code. Considerations:
  - indenting
  - blank lines (separate groups of statements)
  - use meaningful names
  - use comments to document (don't document the obvious, be concise and informative)
  - place header at start of every file