All elemental steps that will get you started for your new life as a computer science programmer.
Week A – C++ Basics

What is C++?
First thing first
- Compiler, Linker
- .h and .cpp files
Hello World – the usual way
Your first C++ Class
Some I/O
Assignment: Hello World – the CG way

Week A: C++ Basics
During this lecture there’ll be some explanation of some basic concepts of C++. It should not get too boring, so we’ll write our first program without a full understanding of what we are doing (do not worry, this happens all the time). After this first week, you’ll have your first working C++ program – wow!
What is C++?

Most superior programming language?

- **Fast** & powerful
- Many libraries and much code available
- Object oriented
- Online help / tutorials available

Everything comes at a price

- Not very intuitive / fast to program
- No Garbage Collector – clean up your own mess!
- Error prone

What is C++?

C++ is arguably the language most often used in the IT world today. It has a lot of benefits compared to all the “new” programming languages that are around. In my opinion, learning C++ is a very important step in your IT career, since many technical concepts are concealed in other languages like JAVA. Not that this is a bad thing in general, but often it helps to understand what happens at the core of things to unleash your code’s full potential.

C++ is fast. There are only a few ways to get your code even faster than to use C++. Most of them include the use of special Hardware (GPU, FPGA and the like). It is also very powerful, you have full control over the memory used by your program. C++ also won’t tell you what to do or not to do with the code.

If there is a implementation of any algorithm on the internet, it is most likely that it’s written in C++. Many very powerful libraries are written in C++ and are readily available – often for various operating systems. Before you start programming anything in C++, it’s always worth the time to check on the net if someone already did program and publish code for your problem!

C++ is also object-oriented, you can use many powerful concepts such as (multi-)inheritance that you should have learned in your basic Programming course.

There are a lot of online and offline resources that help you if you get stuck. You should make use of them often. Here is a tiny selection of sources of information:
http://www.cplusplus.com/doc/tutorial/ - I definitely recommend this for your first steps in C++

http://www.cppreference.com/ - good reference site

There are a lot more sites on the web, google knows them!

But there are also drawbacks when it comes to work with C++! Many people find it very unintuitive and hard to work with. There’s no built-in memory management for you, if you reserve memory for your data, you’re also responsible for freeing it afterwards. Most annoying is the fact that with many possibilities of doing things right, there come many possible ways of doing things wrong. You’ll see!
**Precompiler, Compiler & Linker**

Your program is (pre-)compiled and linked

- Precompile: “glue” together code
- Compile: Translate to machine-readable code
- Link: connect all compiled and external files

**Different errors at different stages**

- Not always easy to understand
- More on that next week
- For now, unzip weekA.zip from website, goto 00_hello directory, run make

**Making your code work**

Make does the compiling and linking for you

The command `make` invokes the interpretation of the `Makefile` in your working directory. This simple `Makefile` tells the compiler to compile and link the file `main.cpp` for you.

Do not worry about this too much, next week we will have some more `Makefile` fun.
Your first C++ program

For now, you should just go to the directory 00_hello/include in the weekA.zip file. Type in make and the program is compiled and linked.

Type ./hello to start it, you will just see a Hello World output string on the console. We will make things more interesting by writing a Hello World program the Computer Graphics way, i.e. we will create a 3D model of the globe and export it to a proper 3D file format.

---

```cpp
#include <iostream>

/* This is our main entry point of our application. */

int main()
{
    // This is the boring, old way of "Hello World"
    std::cout << "Hello World!" << std::endl;

    return 0;
}
```
Your first C++ class

Many concepts of Object Oriented programming work similar to JAVA (...and many do not). For now, let’s take a look in the directory `01_include/` to see how a simple C++ class is created. The first and major difference is that we need two files for our class: `MyClass.h` and a `MyClass.cpp`. It is important to know that you SHOULD ALWAYS keep your declarations and definitions in separate files. Declaration means that you only define which classes and methods exist in your code, but you do not tell people what they actually do. The concept most similar to that in JAVA are interfaces. These declarations are usually stored in *.h files.

Definitions are the actual “meat” of your code, i.e., they are the implementation of your methods. For each *.cpp File, which stores the definitions of your code, a separate *.o object file is compiled by the compiler. All declarations needed by the code in a *.cpp file have to be included via the #include command. Inclusion means that the content of the actual *.h files is copied and pasted into the *.cpp file before compilation. To avoid double includes, you should always start your *.h File with a #ifndef and #define statement like in the `MyClass.h`. It should also always end with an #endif. This way, the precompiler (who does all the copy /paste work) only includes this file once, even if is technically included several times in the code. Keep in mind that #include statements are recursive. Also, the more you include the more time it takes to compile your code.

Your first class: MyClass.h

```cpp
#ifndef _MYCLASS_H_
#define _MYCLASS_H_

class MyClass
{
private:
    int m_counter;
public:
    MyClass();
    void doStuff();
};

#endif
```
You see that the function `TestClass::doStuff()` is implemented in the .cpp file. It just prints a message to the error output of your system (i.e., to your terminal window). That’s it!
And this is how you call your first C++ class, you construct it and call the `doStuff()` function ten times.
In the folder 02structs/ you will find the example of the struct Vector3D.h. Structs are similar to Classes, except that all functions and members are public. I recommend that you only use them as data containers. More complex objects should always be modeled as classes. You may use the Vector3D as a data container to store and exchange 3-dimensional coordinates.
Your first STL data structure

In our last cost snippet in folder 03_io/ you will find everything else for your first assignment. The first thing is the usage of a vector, a data structure you will use a lot the next weeks. Vectors may store an arbitrary data structure. In the above case, it is just simple float values. You may also create a vector of Vector3D structs. The syntax would then be

```cpp
std::vector<Vector3D> vectors;
Vector3D v;
v.x = 1.0; v.y = 2.0; v.z = 3.0;
vectors.push_back(v);
```

Of course, you would have to include the “Vector3D.h” file. Although the person who implemented the vector data structure did not know of your intention of implementing the Vector3D struct, you may use his vector to store your Vector3D structs. This is possible because of the templates mechanism used by C++.

There are a lot more STL data structures, e.g. lists, trees and dictionaries. Just have a look at any C++ reference if you need more information.
Basic I/O: MyClass.cpp readFile()

```cpp
std::ifstream fin(filename.c_str());
fin >> m_title;

int numPoints;
fin >> numPoints;

if(numPoints > 0)
{
    For (int i = 0; i < numPoints; i++)
    {
        float value;
        fin >> value;
        m_numbers.push_back(value);
    }
}
fin.close();
```

Your first file input

This is just a very basic example of reading a file. All you need to have is an `ifstream` object and its `>>` operator. With them, you can open a file and read in the data piece by piece, the `>>` operator interprets empty spaces and line breaks as separators.

You can interpret the content as any basic data structures, such as strings, integers and floats (although you should know what to expect).
Basic I/O: MyClass.cpp writeFile()

```cpp
std::cout << "(MyClass) Writing file " << filename << std::endl;

std::ofstream fout(filename.c_str(), std::ios::out);
fout << m_title.append("*1.5") << std::endl;
fout << m_numbers.size() << std::endl;
for (int i = 0; i < m_numbers.size(); i++)
{
    fout << (m_numbers[i] * 1.5f) << std::endl;
}
fout.close();
```

Your first file output

ifstreams work just the other way around as ofstreams: You open a file for writing (if it does not exist it is created, but the containing directory should exist). You may fill it with any type of basic data types, line breaks are added via std::endl (endl stands for “end line”). Note that the end line symbol can be different on different operating systems and machines.
Have a look at the octaeder.ply file in the assignment_stub/ directory to see the syntax of the *.ply 3D files. At first a header is defined, listing the amount and properties of objects contained in the file. A list of 3D points follows, one point per row. At last, a list of lines is defined. The “3” stands for the three vertices contained (unfortunately, lines with two vertices are not supported by some blender versions) of the face. It is followed by the three indices of vertices from the point list. As a workaround, the second and third vertex are the same, effectively reducing a 3-vertex face to a 2-vertex line.

Play around with the .ply file and import it in Blender to visualize it:

File->Import->Stanford PLY (*.ply)