Lab 0 Contents – Please note that you don't need to upload anything to Moodle in this lab

Introduction

In this course you will learn to write C++ programs to solve different problems. From time-to-time this task may become frustrating and difficult. However, with patience and practice, you will gain more experience and things will get easier. The difficulty in using a computer to solve problems comes from the fact that you need to somehow ask a computer to do things for you, we communicate with it through an operating system (OS). The communication with OS is done via the program that you write. Our program is a set of instructions that a computer will follow. These instructions are not in English but we understand them very well, because they are written in a high-level language. These instructions will, at some point, get to the machine much differently in machine language. The machine language is a low-level language.

In general, you will go through several steps before you can get an output from your program; 1) First you will use an editor to type your program, 2) then you will use a compiler (in our case a C++ compiler) to compile your program and to get an executable file, and 3) at last, you will run the executable file to obtain the output.

The most important part of a programmer's job is solving the problem first. It is much harder to solve a problem than to translate your ideas to a specific programming language. Thus, one should first think about a method and develop an algorithm to solve the problem. An algorithm is a sequence of precise instructions, which results in a solution. The keyword here is precision. If your algorithm has ambiguity in it, then you will not get the correct answer.

What is Computational Problem-Solving?

A computer solves a problem by following a series of steps that are part of a process known as **computational problem-solving**. Here's a breakdown of the process:

1. Problem Definition and Understanding

- **Input**: The problem is defined in terms of inputs, which are data the computer receives.
- **Output**: The desired outcome or solution is specified, explaining what result is expected.
- The problem should be clearly understood and broken down into smaller, manageable tasks.

2. Algorithm Design

- An **algorithm** is a step-by-step set of instructions that solves the problem.
- Algorithms are designed to be efficient, ensuring minimal time and resources are used to achieve the solution.
- The process may involve using well-known techniques such as loops, conditionals, recursion, and sorting.

3. Programming and Implementation

- The algorithm is implemented using a programming language (e.g., Python, C++, Java).
- The code converts the algorithm into instructions that the computer can execute.
- The computer translates the code into machine language through a **compiler** or **interpreter**.

4. Execution

- The computer takes the **input**, processes it using the algorithm, and produces the **output**.
- During execution, the **Central Processing Unit (CPU)** carries out basic operations like arithmetic, data movement, and logical comparisons.

5. Error Handling and Debugging

- As the program runs, there might be errors or unexpected results.

- Debugging tools and techniques are used to fix issues, ensuring the program runs correctly.

6. Optimization

- Once the problem is solved, the program or algorithm can be optimized for better performance.
- This might include reducing memory usage, speeding up execution time, or simplifying the algorithm.

7. Output and Verification

- The final step is checking whether the output is correct by comparing it to the expected result.
- This ensures that the computer has solved the problem correctly and as intended.

The process is iterative, meaning improvements or changes are made until the computer solves the problem efficiently and accurately.

A Simple C++ Program

1) Problem Definition:

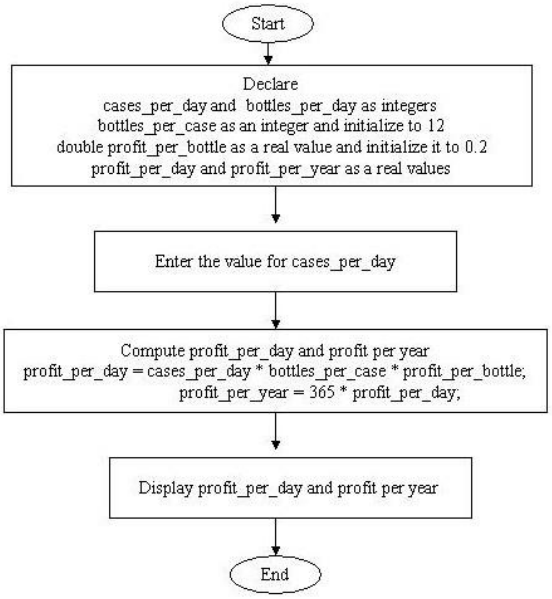
A grocery store sells many cases of soft drink every day. In each case, there are 12 bottles and the store profits 20 cents per bottle. We want to compute the profit that the store makes every day of selling soft drink. We also want to know the profit from selling soft drink in a year. Assume a year is 365 days.

Compute the profit that a store has in one day for selling soft drink?
Compute the profit that a store has in one year for selling soft drink?

This program is so computationally extensive, that we want to write a C++ program to solve it on a computer! At least we will assume it is for now.

2) Program Design - Algorithm

Before we attempt to write the program, let's develop an algorithm for solving the problem. On a piece of paper draw a diagram or write in English the steps. Remember your algorithm must be precise. Below is the diagram showing the steps.



This algorithm must be translated to C++ to obtain the program.

3) Implementing the Algorithm in C++

This is where you will translate the algorithm to C++. Here is a program that is designed based on the algorithm that is given in the previous part. Check to make sure you find everything is translated preciously.

```
//  This C++ Program will compute the profit of selling soft drinks

#include <iostream>
using namespace std;

int main( )
{
    int cases_per_day, bottles_per_day;
    int bottles_per_case = 12;
    double profit_per_bottle = 0.2;  // 20 cents per bottle profit
    double profit_per_day, profit_per_year;

   cout << “Press enter after entering each number \n”;
   cout << “Enter number of cases \n”;
   cin >> cases_per_day;

   profit_per_day = cases_per_day * bottles_per_case * profit_per_bottle;
   profit_per_year = 365 * profit_per_day;
```

```
    cout << "The store has a made : ";
    cout << profit_per_day;
    cout << "  per day. \n";
    cout << "That means the profit for one year will be:  ";
    cout << profit_per_year << endl;

    cout << "Good business?! \n";
    return 0;
}
```

4) What is in the Simple C++ Program?

On the first line of the program, we have:

```
// This C++ Program will compute the profit of selling soft drinks
```

The // tells the compiler that the line is only a comment and do not participate in the computing. Comments are added for readability and/or to describe parts of a program.

On the next line you have:

```
#include <iostream>
```

This is called an include directive. It tells the compile where to find information about some of the items that are used in your program. For example: cout, <<, >>, and cin in the above program. There are other libraries that you will use to include other items. Note that the directive always begin with #.

```
using namespace std;
```

This line will tell the compiler that the names defined in iostream are to be interpreted in a "standard way". Note that your program should work without this, as your default set up is to use a global namespace. We will discuss the namespace in more details in future labs.

```
 int main( )
{
```

Let's just say that this marks the beginning of your main program. Actually, int is used for type "integer", main is a function name and ( ) will mark the boundary of parameters. We promise that you will learn about all this very soon. Note that the { marks the beginning of the main function and } marks the end of it. In general, remember that for every open {, you should have a corresponding }.

In the next four lines:

```
    int cases_per_day, bottles_per_day;
    int bottles_per_case = 12,
```

```
 double profit_per_bottle = 0.2;  // 20 cents per bottle profit
double profit_per_day, profit_per_year;
```

We will declare the variables that we plan to use and we also define their type. We look at different variable types in the future labs. In the above four lines int is used to declare variables of type "integer". On the second line we not only define the bottles_per_case as an integer, we also initialized that to 12. On the 3rd and 4th lines, we have defined variables of type double. Note that each instruction is ended with a ;.

On the following two lines:

```
cout << "Press enter after entering each number \n";
cout << "Enter number of cases \n";
```

we have used cout to display a message on the screen. The cout statement will allow us to direct data from a variable out to the screen.

On the other hand, in the following line the cin directs data from the keyboard into a variable.

```
 cin >> cases_per_day;
```

It is important but very easy to remember what the direction of << and >> is. Note that when we send data to the screen, we send it to cout so the direction must be <<, and when we send data from the keyboard to a variable using cin the direction must be toward the variable >>.

In the sample program, we have performed some calculations:

```
profit_per_day = cases_per_day * bottles_per_case * profit_per_bottle;
profit_per_year = 365 * profit_per_day;
```

In the first line we have multiplied, *, cases_per_day by bottles_per_case by profit_per_bottle, and stored the result, =, into profit_per_day. We will learn about different arithmetic operators in future labs.

```
cout << "The store has a made : ";
cout << profit_per_day;
cout << "  per day. \n";
cout << "That means the profit for one year will be:  ";
cout << profit_per_year << endl;

cout << "Good business?! \n";
```

The statements above will display the calculated result to screen.

```
return 0;
```

return 0 in the main function means that the program executed successfully.