# CSCI 1300 – Practicum 1 Review Guide

## 1a. Data Types :=

So far this semester we have been working with several types of data:
- integers (**int**): any whole number, positive or negative
- floating point numbers (**float**, **double**): any decimal number, positive or negative
- characters (**char**): characters, letters, digits, other symbols. 'c'
- strings (**string**): a continuous series of characters, can include symbols, letters, and digits. "class"
- booleans (**bool**): discrete logical values: true or false

## 1b. Casting :=

Data types are not concrete values themselves, but rather, ways of interpreting values.
This means that the same value can be 'cast' to different data types to create different values.

For example, the number 10 can be cast to a string to produce the string "10".

C++ contains a built in functions to cast a value to each data type
- int()
- float()
- str()
- bool()

## 2. Variables :=

Variables are ways of referring to values. They are not the values themselves. Each variable has:
- a name
  - there are rules associated with variable names.
  - Must start with …
  - Cannot be a reserved word
- a value
- a memory address

## 3. Arithmetic :=

Know how to perform:
- addition
- subtraction
- multiplication
- division
  - integer
  - floating point
  - modulo/remainder
- exponentiation
- taking roots

**4. Print/ Output_to_the_command_window / Display :=**

cout statements:

- text messages (as strings in "…")
- values of variables
- values from functions calls
- special characters "\n"
- endl

**5a. Defining Functions :=**

You need to know how to define functions. All functions have 3 major components:

- the function must have a unique FUNCTION NAME; review the rules for naming a function in C++
- the function takes **some** INPUT PARAMETERS. Must specify the type for each parameter.
- the function **may** RETURN some value; the function may not RETURN anything

INPUT PARAMETERS -> [FUNCTION NAME] -> RETURN VALUE

In C++, functions have the following format:

```
<return_type> <FUNCTION_NAME>(<0 or more INPUT PARAMETERS, with each type>)

{

          <FUNCTION BODY>
          <optional RETURN STATEMENT>
}
```

Example:

```
Int CountABC (string inputString)
{
      int l = inputString.length();
      return l;
}
```

**5b. Calling Functions :=**

Once you define a function you can begin to call it. To call a function, simply type the function name EXACTLY AS IT WAS DEFINED and pass the necessary arguments.

NOTE: FUNCTION NAMES ARE CASE-SENSITIVE

- Calling functions that return a value
- Calling functions that do not return anything

**6. Conditionals :=**

A conditional can be described as any statement that evaluates to a boolean True or False.

A conditional might be an arithmetic expression:

```
      x > 10
      x != 8
      x <= 1000000
```

A conditional might also be the result of function that itself returns a True or False:

```
      bool divisible_by_4(int x)
      {
            return x % 4 == 0;
      }
```

divisible_by_4(10) returns false
divisible_by_4(16) returns true

Conditionals can also be chained together using logical NOT, AND and OR statements.

x > 0 && divisible_by_4(x) => x IS positive AND x is divisible by 4
x <= 0 || !divisible_by_4(x) => x is NOT positive OR x is NOT divisible by 4

**AND** means that both sub-conditionals must be True for the entire conditional to be True. Otherwise the entire conditional with be False.

Truth Table:

```
      X && Y
      T && T=T
      T && F=F
      F && T=F
      F && F=F
```

**OR** means that if either sub-conditional is True then the entire conditional will be True. If both sub-conditionals are False, then the entire conditional will be False.

Truth Table:

```
      X || Y
      T || T=T
      T || F=T
      F || T=T
      F || F=F
```

## **7. If, else, IF-ELSE Blocks :=**

Conditionals allow our programs to have different behavior depending on certain conditions that we specify. Because of this they are also referred to as Control-Flow statements because they CONTROL the FLOW of our programs.

To specify conditions we use if,else if,else blocks:

```
      if (<some condition>)
      {
            <do this>
      }
      else if (<some other condition>)
      {
            <do that>
      }
      else
      {
            <if none of the above cases were true, do something else>
      }
```

Example:

```
      if (x > 0)
            cout << "x is positive" ;
      else if (x < 0)
            cout << "x is negative" ;
      else
            cout << "x is neither positive nor negative; x equals 0";
```

There are several important things to understand about if-else blocks.

- Each block represents an isolated test. This means that within the context of an if-elseif-else block, once a condition is satisfied, the entire block is done.

For example: in the above example, as soon as x is determined to be positive, negative, or zero, the entire block is done.

No additional cases will be tested. However, if we instead wrote:

```
      if x > 0
            cout << "x is positive";
      if x < 0
            cout << "x is negative";
      if x == 0
            cout << "x is neither positive nor negative; x equals 0";
```

then regardless of the value of x, <u>all the conditions will be checked</u> each time the code runs.

- An if-elseif-else block must begin with an if statement.

- That initial if statement can be followed by **0 OR MORE else if** statements that check additional conditions.

- An if statement can be ended with 0 OR 1 else statement.

- The final else statement should serve as a catch-all or default case. i.e. if none of the above cases are true, then "do this".

## 8. Loops:

Like if-elseif-else blocks, some loops also use conditionals to control the flow of execution in a program. Instead of executing some series of statements ONCE based on a conditional being true, a **while** loop will CONTINUE to execute some series of statements AS LONG AS a conditional is true.

The format of a WHILE LOOP is:

```
 while (<conditional>)
 {
     <1 OR MORE statements>
 }
 <first statement outside the loop>
```

Example:

```
  int x=5
  while (x >= 0)
  {
      cout << x << endl;
      x=x-1;
  }
    => 5
    => 4
    => 3
    => 2
    => 1
    => 0
```

## 9. Strings:

Strings are collections or characters. You need to know how to:

- get the number of characters in the string
- use the length() function
- reference any character in the string using an integer index (between 0 and …)
- traverse a string using a loop