Introduction to C++

Lab Assignment 1

Alderliesten, Marroquim, Computer Graphics, TU Delft

Version 1, 2019-2020

Introduction

Welcome to the Computer Graphics course! In this course, we aim to teach you about the principles, theories, and practical applications of the domain of computer graphics. Over the coming weeks, you will be able to apply the knowledge you receive during the lectures in lab assignments. Each of these lab assignments will show you what happens when you take this theory to create visual content that can be rendered on a computer. Gradually, you'll be able to build visual content that has applications within the areas of medical visualization, video games, CGI, and much more.

The first assignment focuses on teaching you the ropes of C++. Although obviously not complete, this assignment will help you in discovering the differences between Java and C++, and aims to act as a *bridging assignment* between the two languages. After completing this lab assignment, you'll be ready to dive deeper into the upcoming lab assignments for this course, as well as be prepared to do some C++ programming yourself, if you become inclined to do so.

Please work through this lab assignment in order. If you have any questions, concerns, or other issues, please attend the shared lab sessions and queue for a teaching assistant. This can be done by logging into the TU Delft queue[1] and requesting a computer graphics teaching assistant. If they cannot help you or you have more serious concerns, please contact the responsible teaching assistant (TA) for the course, David Alderliesten[2]. If you do not feel comfortable messaging the responsible TA, you can contact the responsible professor for the lab, Ricardo Marroquim[3]. If you still cannot solve your problem, and it is an urgent matter (and urgent matters **only**), you may contact the responsible professor for the course, Elmar Eisemann[4].

[1] queue.tudelft.nl
[2] j.w.d.alderliesten@student.tudelft.nl
[3] r.marroquim@tudelft.nl
[4] e.eisemann@tudelft.nl

This lab assumes you are familiar with Java and have followed the lecture on C++ the course gives. This lecture can be found on Brightspace.

Task 0: Setting Up the Environment

C++, like most programming languages, requires specialized dependencies and requirements before it works. To ensure you have all required elements installed, your first assignment is setting-up the require IDE and tools to work with C++. Please read the subsection related to your operating system for a step-by-step guide. Once you have installed a working editor, you can continue to the next task.

Note that the course strongly prefers using Windows for this lab since the TAs are familiar with this OS, but we also give instructions for Linux (Ubuntu).

Device using MacOS

Due to a multiple array of factors, including occasional lack of proper driver support and hardware compatibility issues, we do not support utilizing MacOS for this lab. Please install a virtual machine using a Linux distribution, *Ubuntu* if possible, and refer to the subsection about installing on Linux. If you use C++ on MacOS for this course, you are doing so at your own risk and accept that TAs may refuse to help you with setup issues.

Device using Windows 10

If you are using Windows, we encourage using *Visual Studio* for C++ development. Besides being a strong IDE with extra features not found in the C++ toolbox, it is also an industry standard and you are likely to encounter it if you continue your studies or find a profession which relies on C++.

1: Download Visual Studio. You can do this by going to the official website[5], and installing the community edition. After downloading the installer, select the *.exe* to begin installation.

2: Install Visual Studio. Follow the steps of the program to install Visual Studio. When prompted, choose to install the community edition and select C++ libraries when required. If you have already installed visual studio, or want a visual guide for installation, please consult the official update instruction guide from Microsoft[6].

During installation, you must install additional components when prompted to do so. These components are:

- C++ CLI Support.

[5] visualstudio.microsoft.com
[6] docs.microsoft.com/en-us/cpp/build/vscpp-step-0-installation?view=vs-2019

- Windows 10 SDK for Desktop C++ Development.
- VC++ 2015.3 (v140) toolset for C++ Development. A newer version is also fine.

3: You are now ready to use Visual Studio! If you have issues with this, ask a teaching assistant to help you.

Device using Linux (Ubuntu)

Linux supports compilation, execution, and development of c++ through the terminal by installing the GNU-C++ compiler. If you are a Linux user, using the terminal should be no news to you, and we expect that you already have some experience with the OS. Linux instructions will be available for all assignments considering that you are using a recent version of Ubuntu (e.g. 18.04). Note, however, that many TAs might not be able to help you with Linux related issues, so make sure you are confident enough with your OS.

If you have not done so already, install g++ and other tools for compilation:

```
sudo apt-get install build-essential g++ cmake pkg-config
```

If you use an editor such as **Sublime**, you can compile your code using the *Build* option. For some future assignments *CMake* files will be provided for easy compilation, but for simple programs you can compile it directly from the terminal by typing:

```
g++ -o mycppprogram mycppprogram.cpp
```

and then executing the program

```
./mycppprogram
```

Final Installation Remarks

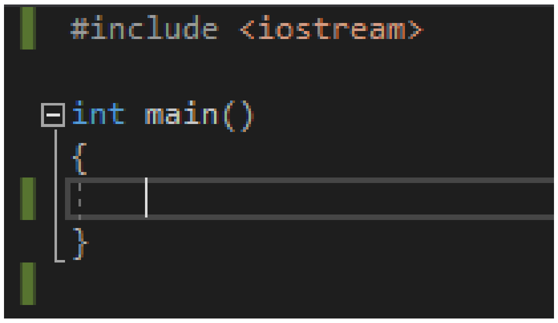
Unlike with Java, there are many things that can go wrong or cause errors during C++ installation. Please start as soon as possible, and if something goes wrong, you are allowed to discuss the installation process with your fellow students.

Task 1: Hello World++

To get started, we're going to create a C++ project from scratch and get used to some syntax and changes by making a Hello World program, with some extensions. When you complete this task, you should have a program that can print out Hello World, and take an argument, allowing some customization.

Create a New Project and Digest It

Create a new C++ project in the IDE and platform of your choice, and give it any name you'd like (if you need a suggestion: choose *Hello* or *Helo*). If you are using visual studio, select the *Console App* option in visual studio. When you are done, you should see either an empty main method (found in {*Name of your Project*}*.cpp*), and an empty main method. In visual studio, you might see a Hello World program already set up as shown below.



```
#include <iostream>

int main()
{

}
```

To start, let us quickly digest what we see here. Just like in Java, we have a main method that dictates program execution. Anything in the main method will be executed when you start running the program. The *iostream* library (or, to use correct C++ terminology, header) is included to allow for basic input and output (abbreviated as io) operations within C++.

The only major difference between Java and C++ that can be seen in this image is the fact that the main method returns an integer (declared as an int, just as in Java). However, if you do not return anything in the main method, the program will still execute! Why do you think this is? How is it possible that a method requiring an int doesn't need an int return statement? Try returning a value of 1 (just like you would in Java, with a return statement), and see what happens.

If you noticed that nothing interesting happened, you've done a good job! You probably noticed that only a value of 1 was returned and shown as output in your debugger or run time console. However, the importance of this function cannot be understated when discovering C++'s inner workings. Do a quick search on the internet and read about the integer requirement.

Printing Content to the Console

Now that you've been able to see what a simple C++ main method looks like, it is time to print content to the console. Akin to *Java*'s *print()* and *println()* methods, we want our program to show something on the console. In C++, you must import the ability to send content to the output stream, but this has already been done with the *#include* < *iostream* > declaration.

The *iostream* library provides the *cout* option with some strange syntax. This is because it is a literal stream (just like in *Java*'s buffers or file streams), and you're streaming content to it and then closing it. Syntax wise, you have three requirements for this:

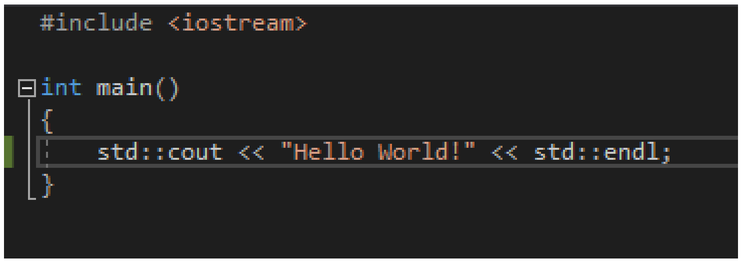
- Call the output stream.
- Send the desired content to the output stream.
- Close the output stream, to prevent memory leaks.

In C++, for many libraries and imports, you must specify the **namespace** in which the method you are calling from exists. This differs per library and framework, but for the iostream, the library is located in the standard (*std*) namespace. To make any calls to this function, you must prepend the namespace to the call. So, for this, you must call *std::cout*. If you were using another library, perhaps called DAVIDLIB, and it provided the DLB library, you would have to do *DLB::cout*.

After this call, we must append the output stream with our desired content. To do this, you must put a character array or string datatype between quotation marks, prepended by two characters which can be seen as two arrows towards the stream (e.g: <<). Make your program print out Hello World!

Finally, we need to close the output stream. This can be done by calling either *end* or *endl*. Can you guess what the difference is between the two? As a hint, consider the difference between Java's *print()* and *println()*.

If you've done everything correctly, you should now have a program that displays Hello World in the console! As a visual aide, your code should look like the code below.



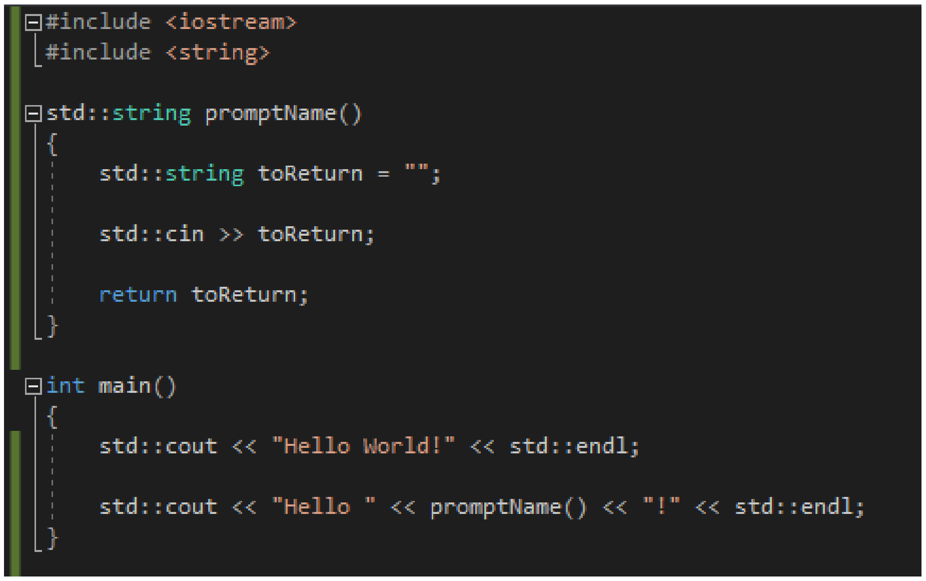
```
#include <iostream>

int main()
{
    std::cout << "Hello World!" << std::endl;
}
```

Adding the Personal Touch

Now that you've created a Hello World application in C++ and are starting to understand the C++ syntax, let's use the input stream to add some personalization by allowing a user of your program to input his or her name. Then, instead of printing Hello World, we can make the application print out their name. As an extra lesson, since we are computer scientists, let us make this functionality in a method such that we can learn the default method calling practices for C++, too!

Make a new method called *promptName*, with no arguments and a return value of a string. This method does not need an accessibility declaration (such as public or private), and should be placed **above** your main method. In future labs you'll learn how to do this properly. You'll instantly notice that the string datatype is not recognized. This is because C++ is a C-derivative language, and C stores strings as an array of characters. However, the standard library for C++ includes a built-in string datatype, which you can use. Use *#include <string>* to include this library. Just as with the *cout* command, you must prepend instances of strings with the namespace, *std*.

Now that this is done, create a string variable that you can return, and then create *cin* call that stores the input to your variable. As a reminder, note that since we are now streaming something into a variable, you must now use the opposite arrow symbol (e.g: >>). Then, return the string variable. If you now call the method in your main method, you should be able to input your name and have it be displayed. This call can be done directly in your *cout* call. If everything is correct, you should have a program that somewhat resembles the image below.



```
#include <iostream>
#include <string>

std::string promptName()
{
    std::string toReturn = "";

    std::cin >> toReturn;

    return toReturn;
}

int main()
{
    std::cout << "Hello World!" << std::endl;

    std::cout << "Hello " << promptName() << "!" << std::endl;
}
```

Task 2: C++ Exploratory Assignment

Now that you've had the chance for some exploratory C++ work, and you've had the introduction lecture for C++, its time to delve further into C++. You must complete the exercises located in the assignment folder. These exercises will be tested against a suite of automated tests, which will be given back to you for a grade. Due to the setup of the project, all your solution code must be placed within the header file (*.hpp*) of the code, whereas some tests and code execution is provided within the *.cpp* assignment files. This test code should not be changed. When executing the test code, you will see a console window containing the status of each assignment. If all five are correct, you can hand-them in through **Brightspace**, where the header file must be changed to your student number. If your student number is 42, then you must hand in 42.hpp. Do not add Task 0 or Task 1 content, they will not be graded.

Getting Started

The assignment has been created utilizing *CMake*, and adapted for Visual Studio. Please follow the setup instructions for your operating system and IDE. If you need help with this, please consult the teaching assistants.

Visual Studio: a project exists within the assignment folder containing a project for Visual Studio. Simply open this project by selecting the *vcxproj* file, and setup should do everything for you. After opening the project, you can see the solution explorer in the top right, where you can find the header file in the *header files* folder.

Ubuntu: Again, if you are using some IDE (such as Sublime) there is probably a Build option. Otherwise, just compile via terminal using the following command:

```
g++ -o Assignment_1_Project Exercises.cpp
```

Now run the program:

```
./Assignment_1_Project
```

As a third option, and if you want to warm up for future assignments, use *CMake*. *CMake*[7] is a very popular tool for cross-platform compilation. It generates a *Makefile* to compile your program taking into account all dependencies, apart from many other handy extra features. The *Makefile*, in turn, contains instructions to a *make*[8] program to compile your code. You could write directly your own *Makefile*, but the advantage of *CMake*, apart from being more readable and user-friendly, is that the same *CMake* file is able to generate Makefiles for different platforms.

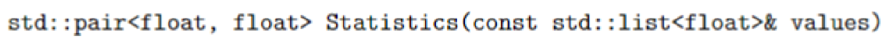
[7]https://cmake.org/
[8]https://www.gnu.org/software/make/

Before running *CMake*, it is strongly suggested that you create a build directory, to keep things organized. The following sequence of commands creates a new folder, generates a *Makefile* using *CMake*, compiles, and runs the program. Make sure you are in the directory with the *CMakeLists.txt* file before following the next steps:

```
mkdir build #creates a new directory build
cd build #enters the build directory
cmake .. #runs the CMakeLists.txt and generates a Makefile
make #compiles the program using the Makefile
./Assignment_1_Project #runs the program
```

Exercise 1 - Statistics

For the first assignment, you must write a method which provides a number of simple statistics. The input given is a list of *float* values. These values are all valid, correct values (we will not give you nasty input, such as invalid floats or other datatypes). The output should be a pair of values, where the first value is the average of all the values in the list, and the second value is the standard deviation of list of values.

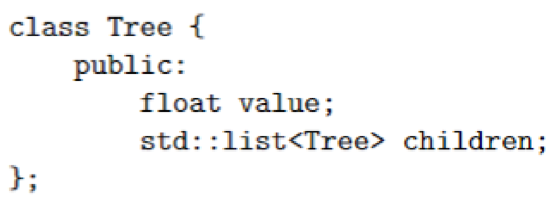


```
std::pair<float, float> Statistics(const std::list<float>& values)
```

An example could be an input list of [-4, -2, 0, 0, 0, 2, 2, 2], which should have an output of (0.0, 2.0).

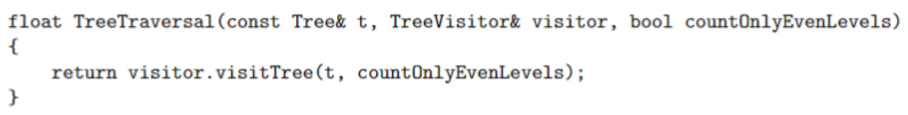
Exercise 2 - Tree Traversal

For the second assignment, you need to implement a method that will perform tree traversal. As you might recall from Algorithms & Datastructures, a tree is a data structure in which each node has at most two children, and at most one parent. In the given tree, each node has a value, defined as a float.

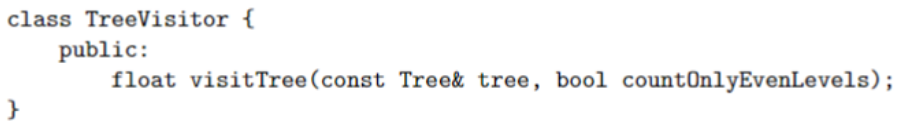


```
class Tree {
    public:
        float value;
        std::list<Tree> children;
};
```

The tree functionality is defined behind-the-scenes in the codebase as shown in the image below.


```
float TreeTraversal(const Tree& t, TreeVisitor& visitor, bool countOnlyEvenLevels)
{
    return visitor.visitTree(t, countOnlyEvenLevels);
}
```

The Treevisitor class is defined partially below.


```
class TreeVisitor {
    public:
        float visitTree(const Tree& tree, bool countOnlyEvenLevels);
}
```

Part 1: Treevisitor You need to write a function called Treevisitor::visitTree such that when this method is called, it traverses over all the nodes in the tree and returns the total value found. Thus, if you iterate over a tree with nodes worth 3, 5, and 6, you need to return the sum of total values $(3 + 5 + 6) = 14$.

NOTE: for both part 1 and part 2, you should not modify the TreeTraversal function.

Part 2: Only Even Nodes If the boolean *countOnlyEvenLevels* is true, you should only sum the tree values from nodes that are on even levels. Note that 0 (the root node of the tree) is considered an even number. Extend your Treevisitor::visitTree method to support this functionality.

Exercise 3 - Complex Numbers and Operations

As you might remember from Calculus or your high school mathematics courses, complex numbers exist partially in the real world (real), and partially in the imaginary world (im). In this exercise, you must define operators[9] for complex numbers such that any essential mathematical operation can be performed. The class Complex, which is used to store the complex numbers within our program, is defined as per the image below.

```
class Complex {
public:
    Complex(float real, float imaginary){};

    float real, im;
};
```

You are also given a *Complexoperation* method, in which you must write your implementation. The second line has been commented out to prevent program compilation errors. It is given as shown below.

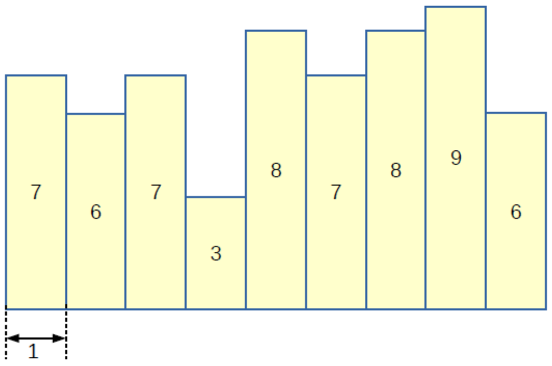
[9]Recall: implementations for functions of '+', '-', '/' and '*'.

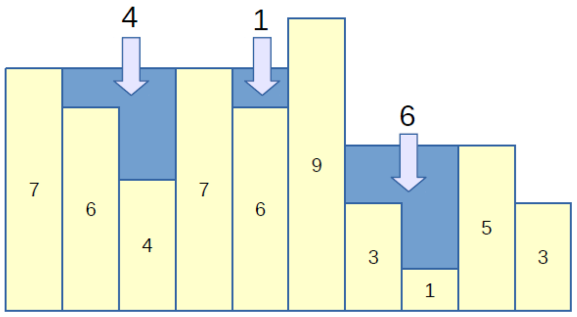
```
Complex ComplexOperation(Complex c1, Complex c2)
{
  return Complex(0,0);
//    return (c1-c2) * (c1+c2);
}
```

For this assignment, you must implement the addition (+), subtraction (-), and product (*) operators for complex numbers. Division is not required. Thus, if your input was (3 + i) + (-1 + 2i), your result would be (2 + 3i). If your input was (3 + 2i) * (1 + 4i), the result would be (-5 + 14i). You should **not change or remove** the *real* and *im* variables given, as these are used for the automatic grading of your code.

Exercise 4 - Water Levels

You are given a structure with differing heights at different indexes. You are tasked with calculating the total possible water that can be held within the structure due to indexes differing in heights. You can see how water could be held by the structure in the two images below.





You are also given a method called *WaterLevels* which takes an *std::list*[10] containing the heights at each index. The method is defined as shown below.

[10]This is an alternative implementation of the *Vector<T>* class provided by the standard library.

```
float WaterLevels(std::list<float> heights);
```

Your goal is to implement the method such that it calculates the total water that can be stored within one of the structures. Note that no water can be held at the end points (so if you have end points that are lower than their direct neighbors, it cannot hold water, such as the case with the rightmost index in the example provided above). The output must only be a float containing the water levels. For an input of [7, 6, 4, 7, 6, 9, 3, 1, 5, 3], an output of 11 is expected.

Exercise 5 - Labyrinth

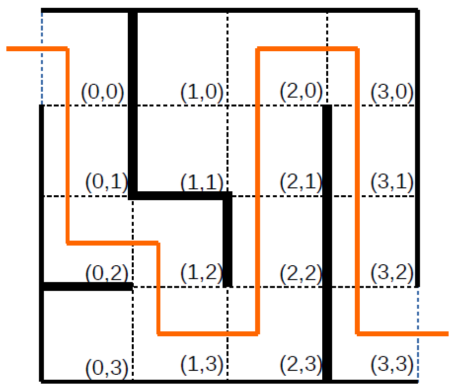
You're stuck in a labyrinth and need to find a way out. The labyrinth is of size *size* * *size*, and is always a square. You need to get from tile (0, 0) to tile (*size* - 1, *size* - 1). You can only move between adjacent squares (up, down, left, right), and you cannot move diagonally. The function you are given is defined as shown below.

```
typedef std::pair<int, int> location;

float
Labyrinth(std::set<std::pair<location, location> > labyrinth,
          int size);
```

The edges of the labyrinth have been blocked off (meaning you cannot pass through them). You are also given a set of walls that indicate adjacent pairs in the labyrinth you cannot move between. For example, if you are given the input set shown below:

```
[( (0,0) , (1,0) ),
 ( (0,1) , (1,1) ),
 ( (0,2) , (0,3) ),
 ( (1,1) , (1,2) ),
 ( (1,2) , (2,2) ),
 ( (2,3) , (3,3) ),
 ( (2,2) , (3,2) ),
 ( (2,1) , (3,1) )].
```

You will get a labyrinth that looks like this:



The method you implement should return the length of the shortest possible path. Each tile crossed counts for 1 unit of length (so crossing 5 tiles is a length of 5, 6 tiles is a length of 6, and so on). If no path exists or the labyrinth cannot be exited, you must return a value of 0. In the example given above, the shortest possible length is 13.

Submission

Please package the implementation you wrote using the given classes for task 2 and rename the header file to [studentnumber].hpp. If your student number is 42, then it should be 42.hpp. If your studentnumber is 12345, you should hand-in 12345.hpp. Please hand-in your header file through Brightspace to the Assignment 2 submission folder. This must be done before 19 September 2019, 23:59. Submissions which contains content from tasks 0 and/or 1, or submissions with an incorrect header file or more content than just the header file will get a 0. Late submissions are not accepted.

You can get help on the forums and during the lab sessions, which take place during the Shared Labs for Y2Q1. These take place each Thursday between 13:45 and 17:45.

Your grade will be published at some point based on the automatic tests. This will be published on Brightspace when it is available.