C++ Guide

Date

Contents

1 C++ and Hardware

1.1 Short History of High Level Programming Language

Machine code is basically a bunch of binary signals of 0s and 1s. Machine code was never constructed for the sake of legibility, so they are difficult to read and definitely not a good choice for programming. Imagine reading through a bunch of 0s and 1s and trying to figure out what it is trying to do. Machine code is also machine specific so the machine code on one machine is different from machine code on another machine. Copy pasting machine code from one computer to another might not work even if the computer hardware are completely the same.

In the beginning, everything that existed was the hardware and the machine code. It was kinda okish at the beginning because computers were rare and very few. However, as computers became increasingly important due to the Cold War, i.e. for analyzing nuclear explosion data and the Space Race, i.e. to calculate rocket trajectory, what payload the rocket carries is another moral and political question, it became increasingly troublesome for people to write a completely new machine code for each machine. Imagine you have to completely rewrite the code each time you copy and paste from internet just because of some very tiny difference between your machine and their machine.

Thus, people decided to first solve the intelligible issue with machine code, since no one wanted to read a bunch of 0s and 1s. Humans read words instead of 0s and 1s, so humans decided to create a translator which would translate words into 0s and 1s. Thus came Assembly, which translate characters that is easy to human to understand into 0s and 1s for the machine to understand.

Machine code	Assembly code	Description
001 1 000010	LOAD #2	Load the value 2 into the Accumulator
010 0 001101	STORE 13	Store the value of the Accumulator in memory location 13
001 1 000101	LOAD #5	Load the value 5 into the Accumulator
010 0 001110	STORE 14	Store the value of the Accumulator in memory location 14
001 0 001101	LOAD 13	Load the value of memory location 13 into the Accumulator
011 0 001110	ADD 14	Add the value of memory location 14 to the Accumulator
010 0 001111	STORE 15	Store the value of the Accumulator in memory location 15
111 0 000000	HALT	Stop execution

Figure 1: Machine Code, Assembly and description of instruction, taken from http://computerscience.chemeketa.edu/cs160Reader/ProgrammingLanguages/Assembly.html

Assembly was much nicer than machine code to write, since you could write words instead of 0s and 1s. However, while assembly is not machine specific anymore, assembly is actually architecture specific, meaning that you can only use assembly with the architecture it's supposed to work with. Different architecture, different assembly. You can think that assembly will work across hardware sharing the same general structure. Of course, what seems similar at the surface can have minute difference that can give weird, very difficult to find bugs, so it is still not recommended to just copy and paste assembly code between machines with what seems to be similar architecture. And people were not happy with that.

Thus, people wanted a code that could work across multiple different computers instead of just computers of a single architecture, i.e. a code where copying from Stack Overflow is possible. (whether moral is another question altogether) Moreover, storage has evolved allowing people to store more stuff on the computer, and nice hardware people have made assembly for their hardware come with their hardware, or just simply make their hardware use the assembly code in another piece of hardware. Since the assembly came with the hardware, people developed a new programming languages, which are even more readable, that will

1. Translate the code the user writes, i.e. in C++, Java, Python etc, eventually into Assembly (the translation from the code to Assembly might be a multistep process)
2. Resulting Assembly code gets translated into machine code
3. Machine code is fed to the hardware to be executed

These newly developed programming languages were dubbed high level programming langauges, and these include C, C++, Fortran, Python, Java, etc and appeared one after another since the 1960s. They are called high level programming languages because they require some steps to reach and produce the machine code, but they are able to be used across many different machines with many different architectures, assuming that you have set them up properly.

In simpler terms, you can copy and paste code from these high level programming languages off internet and expect them to work properly in your machine, and you are justified to leave an angry comment if they don't. Of course, I am not saying that you can disregard the system requirements, so let's say a Python code requiring multi GB of memory will probably not work on a computer with less than 1 GB memory. I am just stating that a hello world in Python will work across different machines of different architecture with a somewhat close Python version.

Different high level programming languages come with fields they are really good at, and fields that they are less good at. These different "specialization" of these high level programming languages were mostly decided by what people they were supposed to serve and also what the people who created them were dissastified with. For example, based on wikipedia, Fortran is "especially suited to numeric computation and scientific computing" because Fortran was "originally developed by IBM in the 1950s for scientific and engineering applications". For example, again based on wikipedia, "Python's design philosophy emphasizes code readability with its notable

use of significant whitespace. Its language constructs and object-oriented approach aim to help programmers write clear, logical code for small and large-scale projects". So the choice of programming language is not trivial. You should use a programming language for what it is good at, and not what it is bad at and never what it was never designed for.

Many high level programming languages were developed, but only very few of them survived to this day, and for those that survived, they must keep on evolving to remain competitive. As an example, Fortran has gone through FORTRAN I, FORTRAN II, FORTRAN III, FORTRAN IV, FORTRAN 66, FORTRAN 77, FORTRAN 90, FORTRAN 95, FORTRAN 2003, FORTRAN 2008, FORTRAN 2018, seeking to keep up with user demands and technological developments.

1.2 Compiling

C++ is a compiled language, and it is different from a scripted language. The distinction between scripted and compiled languages are not that clear cut. In essence, you could consider any language where you would usually have to click on the compile button(or the debugger button), generate a file from the code you have written and then execute the file generated from code to get the desired result to be a compiled language. You could consider any language where you would usually just click on run and it will run to be a scripted language, for example Python.

However, you should understand that while Python might seem to run with a single click of the run button, you are not actually feeding the computer hardware Python code, because remember that computer hardware only accepts machine code and nothing else. Under the hood, when you click on the run button in Python, you are actually translating the Python code through multiple different complicated steps to machine code, and then feeding the generated machine code to the hardware.

Compiler is a computer program that translates computer code written in one programming language (the source language) into another language (the target language). C++ compilers change the C++ high level programming language to assembly code. Similar to how there are always many different ways to solve a problem unless your higher ups have specific requirements, there are also many ways of changing your C++ code into assembly code, thus there are many different compilers available today for C++. Do note that the compiler will translate it honestly, and it is not smart enough like an actually properly trained translator to doubt what is being translated. There has been attempts to create such a smart compiler, but these were not very succesful. Currently, compilers will pick up if there are serious bugs or errors within your code and refuse to compile, but they will not check if the code given to them will produce your intended results, since the compiler are not psychic and cannot know your intentions.

To compile your code, you need to install your compiler of choice. Which compiler is best is very much hardware and problem dependent. Some compilers may be very good for some problems, others might be very good with other problems. Some compilers might be very good for some hardware, some compilers might be very good for some other hardware. There is no way of knowing which compiler is best for

your problem unless you benchmark it for your problem. While there are existing benchmarks for compilers published, these benchmarks are very problem specific, so they are useful if the problem you are solving follows the same steps(algorithm) as the problem that are benchmarked, but might not be so useful if you are solving a completly different problem from what it is benchmarked. It is also important to note on what hardware platform is benchmark done. For example, C++ compilers published by Intel generally run well on intel hardware, and compilers published by AMD generally run well on AMD hardware, or at least **in theory** they should run well on AMD hardware. C++ compilers published by Intel will still run on AMD platforms, and C++ compilers published by AMD will still run on Intel platforms, but they seem to run less well than if they were on their own platform. Compilers also have different support for various features in C++, but do not worry about the feature support for now, **just note that when the group you are working with uses a different compiler than your usual, you should always ascertain if there is a specific reason they are using the compiler.**

1.3 How to Compile and Compiler Flags

To compile a C++ code, if you are using a dedicated IDE, if you have a very simple piece of code with no dependencies, you can easily just click on the run button in the dedicated IDE to generate the executable. The exact location of the run button and whether the run button exist or not is specifically IDE dependent. For example, in Visual Studio, you can just click on the green arrow at the top, while in Visual Studio Code, you might have to install the code runner extension for an arrow (not green) to appear. However, this is generally not ok for larger code, where the best practice is to **compile it from the terminal**.

To compile it from the terminal, you can either open a separate Windows Powershell or cmd, or you can open a linux terminal if you are on linux. Before you do any compile, you should first try to check the version of the compiler. You might be actually interested in the version of compiler if you might need features exclusive to the newer version of the compiler, but here we are just making sure that the compiler is properly installed. A properly installed compiler will tell you the version of the compiler, while if the compiler is not present, then there is nothing to tell you the version info.

The exact code to tell the version of the compiler is compiler specific, but a quick search should give you the code to find the version. If you are using GCC, then you can write in the Powershell, cmd or linux terminal

```
g++ --version
```

If you are using clang on a Mac machine which comes by default with the Clang compiler then

```
clang --version
```

The resulting text should tell you something about the version of the compiler. If it does, this means that the compiler is properly installed.

To compile a piece of code, you can simply write

```
g++ filename.cpp
```

Note that the g++ at the beginning tells that you are going to use a GNU compiler for C++, the g comes from GNU and the ++ tells it is a compiler for C++, so if you are on a Mac, with a clang compiler, you should instead do

```
clang++ filename.cpp
```

The reason we have ++ in front of g or clang is because we are compiling C++ code, and the ++ tells the compiler that you are compiling C++ code. Many C++ compilers are also capable to compiling FORTRAN and C code. If you look through the documentation or the package manager of the GNU compiler, then you will see that it also has optios for FORTRAN and C, and some other programming languages. While we are aware that C++ has some backward compatiblity with C, it is best that you tell the compiler what code you are compiling, so if you are compiling C++ code, use g++ or clang++ and if you are compiling C code, use g or clang

Following the compile, whether using GNU or Clang, it will produce a file called a.out or some other dummy file name which you can then execute using

```
.\a.out
```

Note that in some linux machines, you might have to do the following code beforehand to tell the machine that the file is an executable

```
chmod u+x a.out
```

1.4 Compiler Flags

A compiler flag is a configurable value which may influence the compilation process. These flags are entered together with compiling command in the terminal, and tell the compiler exactly how to translate the high level code you have written into Assembly. One of the most commonly used compiler flag is the flag -o, which allows you to specify the name of the executable file generated from the compiler.

The compiler flags are compiler specific, so different compiler have different compiler flags, but a quick search should tell you what are the compiler flags for each compiler. The following flag are for the GNU compiler

We also note that compiler flags are **case sensitive** so the flag -o is different from the flag -O

To create an executable with a specific name,

```
g++ filename.cpp -o executablefilename
```

There are many other flags for the GNU compiler

- -o file Place output in file 'file'.

- -c Compile or assemble the source files, but do not link.
- -fopenmp Enable handling of the OpenMP directives.
- -g Produce debugging information in the operating systems native format.
- -O1 The compiler tried to reduce code size and execution time.
- -O2 Optimize even more. GCC performs nearly all supported optimizations that do not involve a space-speed tradeoff.
- -O3 Optimize even more. The compiler will also do loop unrolling and function inlining. RECOMMENDED
- -O0 Do not optimize. This is the default.
- -Os Optimize for size.
- -Ofast Disregard strict standards compliance. -Ofast enables all -O3 optimizations. It also enables optimizations that are not valid for all standard-compliant programs. It turns on -ffast-math and the Fortran-specific -fno-protect-parens and -fstack-arrays.
- -ffast-math Sets the options -fno-math-errno, -funsafe-math-optimizations, -ffinite-math-only, -fno-rounding-math, -fno-signaling-nans and -fcx-limited-range.
- -l library Search the library named 'library' when linking.
- -DNDEBUG turns off debugging, faster, but does not report bugs

Here, we make a note that while optimization using the various -O flags, i.e. -Os -O3 etc. might make your executable file run faster and reduce the size of the executable file, we note that this does not come without a cost. More optimized code will take far longer to compile than non optimized code. So if you want to reduce execution time, you should activate the -O flags, while if you want to reduce the compilation time, you should not activate the -O flags. Whether to reduce compilation time or execution time is ultimately problem specific.

Moreover, we note that the -Ofast flag is ignoring strict standard compliance.

We also note that while turning -DNEDEBUG might allow your code to run a little faster and reduce the size a little, you also won't get any error messages when somthing goes wrong, so it is generally not recommended to have -DNEDEBUG turned off at least during development.

1.5 Brief Intro of Computer Memory

An advatange of C++ over Python is that C++ allows direct memory manipulation out of the box, or at least direct memory manipulation on the parts of the memory allocated. Before we talk about memory manipulation in C++, it is important to understand what exactly is memory in a computer. Memory simply refer to a piece of

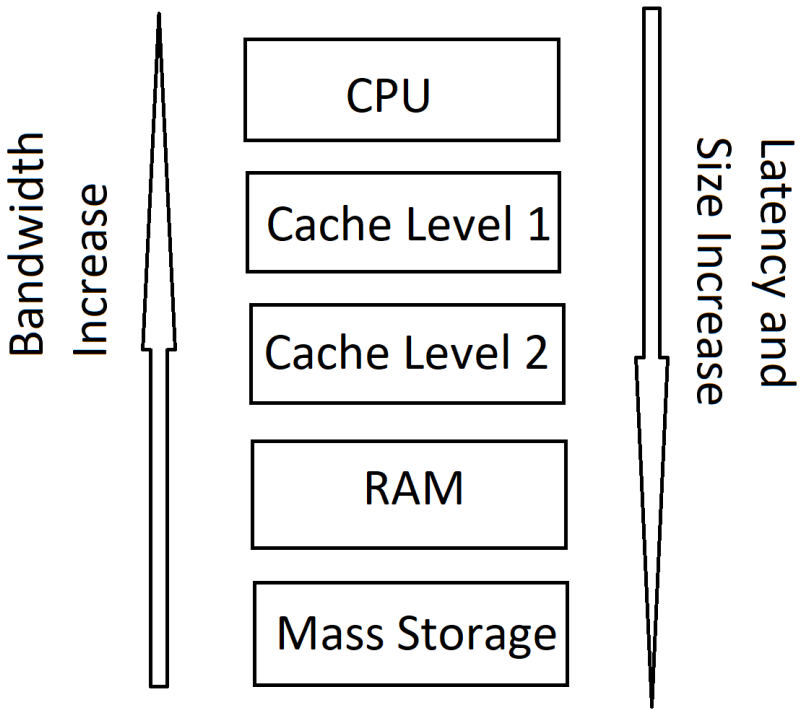

Figure 2: Comparison between Cache, RAM and Mass Storage, created in MSPaint

hardware that can store something inside of it. Pieces of hardware that can store info include the cache (which is actually a part of the CPU), RAM and the mass storage.

There is also the ROM, where the firmware to the computer is stored, but that is something that you should not play around since it might brick your computer. It is called ROM because it is supposed to be **read only**, and older ROM are literally physically made in such that a way that the only way to modify it is to break it apart and resolder it. Modern ROMs can be updated, but one needs to be very careful when updating lest your computer turns into a brick. The user, should, usually have no way of doing anything and should not do anything with the ROM. ROM is used to store firmware which is a type of software that should never be changed and is not expected to change through the life cycle of the software. This is also why you should not try to immediately grab a hardware immediately after it came out, since the hardware companies might not have tested their ROM enough. An untested ROM can lead to unexpected behaviour, and updating the ROM is quite a risky process. Companies will in theory update their ROM based on customer feedback, so that newer batches of the hardware are automatically shipped with the newer, in theory better and more stable ROM.

The cache is a piece of hardware stored closest to the CPU, it is right next to the processing unit of the CPU. The fastest transfer of data happens between the CPU and the cache with the highest bandwidth, and thus the cache stores what is CPU needs to process right now. Cache is extremely small compared to RAM or hard disk, but it can send info to the processing unit very fast. Data inside the cache is not

permanent, and data in the cache is replaced with new data once the CPU finishes processing that old piece of data.

The RAM or random access memory is another piece of hardware that is closer to the CPU than the disk but further away from the CPU than the cache. RAM is slower than the cache at transferring data, but it can hold more data than the cache, and is usually used to hold the programs that you are currently using, so basically things that the CPU might not need to process right now, but can process very quickly if you give it the instruction. Stuff in RAM is transferred to cache, and once processing finishes, the cache is wiped, and new stuff in the RAM is passed into the now empty cache again to the processed again, and this continues until processing is done. If you've opened lots of programs on your OS, then you want to quickly be able to switch back and forth between the programs, i.e. switch quickly between a browser with search engine open for your programming interview, thus you need both the browser and the app for programming interview is loaded into RAM.

Some programs use lots of RAM, some use very little RAM, and whether using so much RAM is neccesary or whether using so little RAM is a good thing is another topic altogether. If you open a picture which is very big, such as a highly detailed satellite image, then you will use lots of RAM, while if you open a small picture which is very small, such as an icon, then you will use very little RAM. This idea can be extended to C++ arrays and data structures. If you are using a very big array in C++, then you will need lots of RAM, if you are using smol arrays, then you will need less RAM. RAM on your computer is not infinite; it is a finite resource, and while modern computers come with lots of RAM and some programs come with clever tricks to allow you to view stuff larger than your maxium RAM, you should generally be wary of the size of your data structure to make sure it will fit inside of your RAM, or at least devise some clever algorithm that will decompose your very big array into a bunch a smoller arrays that will fit in the RAM.

Such algorithm are used to visualize images that are too large to fit into the RAM. Think of satellite images. A high resolution satellite image of the entire Earth will take many hundreds even thousands of the Gb of RAM if loaded completely and personal computers of 2010s - 2020s is not equipped with hundreds of Gb of RAM. A solution is to simply not display the whole image when zoomed out but only a compressed version of the image, and only display a more detailed image when the user actually zooms in. This is why you sometimes will feel a bit of lag when zomming in through very large images. You should also note that it is likely that a free image viewer will crash when trying to view multi Gb images because it is simply not designed for this, at least currently in 2020.

What is stored in RAM is not permanent, that is why if you have an unsaved file in an open program when the computer accidentally loses power, you will lose the new additions to the file. Some modern software have an autosave feature that can to an extent mitigate this, but some software might not have this kind of protection. For example, the old office software did not have autosave, so when your local power station goes up in flames, you lose the entire document or spreadsheet or presentation you have been working on. Newer office software should in theory all have some kind of autosave feature.

Finally, the mass storage, the one with the largest capacity and which is also the furthest away from the CPU. While modern mass storage can be installed much closer to the CPU using the M.2 slot or some other weird slot thing, old hard drives used to require a separate wire (SATA cable and if you are really old, the PATA cable) to connect from the motherboard, i.e. the circuit board where the CPU, cache and RAM is located to the harddrive. When I say mass storage here, I refer to both the classical HDD, and the newer SSD.

The mass storage contains all the files, images, videos etc inside the computer. If you shut the computer down, and the thing remains the next time you boot it up, it's probably in the mass storage. This is because the great advantage of mass storage over RAM or cache is that what it stored in it is permanent, i.e. it does not magically disappear if you cut off the power. Of course, you could delete stuff inside the harddrive to make it disappear, and power outages have a certain risk of damaging the mass storage and potentially breaking it. But if there is no power outage, and you shut down a computer with open programs, then what is inside the RAM and cache are lost, while what is inside the hard drive persists.

Note that the above is an extremely simplified version of how do complicated computer hardware and software work together, and is by no means complete. For example, not only the CPU has cache, SSD also have cache!

Since data structures in C++ are loaded into RAM, the main discussion on C++ memory manipulation will be RAM manipulation.

1.6 C++ Memory Manipulation: Pointers and References

Data structures in both C++ and Python are loaded into RAM, but C++ provides you the ability to modify the data structures from memory based on your needs out of the box, while Python does this for you automatically. While it might be ideal for small applications to just let Python handle it for you, it might not be ideal for HPC, a.k.a High Performance Computing. For HPC, where you try to squeeze every ounce of performance, then C++'s direct memory manipulation is a powerful tool to squeeze even more performance from what are limited computing resources.

In C++, after you have loaded your data structure into memory, in order to work with the data, the computer needs to access the RAM to read (or write) a value in memory, thus the computer needs to know where in memory to look for the value (the memory address), then can it go to that location in memory to read the value in that memory (memory content). In C++, we can actually get the memory address of a value and store the memory address, so the next time, you can use this memory address to find the corresponding memory content. These things that hold the memory address to another thing are called pointers. The exact code for pointers will be later shown, but the conceptual understanding of what is a pointer is far more important than the exact code for it.

Pointers: Variable that holds memory address of another variable.

Similar but different to pointers, C++ also has references. Reference in C++ is simply

an alias, or an alternate name to an existing variable. You can think of reference to the variable as a soft copy of the variable, though this is not strictly correct.

References: Variable that is an alternative name for another variable.

Before delving into the usefulness of pointers and references in C++, it is important to understand that pointer and references are themselves objects that are distinct from the objects they point towards. So the pointer is sth, reference is sth, and the pointer and reference is not the same thing as the thing that the pointer or reference is pointing towards. For example, in a jounral article published online, the hyperlinks and the references in the paper are themselves objects, but they are not the same object as the website or the other journal articles they point towards. This is **crucially important** to know, because many things deal differently when they are given a pointer, or a reference or an object.

C++ people will always preach about how great pointers and references are, but before believing them about how great pointers and references are, we need to understand the risk of using pointers and references, and the inconveniences brought by using pointers and references, which include:

- Greater difficulty in reading, understanding and debugging the code
- Possibility of memory leaks (more on this later)
- Possibility of buffer overflow (more on this later)

Thus, pointers and references are probably not the best tools when working with lightweight, simple code snippets, and even in large projects where performance is not an issue. However, pointers and references are really powerful tools for squeezing performance, because

- C++ function makes a new copy of what is passed into it (*except arrays where decays to pointer and C++ makes copy of the pointer*) and this can be super expensive if what is passed towards it is a very large data structure. Essentially you are using $2\times$ the memory, once for the original and once for the copy. This is a problem if your memory is limited. Passing a reference or a pointer to C++ makes C++ understand that you don't want to make a copy of the original, but you want to work directly on the original.
- C++ function can return only a single thing, and not multiple things. Using pointers and references, since you are not creating a copy of the original and working on that copy, the modifications the function does is directly done on the original. At the end of the function, you could simply get the modified originals by telling the computer to use the very same pointer or reference to the memory location. Of course, you could say that you can return multiple things by simply grouping all the things that you want to return together into a single object, but this is actually highly inefficient, since when you are grouping them together into a single object, C++ actuallys copies each individual thing you want to return into the object that is supposed to contain them all, easily

leading to insufficient memory if your data structure is large. A good practice is that before you are passing the data structure into the function, you should make a copy of it and just save it somewhere, for future reference and debugging purposes, because passing by pointers or reference results in the original, and not the copy being modified.

1.7 Buffering

What is the concept of Buffering?

Imagine you are graded upon how quickly you can write some words, or how quickly you can read some words. Would you do your task faster if you were given each character of the word you had to write down one by one, or would you do your task faster if you were given all the characters of the word you had to write all at once? Would you do your task faster if you were given each character of the word you had to read one by one, or would you do your task faster if you were given all the characters of the word you had to write all at once?

A buffer for C++ is basically a grouping of stuff you have to do. For writing operations, buffer collects the characters that C++ has to write, and then passes them as a group of characters towards C++. If C++ was given instructions to write it out, C++ will write out the group of characters that are passed to it. After the buffer has given C++ what to write, the buffer is flushed, allowing the next group of characters to be collected at the buffer before passing it to C++. **If** C++ was given another instruction to write the new thing the buffer passes it out, then C++ will write out the group of characters that are passed to it, so on and so forth. Buffers allows C++ to write faster.

For reading operations, buffer collects the characters that C++ has to read, and then passes them as a group of characters towards C++. If C++ was given instructions to read it out, C++ will read out the group of characters that are passed to it. After the buffer has given C++ what to read, the buffer is flushed, allowing the next group of characters to be collected at the buffer before passing it to C++. If C++ was given another instruction to read the new thing the buffer passes it out, then C++ will read out the group of characters that are passed to it, so on and so forth. Buffers allows C++ to read faster.

C++ uses buffers for reading and writing, but the concept of buffering can be easily extended to another operations, such as arithmetric operations.

However, buffers can sometimes mess up!

When buffer collects the characters, how does the buffer know when to terminate the collection? Buffer cannot just collect infinitely, thus buffers can have a delimiting character. For example, if the word "ice-cream" was given to the buffer, the buffer for the C++ operation of "cin", the buffer will consider this to be a single group of characters. However, if the word "ice cream" was given to the buffer, the buffer for the C++ operation of "cin" will consider this to be a two group of characters, because the delimiting character is white space and will only pass the first group of

characters, leaving the second group in the buffer. You can use another "cin" to also get the "cream" in another string. The next page shows an exmample.

How can we deal with this problem? A simple, maybe not best way is to use another operation called getline(not empty here), whose buffer does not consider whitespace to be the delimiting character.

Do not worry about what is iostream, namespace and the main, I will get to it later.

An example of buffer messup

```
#include <iostream>
using namespace std;
int main(){
    string word1;
    cin >> word1;
    cout << word1;
    return 0;
}
```

input ice-cream output ice-cream

input ice cream output ice

```
#include <iostream>
using namespace std;
int main(){
    string word1, word2;
    cin >> word1;
    cin >> word2;
    cout << word1;
    cout << word2;
    return 0;
}
```

input ice-cream output ice-cream

input ice cream output icecream

```
#include <iostream>
using namespace std;
int main(){
    char word1[1000];
    cin.getline(word1, 1000);
    cout << word1;
    return 0;
}
```

input ice-cream output ice-cream

input ice cream output ice cream

This does not mean that the cin.getline(not empty) buffer is flawless, it just means it can solve the problem that cin default buffer had. Read the C++ docs and think of situations when cin.getline(not empty) can produce surprising results!

References

Python(Programming Language)

Fortran

Figure 1

2 C++ Object Oriented Programming OOP

C++ is an object oriented programming language, meaning that objects and object classes, further referred to as classes, are a very important part of C++.

2.1 What are Objects and Classes?

Integers, variables, floats (decimals stuff in another name), long integers (integers that are able to contain more digits than your usual integer), strings (a bunch of characters) are all classes, and specific instances of these classes are called objects.

Class is a template for objects. For example, the Integer class is a class that serves as a template for all specific instances of the class Integers. We can create a specific instance of this integer and then that will be an object of the class. For example, we can say that 5 is an instance of the class integer and thus an object of the class integer. Any object in C++ is created from a class. C++ comes preincluded with many classes, and allows the flexibility to create your own classes. Of course, you can create objects based on the class that were user created.

Classes can not only create objects, but can also have class methods, which you can think of as snippets of code that can be executed upon the object created from the instance of the class, that you can very easily call upon. There are also static methods, which act on the class as a whole, but more on that later. Classes preincluded in C++ come with their own preincluded method; these methods are snippets of code which are most often used with the objects created from these classes. You apply the method on the object created from class, and not on the class itself. Of course, for user created classes, you can also define your own class methods.

For example, the class string has a method sizeof() that can be easily called by using stringname.size() which tells you the number of bytes in the string (not the number of entries in the string). So for example, if my string was something like "abcd" then the stringname.size() will give me the number of bytes in the string "abcd".

Methods can modify the class object, but can also simply tells some information about the class object. The size() method for the class string called using stringname.size() simply tells the number of bytes in the string. The insert(not empty here) method for the class string called using stringname.insert(not empty here) modifies the class object, i.e. the stringname. Methods can also have arguments inside of them; the insert method needs to know where to insert and what to insert, which are specified in the (not empty here). For example if I want to insert string "ABCD" to the string "abcd" after b, so the index position 2 then I would have to write

```
"abcd".insert(2, "ABCD")
```

The takeaway is that C++ has many preincluded classes. An object is an instance of the class and created based on the class. Classes have methods, which are the most commonly used snippet of codes with the object of the class. Methods except the Static methods are used upon the object of the class, and not on the class itself.

3 C++ Basics

3.1 The Absolute Minimum of C++

I am assuming here that C++ has been succesfully installed on the system

Since C++ is a compiled language.

What is the minimum amount of code in C++ so that it will compile succesfully?

```
int main(){
     return 0;
}
```

Does this code do anything? **ALMOST NOPE**

Does this code compile succesfully **YES!**

Would anything less than this compile succesfully? **Maybe?**

Some modern compiler will allow you to further delete the line with "return 0;", but **NO MORE THAN THAT!**. Some older compilers might not, so for the sake of backward compatibility, "return 0;" is included in absolute minimum.

What is the meaning of this absolute minimum code?

This code begins with a function with return type "int" and the name "main" with no arguments passed into it "()" and the container for the function "" contains a single line of code "return 0;"

This just sounds like some mumbo jumbo, but it will be clear with progression. For now, just know that this is the absolute minimum of code that will compile succesfully for C++, so in case you have bugs, you could try this to see if the bug comes from the code or from the C++ installation. Generally, if this compiles succesfully, then you should trust that your compiler is ok, and it is your code that is causing the bug. Out of the various bugs I have encountered, 99 out of 100 come from the code I have written and not from the compiler.

Now, we are going to stick with writing inside of the function main() to get some simple concepts of C++ through.

To write new lines of code in C++, always write ";" at the end of the code, it tells C++ to end that line of code, except if you are **ending a function, or a loop, or when including a library** then it's considered bad practice.

3.2 How to get libraries in C++ ?

Just like in Python, where you can import libraries, C++ also allows you to import libraries. However, operations in Python, or some other programming language that comes out of the box might require library import in C++, such as the printing to console functionality. To take input from console, or to output to console, C++ requires the library called iostream. Iostream, a.k.a input output stream is imported as such:

```
# include<iostream>
int main(){
     return 0;
}
```

Various libraries provided various different functionalities, but note that **things that might come preincluded in other programming languages might require a library in C++ to work.** Other common libraries include cmath, vector, string, etc. If I want to have these libraries then,

```
# include<iostream>
# include<cmath>
# include<vector>
# include<string>
int main(){
     return 0;
}
```

The functionalities of the most common libraries in C++ can be found with a quick search, and examples of the usage of these functionalities can also be found. In the codes later, I might include no library whatsoever if the functionality provided by the library is not needed, and it is good practice to only include libraries that you might need, or that you might possibly need in future developments. I might also forget to include the neccesary libraries.

We note that I have place the include at the beginning of the file. This is usually how include is done, at the beginning of the file. While you could place the include elsewhere, this can lead to bugs, and might make the person looking at your code very confused where are the includes, since they reasonably expect them to be at the beginning of the file.

3.3 What is "using namespace std;"?

"using namespace std;" is a line of code often seen during quick and short snippets of C++ code, and it simply means that you are using the standard namespace of C++. As said before, C++ uses libraries for a lot of things, and C++ has come a long way before anything got standardized, meaning there can be naming conflicts between C++ libraries. To prevent library conflicts, C++ requires that you actually specify the library of origin otherwise C++ gets confused at even the simplest of commands.

For example, for the simple "cin" and "cout" it will need to be std::cin and std::cout, so that C++ knows that it is the standard cin and cout, and not cin and cout from some other library. This is very inconvenient when it comes to small pieces of code where you are only going to use the standard libraries of C++, and not some weird external library. However, this can be very useful if you are going to need many different libraries and there are naming conflicts between. Of course, you still have to include the libraries that these come from, i.e. for the cin and cout you still have to include the iostream library.

While C++ people cannot ensure that there will be no naming conflicts between the myriad of libraries of C++, they can ensure that the standard libraries of C++, i.e. the libraries that is within the C++ standard, do not have naming conflicts between themselves. Thus came the standard library, which is a library which has the most common commands in C++, and which is controlled to have no naming conflicts in it. To prevent the hassle of telling C++ to use the standard library each time, you can use "using namespace std;" which automatically tells C++ that all the commands are going to be from the standard library, so basically you are telling C++ that every "cout" "cin" etc. are going to actually be "std::cout" "std::cin" "std::etc.".

An example:

```
using namespace std;
int main(){
     return 0;
}
```

However, the problem with "using namespace std;" is that it defaults everything to use the standard library, which might not be ideal. What if you want some commands to be coming from the standard library, and everything else to not be coming from the standard library? For example, if we only want "cout" command to be coming from the standard library? Then instead of "using namespace std;" we can write

```
using std::cout;
int main(){
     return 0;
}
```

Note that the above will make all "cout" to be "cout" from the standard library. However, for other commands that are not specified, we need to specify which library they come from. For example, if we want to use the "cin" command, we still have to write "std::cin". To make all "cin" also come from the standard library, we can simply write "using std::cin;"

```
using std::cout;
using std::cin;
int main(){
     return 0;
}
```

We note that codes like "using namespace std;" "using std::cout" and ""using std::cin" have scope, meaning they have a place where they are useful, and outisde their useful

place, they have no effect. If they are at the beginning of the file, without being in any "{}" then the abbreviation caused is for the entire file. However, if they are within any "{}" then the abbrevation caused by them is only valid inside the {}. Thus, if I use "std::cout" with {} around it, then for any "cout" inside the same set of {} ,not different of set {}, C++ will know that it is actually "cout" from the standard library, while for anything outside the same set of {}, C++ will not know from which library does "cout" comes from, and you will need to specify. This allows the flexibility of switching libraries between different {}.

3.4 Declarations and Definitions

Unlike in other programming languages, C++ does not generally assume to know what is the variable supposed to be from the value the variable is assigned to. For example, in Python, if you write "x = 5.", Python automatically thinks that the variable "x" is assumed to be float. C++ does not do that. You have to explicitly tell C++ what the variable "x" is supposed to be. Telling C++ what is the type of a variable is called a declaration. When doing a **declaration**, you can, but do not neccesarily have to tell C++ also what value is that variable, i.e. you could do

```
using namespace std;
int main(){
      int x;
      return 0;
}
```

Here, I am just telling C++ that x is a variable and x is a variable of type integer. However, I do not have to tell C++ what exactly is the value of x. C++ will just seemingly assign some seemingly random value for the int x (it's not actually random, it just seems random)

Following the declaration, I can of course tell C++ what is the value of x and tell C++ to discard that random value it gave to x, and instead use my value for x. However, I need to be sensible, meaning that if I tell C++ that x is an int, my new value for int that I give to C++ should be an integer, An example is shown below.

```
using namespace std;
int main(){
      int x;
      x = 5;
      return 0;
}
```

Here, I am telling C++ that x is a variable and x is a variable of type integer. Then in the next line I have told C++ that x, which is an integer, will have value 5.

I can of course also do the declaration and the definition in a single line of code.

```
using namespace std;
int main(){
      int x = 5;
```

```
    return 0;
}
```

I can also declare multiple variables and I note that declaration without definition results in C++ assigning seemingly random values to the variables x, y and z

```
using namespace std;
int main(){
    int x, y, z;
    return 0;
}
```

I can of course also do multiple declarations and definitions together in a single line of code.

```
using namespace std;
int main(){
    int x = 50, y = 100, z = 314;
    return 0;
}
```

Thus, in C++, Declaration and Definition are different. They can be done in a single line of code, but you can also declare and not define. However, to define, you must declare it first, not neccesarily right above, but somewhere reasonable at least.

We also note that if you have declared an integer variable, just actually assign a float to your integer variable, C++ will convert the float to int when assigning. This can be great if you intend to do it, but can be troublesome if that's not your intention. It is also considered bad practice since it can be a rather annoying bug.

3.5 C++ Numbers and their Limits

In C++, several variable types can be used to store whole numbers, including "short", "int","long" and "long long" etc. There is also "float" and "double" variable type for floating point numbers (it's called floating point because the decimal point is like floating around between the numbers)

Because a computer has limited memory, a computer cannot store unlimited number of values. Every variabletype for storing numbers in C++ is given a number of bytes to store, and anything going beyond that cannot be stored in that variabletype. Thus C++ can only store finite numbers in its variabletypes. Storing a number larger than the limits of the type will either result in the compiler giving you a warning, or worse, the compiler not giving you a warning, and giving you a seemingly right but actually wrong result. Thus in C++, it is always important to have an idea what is the maximum possible value for the number or the minimum possible value for the number, and use the correct variabletype for it.

C++ variable types for numbers can be signed or unsigned. Signed means that one bit (not bytes) will be used to store the sign, while unsigned means that no bit will be

used to store the sign. Thus, unsigned can only store positive numbers, while signed can store both positive and negative numbers. While unsigned and signed have the same range, half the range of signed is in negative and the other half in positive, while the entire range of signed is positive.

Note that the limits are ultimately OS and compiler dependent and sometimes hardware dependent, so the table below is just a general guideline for what range the variabletype is suitable. Note the values in the table are actually smaller, but pretty close to the limit. The table might not hold true for your specific OS and compiler. Check the specification of OS and compiler for details.

variable type	**limits**
short unsigned	$0 \rightarrow +6e4$
short signed	$-3e4 \rightarrow +3e4$
int unsigned	$0 \rightarrow +4e9$
int signed	$-2e9 \rightarrow +2e9$
long unsigned	$0 \rightarrow +4e9$
long signed	$-2e9 \rightarrow +2e9$
long long unsigned	$0 \rightarrow +1.8e19$
long long signed	$-9e18 \rightarrow +9e18$

For any number beyond such limits, of course there exists external libraries in C++ which have other variabletypes in them that allow you to store even these numbers beyond these limits. **However, you should really question yourself if you actually need to do operations on numbers larger than 10 quintillion? or maybe you can optimize it a bit so that C++ does not have to deal with such large numbers? Since large number arithmetic is really expensive!**

Since the limits seem to be OS dependent, below are snippets of code you can use to find the limits of the variabletypes.

```
#include <iostream>
using namespace std;

int main(){
      cout >> std::numeric_limits<short>::max() >> "\n";
      cout >> std::numeric_limits<short>::min() >> "\n";
      cout >> std::numeric_limits<int>::max() >> "\n";
      cout >> std::numeric_limits<int>::min() >> "\n";
      cout >> std::numeric_limits<long>::max() >> "\n";
      cout >> std::numeric_limits<long>::min() >> "\n";
      cout >> std::numeric_limits<long long>::max() >> "\n";
      cout >> std::numeric_limits<long long>::min() >> "\n";
      return 0;
}
```

There are also limits to floating point, i.e. "float" and "double" in C++. Unlike the other variabletypes, **"float" and "double" can have infty in C++,** but this infty does not actually mean infinity, it just means that the number is way too large.

"float" allows precision until the 6th significant figures, "double" allows precision until the 15th significant figures and "long double" allows precision until the 18th significant figures. Of course, again, you can use external libraries for even more significant fitures, but you do need to think if such precision is justified or neccesary.

To find the number of significant figures for float and double and long double, you can write the following snippet

```
#include <stdio.h>
#include <float.h>
#include <iostream>
using namespace std;

int main(void)
{
  cout << FLT_DIG << "th sig_figure for FLOAT" << "\n";
  cout << DBL_DIG << "th sig_figure for DOUBLE" << "\n";
  cout << LDBL_DIG << "th sig_figure for LONG_DOUBLE" << "\n";
  return 0;
}
```

3.6 What are Operators and Operands?

C++ also have operators and operands. Common operators in C++ include + − ∗/% <<>>= What these operators operate upon are called operands. Different operators accept different operands, and may work differently depending on the operand. For example

```
"five" + "five"     \\ strings + does Concatenation fivefive
5 + 5               \\ integers + does Addition 10
"five" + 5          \\ Error, sumimasen nani?
```

Here, when the operator + is used on two integer, C++ understand that it is supposed to add them up. When the operator + is used on two strings, C++ understand that it supposed to concatenate the strings together. Of course, there are other ways to concatenate strings. When the operator + is used on a integer and string, C++ gives you an error because it gets confused and does not know whether to do addition (since there is an integer) or to do concatenation (since there is a string).

Since C++ allows you to define your own classes, and create objects from your own classes, you can not only define class methods, but also class operators and operands. After all, int is also a C++ class, and the symbol "+" can be used between two ints because the symbol "+" is defined in the class int to mean the operation of addition. We will see this later too.

3.7 Console Input Output Error using iostream

An extremely useful library in C++ is the iostream library, which provides the "cin", meaning console in, which allows you to read input from the terminal, and the "cout", meaning console out, which allows you to print to the console. There is also the "cerr", which is console error, which is used for getting errors and debugging purposes.

We note here that the default buffer for "cin"'s delimiter is the whitesapce character, as discussed in the buffering section. We also note that without "using namespace std;" or something equivalent, then we will need to write "std::cin" and "std::cout".

3.7.1 cout

An example for cout, which will print some string of characters

```
#include <iostream>
using namespace std;

int main(){
     cout << "This is some string of characters" << "\n";
     return 0;
}
```

You might have noticed that

- << is used to separate objects in cout (of course we could have written "This is some string of characters \n", but whether to separate them or not is situation dependent)
- what is contained within "" is printed out as a string, so C++ recognizes what is inside of "" as a string. However, what is inside " is instead recoginized as a character array.
- \n is not printed out, why is it there? Because \n is special escape character, which tells the computer we should have a new line. You might also see endl used instead of " \n " in order to have a new line. For simple snippets of code, these are essentially the same, but there are still subtle differences.
- we have printed some strings here, but what can cout print, and what can cout not print?

What are escape characters?

Escape characters are special characters used to tell the computer to do sth. In particular, the newline and the tab character are very useful for formatting the console output.

Escape Character	Purpose
\n	Newline character
\t	Put a tab
\0	End of a C(not C++) string
\"	Allow typing of "
\'	Allow typing of '

What is the difference between \n and endl?

If you use endl, each thing before each endl is subsequently queued into subsequent output buffers, which are subsequently flushed. For example, if you are trying to cout the first 5 letters on each line, if you use endl, you will first have the letter a queued into the buffer, then the letter a is flushed, then you have letter b queued into the buffer, then the letter b is flushed, then you have letter c queued into the buffer, then the letter c is flushed, then you have letter d queued into the buffer, then the letter d is flushed, you have letter e queued into the buffer, then the letter e is flushed.

If you use \n, everything will be queued into the output buffer, meaning you are essentially throwing a \n b \n c \n d \n e \n into the buffer, and then the buffer with a \n b \n c \n d \n e \n is flushed. Notice here that you are flushing only once, instead of five times if you used endl. Then your console will use \n to know when there is a newline.

Thus, **\n is faster than endl**. From the example \n only has to flush buffer once while endl has to flush buffer many times.

However, **endl is safer than \n**. Let's consider the case where there is an illegal output in one of the five output letters in the example. endl will still show everything that is before the erroneous output, since it queues then flush, queues then flush. \n will not be able to show anything, since everything is loaded into a single buffer, even the erroneous output, and it will fail to load into the single buffer.

You can think of it like a factory making some products. endl is like making each individual product and then checking the product is alright, then sending the product off. \n is like taking a bunch of individual products, putting them together in a single box, and then checking that every product in the box is alright, before sending the box off. When the products are all of good quality, then you can imagine that it would be faster to send off a box products compared to sending each product individually.

However, what if there is a defective product somewhere in the middle of a batch of products? Well, endl checks if every product is alright, and it will send the product off if the product is alright, so the alright products before the defective product will still be sent before endl encounters the defective products and stops the production line. The customer will still receive some products.

With \n, a bunch of individual products is placed into a single box, and then the contents of the box is checked. With the defective product inside the box, the box will not pass the quality check, and the entire box will not be sent off, so the customer will receive nothing.

For speed use \n; for safety, use endl

What can cout not print?

While cout is ok with printing strings (as shown above), int, other variables that stores individual number, cout is unable to print all objects in C++. cout cannot directly print vectors, arrays, functions etc. directly. To print elements inside a container, you have to loop through the container. This is very different from print statement in Python, which can print list, arrays etc.

The snippet of code below shows the looping over an array to print out the elements of the array.

```
#include <iostream>
using namespace std;

int main(){
    int A[3] = {1, 2, 3};
    for(int i = 0; i < 3; i ++){
    cout << A[i] << "\n";}
    return 0;
}
```

3.7.2 cin

An example of cin, which takes in the user input into a variable called variablename of type int (or any other valid varaible type)

```
#include <iostream>
using namespace std;

int main(){
     int variablename;
     cin >> variablename;
     return 0;
}
```

3.7.3 cerr

cerr is very similar to cout, but it used primarily for displaying or storing error messages.

An example usecase is that cout gives the result, while cerr gives the error, and these results go into a file to store results, while errors goes into another file for storing errors. Sometimes, the user might be completely disinterested in the error and simply wants results, so it is good for debugging to separate the error messages from the results. i.e. if the user encounters a problem, the user can simply send the file generated from cerr instead of cout, since cout might be very big. Thus, it is always a good practice to put error messages in your code to allow for easier debugging later on.

4 Functions in C++

Here, all functions are named with functionname, but feel free to change the name as needed, functionname is just a placeholder, just remember to change the name also everytime you call the function

If you encounter bug, then look at the Common Pitfalss subsection

Functions are an essential part of C++. To specify a function in C++, you need

- Variabletype of what is returned by the function
- Name of the function
- () to hold the arguments of the function
- {} to hold what the function is supposed to do which needs to have a return statement within if anything is to be returned

Note that if the variabletype for the function can be void, meaning that there is no return to the function, so you can skip the return statement in {}

It is important to understand in C++ that main(){} is also a function, and in fact it is a special function. It is special because no other function in the cpp file is allowed to have the name main, i.e. there can only be a single main in a cpp file. This is because main keyword indicates to the compiler that what is inside of main is what the compiler is supposed to execute. Thus, the compiler will only compile what is inside of main, and everything that is called inside of main, and ignore everything else. If you have two main functions, the compiler gets confused as to what it is supposed to compile.

Functions need to be declared before the main, but can be defined elsewhere. **Note that functions in C++ needs to be both declared and defined if you want to ever call the function**, and cannot simply be declared, because C++ gets confused on how to assign random stuff to a function.

4.1 Simplest Possible Function in C++

An example of the simplest function defined before main. Let's say that the function's return variable type is void, so that we can avoid needing a return statement in {} because void simply tells C++ that the function returns nothing. We note that () and {} can, but do not neccesarily have to contain sth in them; they simply need to exist so that C++ knows where to look for to find the arguments of the function and the code of the function. If C++ find nothing in them, C++ is content that it got nothing, and C++ understands that it is your intention to not have anything in it. Otherwise, C++ gets confused because it cannot find the () and {} and it does not know if it is stupid and thus cannot find things or is it your intention to not have anything in them. **Pls No bully C++**

```
using namespace std;

void functionname(){};

int main(){
    functioname();
    return 0;
}
```

4.2 Declaring a function but Defining it elsewhere

We can also declare a function before main and define it after main. However, you need to declare a function before defining it, i.e. you can declare a function but not define it, but you cannot define a function without declaring it. Also, trying to call a function which only has a declaration and not a definition will result in an error, because C++ gets confused since it cannot just assign random stuff to the function, it does not know what to do when the function is called. The code below shows a correct example of declaring a function, then defining it elsewhere.

```
using namespace std;

void functionname();  // Declare before main

int main(){
        functionname(); // Call during main
    return 0;
}

void functioname(){};  // Define after main
```

We note that to declare a function, we only need

- Variabletype of what is returned by the function
- Name of the function
- () to hold the arguments of the function

Thus we know that these are the things that C++ uses to identify and differentiate one function from another. We will return to the importance of these in the Function Overloading part.

4.3 Return sth from a function

The snippets of code above all had return variabletype to be void, thus they do not need a return statement. However, if any other variabletype is used, then the function will need a return statement. Thus, the code below will give an error.

```
using namespace std;

int functionname(){};

int main(){
    functioname();
    return 0;
}
```

To have a return statement, we can simply do :

```
using namespace std;

int functionname(){
        return 0;
};

int main(){
    functionname();
    return 0;
}
```

4.4 Arguments in a Function

Functions can also take in no argument (as shown above), a single argument or multiple arguments. C++ requires that if you are using arguments, you need to declare the variabletype of the arguments in the (). Thus, if you had 1 argument, the syntax will be functionname(varaibletypeofargument1 argument1), if you had two arguments the syntax will be functionname(varaibletypeofargument1 argument1, variabletypeofargument2 argument2), and you could probably imagine what the syntax for 3 arguments, 4 arguments etc will be. **We note that functions can do, but do not need to do anything with the argument they take in.**, though you do have to think about the raison d'etre of the argument.

For example, a function with a single argument, let's call this argument x which has an argument type of int will be

```
using namespace std;

int functionname(int x){
        return 0;
};

int main(){
    functioname();
    return 0;
}
```

For example, a function with 2 arguments, with one argument called x, which has variabletype int, and another argument called y, which has varaibletype float is:

```
using namespace std;

int functionname(int x, float y){
        return 0;
};

int main(){
    functioname();
    return 0;
}
```

Of course, you can also do operations with the arguments within the function, and return the result of such operations.

```
using namespace std;

int functionname(int x, float y){
        x = x + 1;
        y = y + 10;
        return x;
};

int main(){
    functioname();
    return 0;
}
```

We warn that if you are making addition of an integer and a float, and then assigning the result to an integer, the result will be rounded before being assigned to the integer

4.5 Common Pitfalls when returning

4.5.1 No Nested Function

We note that unlike Python, C++ does not support nested functions, meaning that you cannot declare or define a function inside another function; however, you can call another function though. The code snippet below is incorrect and will give an error because I am seeking to define a function inside of main. Remeber main is also a function, just a special one!

```
using namespace std;

int main(){
    void functionname(){};
    return 0;
}
```

I can however do the following, where I define and declare a function outside of main, and then call the function inside of main

```
using namespace std;

void functionname(){};

int main(){

        functionname()
    return 0;
}
```

4.5.2 Return a single object

We note that you can only return a single thing using C++ functions. Of course you can create a container to contain multiple things that you want to return, but as stated in the C++ Memory Manipulation, this is terribly inefficient. Thus, the following code trying to return 2 things is incorrect

```
using namespace std;

int functionname(int x, float y){
        x = x + 1;
        y = y + 10;
        return x, y;  // YOU CANNOT RETURN TWO objects
};

int main(){
    functioname();
    return 0;
}
```

4.5.3 Autocast during return

We also note that when you return something, you are by default actually casting. This means that should you try to return a float as an int, then the float will be casted into an int, i.e. C++ will automatically try to convert the float you passed for return into an integer and thus you will lose all the numbers after the decimal of the float. For example if you try to return 2.5 which is a float as an int, then you will not return 2.5, you will return 2. Thus it is very important when writing your function to think about what do you want the function to return, lest C++ autocasts it into sth weird. You should also note that while C++ can autocast a float into an integer, C++ cannot always cast a variabletype into another, i.e. C++ does not know how to cast a string into an integer, so if you give C++ the string "five" it will not cast it into the integer 5.

An example of the autocast pitfall is shown below

```
#include <iostream>
using namespace std;

int functionname(){
  return 2.5;
};

int main()
{
  cout << functionname() << "\n";
  return 0;
}
```

Here the function is supposed to return the number 2.5, but if you actually run, you will get 2 instead of 2.5 due to autocasting to an integer, because the return variabletype of the function is int.

```
#include <iostream>
using namespace std;

int functionname(){
  return "five";
};

int main()
{
  cout << functionname() << "\n";
  return 0;
}
```

This will not even compile, because C++ does not know how to cast "five" which is a string into an integer.

4.6 Stack, Heap, Scope

While this isn't exactly a pitfall since C++ is actually doing what it is supposed, it is nevertheless important for the sake of performance.

Here, we need to introduce the concept of stack memory and heap memory, which are concepts discussed in greater detail later in Dynamic Memory Allocation.

The heap memory is the memory used by programming languages to store global variables. The stack memory is the memory specific to a part of a programming language, such as a function in C++, which is used instead of store local variables.

In C++, variables defined globally are stored inside of the heap memory. When a function is called in C++, some memory is allocated to the function, called the stack

memory. The function can take as arguments the global variable, and **although the variablename inside the function and the variablename outside the function can be the same, it is important to note that they are not actually the same. The variable outside the function lives in the heap memory and the variable inside the function lives in the stack memory. When you are using that variable outside the function, you are actually using the variable on the stack memory; when you are using that variable inside the function, you are actually using the variable on the heap memory. Changes done on the variable in the stack memory will not affect the variable on the stack memory, and changes done on the varaible in the heap memory will also not affect the variable living in the stack memory, unless of course the return of the function is then assigned to the variable in the heap memory using the assignment operator.**

Stack memory is for each and every function. So variable inside the stack memory of one function can have the same variablename as another variable inside the stack memory of another function, but they are actually different. Changes to the variable in one function will not affect the variable in the other function, unless there is sth linking the functions together

It is very important to understand that objects inside of functions, except for arrays (which will be discussed later), only live within the function itself, and do not live outside the function. In other words, these objects only live within the **scope** of their respective function.

4.6.1 Stack and Heap, separate

Below is a code snippet demonstrating that what is on the stack memory only lives within the stack memory. The output obtained, without the text, is 10, 30, 10.

```
#include <iostream>
using namespace std;

int functionname(int x){
  x = x + 20;
  return x;
};

int main()
{
  int x = 10;
  cout << "This is the value of x before " << x <<"\n";
  cout << "This is the return of the function ";
  cout << functionname(x) << "\n";
  cout << "This is the value of x after " << x << "\n";
  return 0;
}
```

- The code snippets begins with initializing x living inside the heap memory with value 10.
- Before calling the function, in the first cout statement, we are asking for x, and we are asking for x outside of the function, so of course we will get x from the memory not of the function, i.e. the heap memory. The x in the heap memory is 10 ,so we get 10.
- The function takes an argument, and we have given the argument of the function to be x. Here the function will take x of the heap memory, and actually copies the value for the x in the heap memory, which is 10, to the x in the stack memory of the function, which will begin as 10. We again emphasize that x in the heap memory is not the same as the x in the stack memory. Then the x of the stack memory gets incremented by 20 and then returned, resulting in 30
- However, since x in the heap memory is different than x in the stack memory, when we try to get x, we actually get 10, because the function only affected the x in the stack memory and not the x in the heap memory, so the x in the heap memory is actually still 10.

4.6.2 Stack and Heap, linked

Of course, you can easily link the stack and heap memory x using a simple assignment operator. You can also use other ways to link them, but the assignment operator is the most simple and straightforward. A simple example is shown below.

```
#include <iostream>
using namespace std;

int functionname(int x){
  x = x + 20;
  return x;
};

int main()
{
  int x = 10;
  cout << "This is the value of x before " << x <<"\n";
  cout << "This is the return of the function ";
  cout << functionname(x) << "\n";
  cout << "This is the value of x after " << x << "\n";
  x = functioname(x);
  cout << "This is the value of x after assignment" << x <<"\n";
  return 0;
}
```

The output obtained without the text is 10, 30, 10, 30

Here through the line "x = functionname(x)", what you have actually told C++ to do

is to actually fetch the value of x from the stack memory and then copy the value of x from the stack memory, and use the value of x from the stack memory to overwrite the value of x in the heap memory, thus resulting in the x of the heap memory to be 30. Thus, in the next statement, where you tell C++ to fetch the value of x in the heap memory, C++ will fetch 30, i.e. the result of the overwrite.

Here, in essence, you have created a link between the value in the heap memory and the value in the scope memory.

4.6.3 Stack and Heap, Performance

We note that it is extremely expensive to do copying of data. While copying might be trivial for small amounts of data, copying large amounts of data is very expensive, both in time and the memory that it will take.

When you passing something from the heap memory to the stack of a function, you are actually making a copy of what you are passing from the heap memory and making that copy live in the stack memory. (Other than arrays, which auto decays into pointer when it is passed) Essentially, if I had let's say an 100 floats in the heap memory and I pass those 100 floats into the function, then I will actually need storage space for at least 200 floats of space, so 100 floats of space for the original in the heap memory, and 100 floats of space for what is inside the stack memory. Storing 200 floats is not a problem for computers nowadays, but you can see how **passing by value i.e. passing the values as they are** is not exactly memory efficient.

Thus, is there a way so that when we need to work on 100 floats, to only work on those 100 floats, and not have to make a copy?

Well, here comes pointers and arrays! Arrays in C++ are contiguous pieces of memory, so everything inside an array are stored next to each other. So the 1st element of the array is stored next to the 2nd element, and the 2nd element is between the 1st and 3rd element, and the 3rd element is between 2nd and 4th element etc. Thus, you dont't actually have to pass by value an array, i.e. copy the entire array from the heap memory to the stack memory, you could simply pass a pointer to the array. Remember, pointer simply points to a memory address which hold some sort of memory content.

You actually only need to pass a single pointer, and not a pointer for every element inside the array, i.e. if the array has n elements, I still only need 1 pointer and not n pointers. Because C++ simply needs to know where the pointer to the 1st element of the array is, and since C++ knows that the array is by definition contiguous, C++ knows that the thing right next to the memory address of the 1st element will hold the 2nd element, and from that, C++ knows that the neighbour of the 2nd element is the 1st and 3rd element, and since it already knows where the 1st element is, it can easily find where the 3rd element is, and thus where the 4th, the 5th, the 6th etc. element of the array is. Thus, C++ only needs to know where is the 1st element of the array, i.e. the pointer to the 1st element, and can simply use this pointer as a basepoint to find the 2nd element, 3rd element, 4th element etc.

Even if you have 2D arrays, or 3D arrays, or nD arrays, C++ can still use the pointer to the 1st elements, and use it as a base point to find the other elements. For a 2D array, C++ will have the memory location of element 00, and can easily use the size of element 00 to find element 01, 02 etc. C++ can also easily find the memory location of 10 by using the size of the element 0, i.e. the sum of the size of everything contained in element 0, so the elements 00, 01, 02 etc. However, **using 2D arrays is slower than using 1D arrays, since C++ has to do an extra summation step to find the memory location for element 1, element 2 etc.** Thus, if the array is **regular**, you can improve performance by collapsing the 2D array into a 1D array (column major or row major is ultimately problem dependent) can use as index $i * row + j$. However, this does make indexing for boundary conditions a bit more tricky and annoying. The same logic can be applied for 3D arrays, nD arrays etc.

This is why, when an array is passed into a function, the array is not passed by value, the array that is passed into the function actually decays by default into a pointer, pointing to the 1st element of the array. Thus, you do not actually copy the array when you pass the array into a function, unlike other variables like integers, strings etc. you actually only give C++ the pointer to the 1st element of the array. Using simple pointer arithmetric, and using the 1st element of the array as a base point, C++ can find the 2nd element, the 3rd element etc.

We note that C++ does not actuall go from 1 element to the next, and we will below how using the 1st element as a basepoint, C++ can easily get to the nth element in the array

Since the elements of the array are homogenous, i.e. they are by definition of the same type (NO heretical heterogenous arrays here!), C++ will know how many bytes of memory each element in an array will take. Thus, if let's say each element of an array takes 4 bytes of memory, thus taking 1st element of the array will correspond to the 1st to 4th bytes in the contiguous memory of the array, it should be obvious that the 2nd element will correspond to the 5th to 8th byte in the contiguous memory of the array, then it should be pretty obvious that let's say the 35th element it will take the 141th byte to the 144th byte in the contiguous memory of the array. Thus, C++ can, but does not have to go from 1st element, then find the neighbour to find the 2nd element, then the neighbour of the 2nd to find the 3rd element; in the example above, if C++ wants to find the 35th element, it will simply look for the 141th to the 144th byte of the contiguous memory of the array. Of course, different variabletypes require different number of bytes of memory, i.e. for example a variable can take 8 bytes of memory, 15 bytes of memory, or actually any sensible number of bytes, but the above logic still applies. i.e. let's say that every element takes 21 bytes, then the 35th element will be at the 736th byte to the 757th byte. It should be pretty trivial by now to work out from which byte to which byte will the nth element be stored.

We note that if you pass something's pointer or reference into a function, then you are passing into the function the thing's pointer or reference and making a copy of thing's pointer or reference, and thus you are **working on the original** and not a copy of that thing, since how can you work on a copy if you did not make copy of that thing? Remember, you made a copy of the pointer to that thing, and not tht thing itself.

5 C++ Intermediate

5.1 Header Files

Header files are files ending with .h that will contain stuff that you want to use repeatedly across multiple cpp files. The absolute minimum for a header file in C++ is an empty file, since you might simply not have anything that you want to use repeatedly across multiple cpp files.

However, header files typically look something like

```
#pragma once

int functionname(args here){
whatever code in the function
}

class classname{whatever is inside the class};
```

To include the header file in your cpp file, you simply have to write #include "filenameofheaderfile.h" A simple example is shown below

```
#include "filenameofheaderfile.h"
using namespace std;

int main()
{
  return 0;
}
```

Header files can include other header files, thus you could create a hierarchy of includes. However, you should be careful not to create circular includes, i.e. headerfile1 includes headerfile2 and headerfile2 includes headerfile1, because you will just make C++ run around in circles. An example of a nested header

```
#pragma once
#include "anotherheaderfilename.h"

int functionname(args here){
whatever code in the function
}

class classname{whatever is inside the class};
```

We note that there are several distinctions between the header file and the cpp file

- Header files contain no main function, this is because header files are designed to be included in another cpp file, and not designed to be compiled. Of course

you can give a header file to a compiler, but a compiler will not know what to do with it since there is no main. Even if you make a main in a header file, the compiler will still be confused why there is main inside a header file, because the compiler is good and telling you that a header file should not have main and should refuse to compile

- Header files typically include functions and classes
- Be very careful about circular includes
- Be very careful which header file you are including and whether you truly want **everything** in the header file to be available for your cpp file, or is there anything unwanted?
- #pragma once is a special line of code used to avoid the possibility of circular includes, but you should not try to rely on it and should think about your include hierarchy.
- you might see old header files that have instead of pragma once, 3 statements with ifndef, define and endif. These serve a similar purpose to the more modern pragma once, in that they are there to prevent circular includes from happening. An example is shown below.

```
#ifndef some_mumbo_jumbo
#define same_mumbo_jumbo

int functionname(args here){
whatever code in the function
}

class classname{whatever is inside the class};

#endif
```

Usually, header files are placed in the same folder as the cpp file that is using the header file, but it can also not be in the same folder as the cpp file. If the header file is one level above, then it can still be included using

```
#pragma once
#include "../anotherheaderfilename.h"

int functionname(args here){
whatever code in the function
}

class classname{whatever is inside the class};
```

5.2 C++ Boolean, Conditionals and Loops

Similar to Python, C++ have various operator that return a boolean value. These include <>==<=>=

Operator Usage	Description
$a == b$	returns true if a is the same as b
$a < b$	returns true if a is smaller than b
$a > b$	returns true if a is larger than b
$a <= b$	returns true if a is smaller or equal to b
$a >= b$	returns true if a is larger or equal to b
!	reverse true and false values of operators

C++ have conditionals using if, else if and else, an example syntax is shown below

```
#include <iostream>
using namespace std;

int main()
{
  int x = 10;

  if(x == 10) {cout << "x is equal to 10"<<"\n";}
  else if(x == 9) {cout << "x is equal to 9"<<"\n";}
  else{cout << "x is not equal to 10 or 9 " << "\n";}

  return 0;
}
```

C++ also have loops, such as for loops, while loops and also do while loops. **Careful not to create infinite loops!**

For loops in C++ require initiliazing an index, a condition to break out of the loop, and something to do at the end of each loop. An example of a for loop is shown below, where an index called i is declared and defined to be 0, the loop stops when i is no longer smaller than 10, and i gets incremented by 1 at the end of the each loop.

```
#include <iostream>
using namespace std;

int main()
{

  for(int i = 0; i < 10; i ++){
    cout << i << "\n";
  }

  return 0;
}
```

while loops in C++ are simpler, and simply need an ending condition. However, make sure to include inside the while loop something that will allow the while loops to reach the ending condition, otherwise the while loop loops forever. An example is shown below. This while loop does the same thing as the for loop above it. The

integer i is initialized outside the while loop, and then it is incremented by 1 at the end of the while loop, with the loop keep on looping as long as i is smaller than 10.

```
#include <iostream>
using namespace std;

int main()
{
  int i = 0;
  while(i < 10){
    cout << i << "\n";
    i = i + 1;
  }

  return 0;
}
```

Do while loops are not that popular, but essentially, unlike the while loops which first checks the conditions in the () before doing what is inside the loop, a do while loop will execute the loop once before checking the condition in () to see if it will continue looping. An example is shown below.

```
#include <iostream>
using namespace std;

int main()
{
  int i = 0;
  do{
    cout << i << "\n";
    i = i + 1;
  } while (i < 10);

  return 0;
}
```

5.3 String, String Methods and Char

```
#include <string>
```

Strings are a class in C++, specific instances of strings are objects and strings also have string class specific methods.

Method	Description
string stringname;	Declaring a string
string stringname = ”soemthing”;	Declare and Define string
string1 + string2	String Concatenation
string1 += string2	String Concatenation
string1.append(string2)	String Concatenation
string1.length()	Returns actual length of string
string1.size()	Returns actual length of string
string1.insert(index, string2)	Insert string2 at index
string1.erase(index, how many)	Erase that many characters starting from index
string1.replace(start, end, string2)	Replace from start to end with string2
string1.substr(start, length)	return a part of the string from start with length
string1.find_first_of(string2)	Find index where string2 occurs, else return npos

Comments

- Use ”” for strings since ” are for Char
- length() and size() actual return the number of characters and not the number of bytes
- npos is basically a large gibberish number due to trying to assign -1 to an unsigned thing.

Char

Char is for character array, which was the C way of storing a string, basically storing each character into an array. You might still see it in old code.

5.4 C++ Basic Math

Other than the typical arithmetic operators described, C++ supports various mathematical operations through the external library *cmath*. To use cmath, simply do

```
#include <cmath>
```

All of the functions below are going to be std::functionname if namespace std is not used, and functionname if we have using namespace std or some equivallent. For simplicity, assume that namespace std is used. Here I am calling these functions because they are coming from a library and not specific to a certain class. For cmath, you should use C++ classes that contain numbers.

Method	Description
sqrt(1 arg)	Returns sqaure root
remainder(1 arg)	Returns **float division** remainder
fmax(args)	Returns the largest number amongst the args
fmin(args)	Returns the smallest number amongst the args
ceil(1 arg)	Ceiling the number
floor(1 arg)	Flooring the number
trunc(1 arg)	Removes what is after the decimal
round(1 arg)	Rounds up or down

What is the difference between floor and trunc?

While floor and trunc might be essentially the same for positive numbers, they can produce different results for negative numbers. For example, if we have -1.5 as argument, then floor will return -2 since it rounds downwards, while trunc will return -1 since it removes everything after the decimal place.

5.5 C++ Constants

Sometimes in C++, you really want to restrict anyone from touching some numbers without proper thought or consideration, so you can set the number as a constant. To define and declare a constant in C++, you do

In the code below, I have declared a constant integer, and defined the value of the integer to be 5

```
const int x = 5;
```

What the keyword const does is that it will create a read only variable, which can only be read, but cannot be overwritten. Thus, while you can use x here for arithmetic operations, booleans etc, where you are only reading the value of x, you cannot use the assignment operator to reassign a new value to x.

We note that const is scoped. That means that if the constant is declared and defined without {} around it, then the constant is for the entire file. Otherwise, the const is only valid within the {} that it is contained within.

Another way to create constants, which is a legacy from C code is to do, though it is generally considered bad practice since it is not scoped, at the beginning of the file

```
#define variablename variablevalue
```

Below I have defined a constant with the name x and with the value of 5. We note that since this is defined at the the top of the file using #, this means that the constant is for the entire file.

```
#define x 5
```

We can also create a constant which can take only one of a possible list of values

```
enum{variablename = variablevalue1, variablevalue2, etc.};
```

Below, I ahve defined a constant with the name y, which can only have the values 100, 0 or 50

```
enum {y = 100, 0, 50};
```

We note that enum is actually a distinct type from the const type. The const type can only have a single value, while the enum can have a finite number of different possible values.

6 C++ Basic Data Structures

Containers are objects that contain other objects. Vectors, Arrays, Lists etc are all different types of containers in C++. **We note that while containers may share names between different programming language, i.e. Python has its own arrays and vector, and while they can share some similarities, you should be aware that there are also various differences. What works for a container in a programming language might not work for a container of the same name in another programming language**

6.1 C++ Vectors

Vectors only exist in C++ 11 and later

C++ vectors are containers that can contain objects. By definition, C++ vectors are contiguous container, meaning that it stores everything in a contiguous piece of memory. C++ vectors are also homogenous containers, meaning that it contains objects of the same class inside of it. It is possible to have heterogenous containers, but this is heretical and usually not recommended.

Vectors can store various objects, and since vectors are also objects, vectors can also store vectors. This allows you to create 2D vectors and 3D vectors or nD vectors.

Vectors are dynamically sized, meaning that you can insert and remove elements from a vector; however, do be careful that these operations can be very expensive. Vectors usually occupy more space than static arrays, because more memory is allocated to handle future growth. This way a vector does not need to reallocate each time an element is inserted, but only when the additional memory is exhausted. The total amount of allocated memory can be queried using capacity() function. This means that if you only have to do a few insertions or deletion, then vectors are alright, and will not significantly affect performance since extra memory is allocated for **some** future growth. However, vectors becomes very slow if you insert a lot or delete a lot of elements. Thus, it is generally recommended to already have sufficient space allocated to the vector so that you can avoid as many of the insertion and deletion operatons as possible.

6.1.1 Declaring and Defining Vectors

To use vectors in C++, you need to include the vector library

```
#include <vector>
```

Vectors can be declared in C++ as

```
vector<int> vectorname
```

An example of a vector with vectorname vect and with integers inside is shown below.

```
vector<int> vect;
```

However, this way of declaring the vector is problematic, since you have not specified a size for the vector, and C++ gets really confused about it. For example you cannot print the first element of this vector, because there is no 1st vector if the vector does not have a size. You can still insert elements into the vector, and once you insert, the vector will now have a size due to your insertion. You cannot however delete elements from empty vector since what there is to delete?

A much better way of initializing a vector is to also define a size for the vector during the declaration.

```
vector<element datatype> vector name(number of elements);
```

An example of a vector with vectorname vect and with 10 elements is shown below.

```
vector<int> vect(10);
```

We note that since we have not told C++ what the value for the elements are, C++ will just assign seemingly random values to them

```
vector<element datatype> vector name(number of elements, value of elements);
```

An example of a vector with vectorname vect and with 10 elements each initialized to 0 is shown below.

```
vector<int> vect(10, 0);
```

Vectors can also be declared by specifying each element

```
vector<element datatype> vector name{element1, element2 , etc.};
```

```
vector<int> vect{ 10, 20, 30 };
```

Since vectors are dynamically allocated,

You can also have nested vectors. Since vectors themselves are C++ object you can store vectors inside of vectors. Vectors inside of vectors do not have to be of the same size, and they can be different size.

6.1.2 Basic Vector Methods

Vectors are still classes in C++, thus vectors have their own class methods. Here, some of the most common vector class methods are introduced.

Method	Description
operator[]	Access element
at(index arg)	Access element with bound checking
empty(no arg)	returns true if vector is empty
size	returns the number of elements in array (NOT BYTES)
capacity	returns the maximum capacity before copy is called
shrink_to_fit	reduces memory usage by freeing unused memory
pushback(1 arg)	Adds element to the end

More methods on vectors can be found at https://en.cppreference.com/w/cpp/container/vector

6.2 C++ Arrays

C++ Arrays are container that contain objects. By definition, C++ arrays are contiguous containers, meaning that it stores everything in a contiguous piece of memory. C++ Arrays are also homogeous containers, meaning that it contains objects of the same class inside of it. It is possible to have heterogenous arrays, but this is heretical and usually not recommended.

Arrays can store various objects and since arrays are objects, arrays can also store arrays. This allows you to create 2D arrays and 3D arrays or nD arrays.

Arrays are statically allocated (different from dynamically allocated of the vector), meaning that once you have declared an array, it is no longer possible to resize the array, insert elements into the array, delete elements from the array. (Setting the element to 0 is not considered deleting because the element is still there, it is just 0). Thus, arrays in C++ are much more limiting in terms of the methods available compared to vectors. However, since you are unable to do those expensive operations specific to the vector, you are forced to avoid these expensive operations, and think of novel ways to deal with the problem using simpler array operations that are much less expensive and much faster.

Because arrays are statically allocated (unless they are dynamically allocated), you need to make sure that you have some idea of the maximum size of the array when creating the array. You also need to have an idea of how much memory does your computer have. You cannot allocated more memory to the array than what the computer have, so if your array will take 2Gb of memory and your computer memory is only 1Gb, then it is not physically possible to store the array on the computer. Typically, 1Gb of memory, i.e. 1073741824 bytes of memory can hold 268435456 4 byte integers. Of course, you can array shrinking algorithms or clever data manipulation to shrink the size of the array, but you are modifying the size of the array through the process, so the remark about the physical limitations still stand.

6.2.1 Declaring and Defining Arrays

Arrays in C++ are declared as

```
variabletype arrayptrname[number of elements];
```

Below, I have declared an array which holds inside it elements of variabletype int with the name arrayname and with the number of elements being 10

```
int arrayname[10];
```

Arrays in C++ can be defined and declared as:

```
element variabletype arrayptrname[] = {element 1, element 2 etc};
```

Below, I have declared an array which holds inside it elements of variabletype int with the 1st element being 10, the 2nd element being 20, the 3rd element being 30, the 4th element being 998. Here, although I am not specifying the number of elements for the array, C++ is smart enough to understand that the number of elements in the array is 4.

```
int arr[] = { 10, 20, 30, 998 }
```

Of course, you can also tell C++ the number of elements

```
element variabletype arrayptrname[number of elements] = {element 1, element 2 etc};
```

Below, I have declared an array which holds inside it elements of variabletype int with the 1st element being 10, the 2nd element being 20, the 3rd element being 30, the 4th element being 998. Here, although I am specifying the number of elements for the array, C++ is smart enough to understand that the number of elements in the array is what I have specified, and will fill the rest of the array elements that I have not specified with seemingly random gibberish.

```
int arr[10] = { 10, 20, 30, 998 }
```

We note that in the above, I have used arrayptrname instead of arrayname. This is because in the above way of declaring and defining, the name is not the name of the array, it's actually the name of the pointer to the 1st element of the array. To declare an array with the name of the array, and not the pointer to the 1st element of the array, we do

```
array<element_type, number of elements> arrayname = {element 1 etc.};
```

Below, I have declared an array which holds inside it elements of variabletype int with the number of elements of 4 with the 1st element being 1, the 2nd element being 3, the 3rd element being 5, the 4th element being 7.

```
std::array<int, 4> nums {1, 3, 5, 7};
```

We note that when we usually work with arrays in C++, we will work with the the arrayptrname instead of the arrayname because the whole reason you are using arrays is for performance, right?

6.2.2 Array Methods

Because of the limitations of being statically sized (we will talk about dynamically sized arrays later), arrays have far fewer methods than vectors. Some common array methods are shown below

Method	Description
operator[]	Access element
operator=	Assignment a value to the element
at(index arg)	Access element with bound checking
empty(no arg)	returns true if vector is empty
size	returns the number of elements in array (NOT BYTES)

These methods except the operator[] and operator = works only on the arrayname declared through the arrayname method above, and will not work on the arrayptrname!

Thus, to get the size of an arrray from arrayptrname, **assuming that every element of the array is of the same size**

```
int size = sizeof(arrayptrname)/sizeof(arrayptrname[index in bound]
```

```
int size = sizeof(arrayptrname)/sizeof(arrayptrname[0]
```

Note that this will not work if called in a function, because the pointer in the function are going to be decayed. A workaround is to do this before function and store the size in a variable, and pass the variable storing the size into the function

Note that this only gives the maximum total number of elements that the array can hold, and NOT the number of sensible elements. For example, if I have only defined 4 sensible elements in an array which can hold 10 elements, then the method above will give 10 and not 4

For arrays where elements are not of the same size, this becomes a lot more complicated to do the counting, and you will most likely need some extra information.

More methods on arrays can be found at https://en.cppreference.com/w/cpp/container/array

6.3 Vectors vs Arrays

While many people might preach that arrays are faster than vectors, this is in fact not the case. The reason why vectors give off the impression that it is slower than arrays is that people often do vector operations without being aware of the capacity of the vector, and thus the expense of such vector operations. While vectors are dynamically allocated and are designed so that you can insert or delete elements, these operations

are expensive operations, and doing expensive operations a lot of times makes your code slow.

Moreover, inserting elements beyond the capacity of a vector when there is no more free memory next to the last element of the vector to give to the new element inserted results in C++ copying the entire vector into a much more free location of memory, significantly slowing the vector operations down. Thus, even though vector give off the impression you can just add as many elements as you want to it, there is a caveat that if you exceed the capacity of the vector with the elements that you add and there is no more free space next to the last element to give to the newly inserted element, then the vector will do a copy operation which is super slow.

The reasoning vectors do a copy operation is that vectors are supposed to be contiguous containers. While growing the vector, you are growing it contiguously, i.e. as a single piece of memory. However, you also have other programs in your computer taking up memory, and if you grow the memory needed by a vector, the contiguous piece of memory of the vector will eventually border the memory allocated to another program. Now, C++ can infringe and steal a piece of memory away from the program, which can cause the program to catastrophically fail, or it can copy the whole vector into a new freeer piece of memory, thus doing an expensive copy operation but preventing the program from crashing. C++ chose to do the later since it is the polite thing to do and C++ is a good person who does not infringe on other people's property. You can imagine the confusion or anger you feel when you are watching a stupid cat video when your C++ program crashes your cat video and forces you out of your procrastination.

In fact, vectors are nothing more than a wrapper around an array, and was developed because people were annoyed with the array. Thus, if you do the same operations on a vector or on an array, then there should be no significant performance difference between vector and array. However, if you are going to use the methods for the vector that are not present for the array, you must be aware about the performance of these operations and the size of memory used by the vector and the amount of memory on your computer to avoid the expensive copy the whole vector to a freeer piece of memory operation. **You cannot enjoy the benefit of vector operations without some sacrifices to performance! You are going to make great sacrifices to performance if C++ is going to doexpensive copy the whole vector to a freeer piece of memory operation.**

6.4 Other Ways to store data

There are many other common data structures like queues, lists and stacks that come with their own methods, advantages and disadvantges. We can classify containers generally into 3 types **sequence containers, associative containers, and unordered associative containers** Exactly which type your are going to use is problem dependent.

We have already introduced the concept of arrays and vectors, which are the most commonly used containers. Vectors and arrays are sequential containers because you can access the elements of vector and array using a sequential index. Sequence

containers implement data structures which can be accessed sequentially, i.e. the 1st element, 2nd element, 3rd element etc.

Other than sequential containers, there are also **associative containers**. These elements are not sequentially stored, but they are stored associatively, i.e. there is not 1st element, the 2nd element, the 3rd element etc. You can think of associative containers more like a Python dictionary.

Unordered associative containers implement unsorted (hashed) data structures that can be quickly searched (O(1) amortized, O(n) worst-case complexity, i.e. how quickly it can be searched). Basically, it's just a bunch of completely disorganized stuff together.

More info on the different containers can be found at https://en.cppreference.com/w/cpp/container

7 C++ Struct

Before diving into Struct and Class, it is very important to be aware of the difference between the Struct and Class. The most important difference between Struct and Class are

- By default, everything in Class is private, while everything in Struct is public
- Class allows templetization while Struct does not

Thus, if your object can be simply described and you do not care about the security of the data inside of your object and you are sure you are not going to use templetization, then you can use either struct or class; but otherwise, use class.

7.1 Struct and OOP

A struct is a type which has members inside of it and has sequence allocation. Basically, struct is sth like a custom user defined object type. You can use struct to describe some object that are not preincluded within C++. A struct also has sequence allocation, which we will see when we construct an object from the struct.

Say that I have an object, let's say a phone, and you try to guess what is that object, and you can ask me questions about the object. Then it is likely that the most common question that you will ask to know what phone it is? You might ask well, what is the width, height and thickness of the phone? What is the brand of the phone? When was the phone released? What is the processor for the phone etc.

Say that there is a person, and you try to guess who that person is, and you can ask me questions about the person. Then it's likely that the most common question that you will ask to know who is that person? Well, you might ask the age, the affiliation, the city the person was born etc.

Thus, we can see that for a general class of objects, such as a phone, a person etc, there are number of characteristics(i.e. members) that define how one object is different from another. We don't neccesary need all the possible members to know what object it is, just enough member that we can distinguish it from the others.

7.2 Defining and Declaring Struct

The simplest possible struct in C++ has syntax

```
struct structname{no members}
```

Below is an example of the simplest possible struct in C++

```
struct structname{};
```

However, while this struct has no compile problems, it is also not very useful, because the struct simply has a name and no members. So yes, I am telling you that this is a struct(i.e. a phone), but can you guess what it is if I tell you nothing about it? Nope.

A better struct in C++ has some members in it. Here I will use a 3D point to demonstrate a struct.

When declaring and defining a struct, and deciding what are the arguments and what are the member functions, you need to think about what are the traits of the struct that make each object different from one another? For a 3D Point, it will be the x, y, z coordinates of the point. After,you ask, what arguments would there be for the members? For a 3D Point, it will be the x, y and z coordinate, thus

```
struct Point3D{

    int x;
    int y;
    int z;

};
```

Once we have the struct, we can create objects from the struct, or sometimes called instances of the struct. **We note that you cannot create an object from the struct if the struct has not been defined AND declared before.**

```
struct Point3D p1{56, 2, 3};
```

Here I have created an object p1 which is of the type Point3D, which has as its x-coordinate 56, y-coordinate 2 and z-coordinate 3. Here, we see sequence allocaiton, so the 1st member x has 56, the 2nd member y has 2 and the third member z has 3.

To access them, I can simply do, assuming using std::cout;

```
struct Point3D p1{56, 2, 3};
cout << p1.x << "\t" << p1.y <<"\t" << p1.z;
```

You can also create pointers to struct, an example of creating a struct pointer and how to access what is inside such a struct pointer is shown below. Struct pointers are useful if the struct has lots of members and storing the whole struct in memory will be inefficient, but **DO NOT** use pointers for your struct if there are very few memebers in the struct. You are making your code annoyingly complex for only marginal performance improvements.

```
struct Point3D p1{1, 2, 3};
struct Point3D* p2 = &p1;
cout << p2 -> x << "\t" << p2 -> y ;
```

We will see more on pointer and references later, but here the code just demonstrates how we can have a pointer to a struct.

8 C++ Class

Classes are a fundamental concept for C++ and any OOP language. For some conceptual understanding of classes, see the C++ OOP section. Here, we will see how can we construct a C++ class.

Here I am assming that the needed libraries, such as iostream for cin, cout and string for string are already there, i.e. you have all the neccesary libraries

8.1 Simplest Possible Class

The simplest code for a C++ class will have the structure

```
class classname{};
```

Below, I have created a class with name of phone with no members in the class

```
class phone{};
```

8.2 Class with Members

Similar to our discussion with struct, we understand that while the class phone shown above will compile succesfully, there are no members or methods inside of the phone class. Thus, all phones are essentially the same. We do not really want that, we want to have various different phones based on the needs and wants of people. Thus, we can add some members to the phone. To know what members to add, always think about what are the traits or characteristics of this class that you think will be important for your code. What traits or characteristics of your phone matter is a question left at your discretion. If you are buying a phone based on looks, then you might care about the width, length, height, color, so the width, height, length, color, would be the members of your phone class. If you are interested in photography and vlogs, then maybe you are interested in the camera specs of your phone, so these camera specs will be the members of your phone. Thus, what are the members for your class ultimately depends on what are the traits that you will be interested in for the objects created from this class.

Personally, I care about the brand of the phone, the price of the phone, and if the phone is minimalist, i.e. has no weird preinstalled apps that cannot be deleted. Thus I know that my phone will have 3 members. Before adding the members to the phone, I need to understand what are the datatype for these memebers.

For the member "price", I will probably use an int to store the price; you don't neccesarily have to use int, you can also use float, double etc. as long as the datatype is suitable to store the "price". For the member "brand", I could probably use a string, char array, or even an enum if I want to limit it to just some brands. Here I will use a string for the member "brand". For the member minimalist, I will use

bool to store a true or false value. If true, then the phone is minimalist and does not come with preinstalled weird apps, and vice versa.

Below, I have updated my phone class based on the previous paragraph.

```
class phone{

    std::string brand;
    int price;
    bool minimalist;

};
```

8.3 public, private, protected

Now that the class phone has been created with some members, it is important to start understanding what does public, private and protected mean for classes in C++.

Unlike struct, Classes in C++ by default has everything under private

The public, private and protected keyword are access specifier. Access specifiers define how the members of a class can be accessed.

These access specifiers are defined as

- public - members are accessible from **outside the class**
- private - members cannot be accessed (or viewed) from **outside the class**
- protected - members cannot be accessed from **outside the class**, however, they can be accessed in inherited classes. You will learn more about Inheritance later.

For now, we will be focusing on the public and private class. Let's look at our example with the phone class. This is the code for our current phone class.

```
class phone{

    std::string brand;
    int price;
    bool minimalist;

};
```

Since no access specifier is used, and since members in classes are by default private, then the code above is **equivalent** to

```
class phone{

    private:
    std::string brand;
```

```
    int price;
    bool minimalist;

};
```

Since the members of the class are private, by definition, these cannot be accessed from outside the class. Thus, if I do

```
class phone{
    private:
    std::string brand;
    int price;
    bool minimalist;

};

int main(){

    phone phone1;
    std::cout << phone1.brand;

    return 0;
}
```

Here I have created an object from the class called phone1 and I am attempting to access the brand member of the phone1 object. However, because it is private, I cannot access the brand member, because the code I use to access thr brand member, so

```
std::cout << phone1.brand;
```

is outside the scope of the class, i.e. outside the {} of the class phone. My compiler will give me an error saying that it is inaccesible.

Thus, if I want to access it from outside the class, I will need to change it from private to public, so

```
class phone{
    public:
    std::string brand;
    int price;
    bool minimalist;

};

int main(){

    phone phone1;
    std::cout << phone1.brand;
```

```
    return 0;
}
```

This will no longer give a compiler error, and it will compiler succesfully. However, we note that there will be nothing from

```
std::cout << phone1.brand;
```

because I have not actually specified what is the brand of phone. I have just created an object phone1 from the class phone, but I have not specified what is the brand of phone1 and C++ cannot simply just guess what is the brand. However, if you try to access the price, so if you instead do

```
std::cout << phone1.price;
```

then C++ will just give you some seemingly random value for the price. This is again because I have not specified the value for the price of the phone, so C++ just assigns some random value to it. For the boolean minimalist, C++ defaults all booleans to true unless otherwise defined.

You do not have to make everything public, or everything private, like demonstrated above, you can make some members public, while other members private. This idea can also be extended to the protected access specifier and also to class methods which will be introduced later. The example below shows a modified phone class where the brand is public, but price and minimalist is private.

```
class phone{
    public:
    std::string brand;

    private:
    int price;
    bool minimalist;

};

int main(){

    phone phone1;
    std::cout << phone1.brand;

    return 0;
}
```

The code above will compile because I am trying to access the member brand, which is public, meaning I can access the member brand from outside the class. However, if I instead try to do

```
std::cout << phone1.price;
```

then it will not compile because I am trying to access the member price, which is private, meaning that I cannot access the member price from outside the class.

An important note to make is that **while the access specifier affects whether members of the class can be accessed from outside the class, they are not controlling whether the member of the class is accessed from inside of the class.** For example, I can create a new public member for the class phone which holds the result of multiplication between the member price and the member minimalist, which I can give the arbitrary name "worth". This member will give me the price if the phone is a minimalist phone and will give 0 if the price is not minimalist. Essentially what I am saying is that a non minimalist phone is worthless. (just a personal opinion)

```
class phone{
    public:
    std::string brand;
    int worth = price * minimalist;

    private:
    int price;
    bool minimalist;

};

int main(){

    phone phone1;
    std::cout << phone1.worth;

    return 0;
}
```

The code above will compile succesfully because you are accessing the member worth, which is a public member, meaning that the member can be accessed from outside the class. Even though to find worth, you actually have to do the multiplication of price and minimalist, so you have to access the private members price and minimalist members, you still can find worth because the multiplication of the members price and minimalist are inside the {} of the class, i.e. in the scope of the class phone, so you are accessing the members needed for the multiplication **from inside the class, and not from the outside. Remember, private only stops access from outside the class and not inside the class; when you are finding worth, you are accessing the members price and minimalist from inside the class, not outside, so private cannot stop you!**

8.4 Class Methods

Similar to how strings, which are themselves a class, have their string methods, user defined classes in C++ can have their own methods, which are snippets of code most likely to be used on the objects of the class. (Static members are different from class

method, and do not act on the object of the class)

Class Methods can do various things. Class methods can be used to view the members inside of the class. Class methods can be used to modify the value of the members inside of the class, etc.

The simplest method for a C++ class

We continue from our example of phone

```
class phone{

    public:
    std::string brand;
    int worth = price * minimalist;

    private:
    int price;
    bool minimalist;

};

int main(){

    phone phone1;
    std::cout << phone1.worth;

    return 0;
}
```

A method has a similar syntax to a C++ function. It is ok to have a method defined and declared inside a class. It is also ok to have the method declared in the class and defined elsewhere. However, you are not allowed to have a method defined in another method, since C++ does not support nested methods, just like how C++ does not support nested functions. However, you can call a method inside another method.

The simplest syntax for a class method in C++ is

```
return_variabletype methodname(args){what code to do};
```

We note that

- If you do not want the method to return anything, then you should use as return_datatype void. If you use void, then you don't need a return statement. void is useful when you are for example viewing the members of the objects and even when you are doing modifications to the members of the objects. void is not useful when you wish to assign some info extracted from the object to some variable, since because there is no return, there can be no assignment,

i.e. what are you doing when you are using an assignment operator? You are taking the value generated from the right of the assignment operator and using it to overwrite the value of the variable to the left. However, if your RHS of assignment operator is void, i.e. nothing, C++ has nothing to use for the overwriting.

- You can **almost** give any name to the methodname, except of course the C++ keywords like main. It is generally considered a bad practice to use any C++ keyword inside the name of the method. You should also note that the name of the class is a specially reserved name for a type of method called constructors, and the ñame of class is a specially reserved for a type of method called destrcutors. Unless you are wanting to create a constructor or a destructor, do not name the method using the same name as the class.

Here I will be making the simplest possible method in C++. This method does literally nothing, but the code will compile succesfully. We note that we call the method on the object created from the class and not on the class itself (unless it's a static method, more on that later). Thus, here in main, I have created an object phone1 from the class phone, and then I have called the method inflateprice on the object phone1 of the class phone.

```
class phone{

    public:
    std::string brand;
    int worth = price * minimalist;

    void inflateprice(){}


    private:
    int price;
    bool minimalist;

};

int main(){

    phone phone1;
    phone1.inflateprice((();

    return 0;
}
```

If I execute the code above, although it will compiler succesfully, it will not do anything. This is because my specprint method is actually empty, i.e. it does not return anything since its return datatype is void, it take in no arguments since () is empty and it does not have any code inside of it since {} is empty.

To make the method more useful, I can add arguments to the method and to add some code to the method. I can also change the return type from void to int, so that

it actually returns sth, and I will need a return statement for that. For example, an argument that I could add would be the percentage that the price is inflated by, so a float. And the code will compute the the new inflated price and overwrite the old price with the new inflated price.

```
class phone{

    public:
    std::string brand;
    int worth = price * minimalist;

    void inflateprice(float inflate_amount){

    price = price * (1. + inflate_amount);

    }


    private:
    int price;
    bool minimalist;

};

int main(){

    phone phone1;
    phone1.inflateprice(0.1);

    return 0;
}
```

If I try to find the price now, I understand that since I have only created the object phone1 and not given the object a price, C++ just assigns some seemingly random number for the price. Calling the inflateprice method here with the float argument 0.1 will increase the price of the object phone1 (**NOT THE CLASS phone**) by 110

A class method only needs to be declared in the class. To call a class method, the method must be defined, but the method can be defined either inside the class or outside the class. However, note that you cannot define a method inside of another method; C++ does not support nested methods. When defining the method outside the class, we make sure that we are also indicating to C++ which class does the method belongs to. An example is shown below. A class method can even be defined elsewhere inside of a header file, noting that the header file containing the definition for the method must be included to run the method.

For example, for the inflateprice method, I can simply declare the method in the class, and define the method outside of the class.

```
class phone{
```

```
    public:
    std::string brand;
    int worth = price * minimalist;

    void inflateprice(float inflate_amount); // DECLARE

    private:
    int price;
    bool minimalist;

};

//DEFINE
void phone::inflateprice(float inflate_amount){
    price = price * (1. + inflate_amount);
}

int main(){

    phone phone1;
    phone1.inflateprice(0.1);

    return 0;
}
```

Here I have declared in the class that there is a method called inflateprice. We note that for C++, it distinguishes methods from one another by using not only the name of the method, but also the arugments in the method, which might be different for other programming langauges. This feature of C++ allows for method overloading, but more on that later. The method is then defined outside of the class; to link the method defined outside of the class to the method declaration inside the class, we will add classname:: before the methodname.

8.5 Setting up members and Defaults

Setting up members

Currently, inside our class phone, we have only created the members, and we have not told C++ what exactly are the members, i.e. if we create our object phone1 from the class phone, and we try to access the price, brand, minimalist etc, C++ will return a seemingly random number for price, nothing for brand, and true for minimalist.

Thus, our phone1 cannot really represent a phone now. However, I could assign things to the phone1 to really make it into a phone. An example is shown below where I tell C++ that the brand for the phone1 is Nokia. You could change the string to anything other than Nokia, like Samsung, Apple, Huawei, Xiaomi, Oppo etc. We note that strings in C++ are case sensitive, so "Nokia" is not the same as "nokia".

```
class phone{
```

```
    public:
    std::string brand;
    int worth = price * minimalist;

    void inflateprice(float inflate_amount);

    private:
    int price;
    bool minimalist;

};

void phone::inflateprice(float inflate_amount){
    price = price * (1. + inflate_amount);
}

int main(){

    phone phone1;
    phone1.brand = "Nokia";

        std::cout << phone1.brand;

    return 0;
}
```

Now if I execute the code, I will do what is inside of main. In the main, the first line will created an object phone1 of the class phone. Then the next line tells C++ that the brand member of the object phone1 is not empty and it's actually the string "Nokia". The next line then prints out the brand of phone1, which will output to console Nokia because I have told C++ that the brand of phone1 is Nokia in the second line.

Of course, you could also do something similar with the public member worth. We note here that the worth of the phone initiallly will be just some seemingly random number. If you do the change to the worth here, although we have stated that worth is the product of price and minimalist, the changes that you make to worth will not cause a corresponding change to price and minimalist. This is because the initial worth is calculated during the creation of the object phone1. Then if you change worth, C++ is not able to change the price and minimalist, (**NOT BECAUSE OF PRIVATE ACCESS MODIFIER**) since you are not telling C++ to change worth, price and minimalist; you are only telling C++ to change worth, and C++ will only change worth.

Another important point to note is that you cannot set up the values for private members of the class outside the {} i.e. scope of the class. This is because the private access modifier prevents you from even accessing the private members from outside the {} of the class. However, you can change the private modifiers from inside the class though since private only stops access from outside the class and not inside the

class!

Default Values

In the previous example, we have only defined the members inside the class. You can actually both define and declare the members inside a class. Thus, all objects created from the class will have such values for its members by default. You could of course later on change these default values later. I don't need to have default values for each member, I could have default values only for some members.

For example, I could say that the default for all phones are that their brand is Huawei and their price is 500, they are not minimalist.

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

class phone{

    public:
    std::string brand = "Huawei";
    int worth = price * minimalist;

    void inflateprice(float inflate_amount);

    private:
    int price = 500;
    bool minimalist = false;

};

void phone::inflateprice(float inflate_amount){
    price = price * (1. + inflate_amount);
}

int main(){

    phone phone1;
    std::cout << phone1.brand;
    phone1.brand = "Nokia";
    std::cout << phone1.brand;

    return 0;
}
```

Here the console output will be HuaweiNokia. Let's work thorugh the main. In the 1st line of main, we have created an object phone1 from the class phone. Then we have asked C++ to console out the brand of the phone1 object, and C++ tells us it's Huawei because Huawei is the default. In the 3rd line, I have set the brand of phone1 object to be Nokia. In the final line, I have again asked C++ to console out the brand of the phone1 object, and now, since I have set the brand of the phone1 object to be Nokia, C++ will tell me it's Nokia.

We note that if we have created a new object here, i.e. phone2, then the brand of phone2 takes the default to be Huawei. The 3rd line of the code above acts on the object phone1 object and not on the phone class, so the change of Huawei to Nokia in the 3rd line does not cause the default brand when creating an object of the class phone to change.

8.6 Static Members and Static Member Functions

Static Members

When we declare a member of a class as static it means no matter how many objects of the class are created, there is only one copy of the static member. A static member is shared by all objects of the class. All static data is initialized to zero when the first object is created, if no other initialization is present.

Static member can be initialized outside the class as done in the following example by redeclaring the static variable, using the scope resolution operator :: to identify which class it belongs to.

As an example, I will continue from the class phone that was given above, and add a static member called phonecount. For now, let's just see how to declare and define a static member.

The typical syntx for static member is

```
static variabletype staticmembername;
```

And to using the above syntax, I can define a static member phonecount inside of my phone class with variabletype int.

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

class phone{

    public:
    static int phonecount;
    std::string brand = "Huawei";
    int worth = price * minimalist;

    void inflateprice(float inflate_amount);

    private:
    int price = 500;
    bool minimalist = false;

};
```

```
void phone::inflateprice(float inflate_amount){
    price = price * (1. + inflate_amount);
}

int phone::phonecount = 0;

int main(){

    std::cout << phone::phonecount;

    return 0;
}
```

Here, we note that when using static members, we need to be very careful about the scope of the static member. Static members work differently from the usual members of a class. Static members are declared inside the class, but they need to be defined outside the class, and they cannot be initialized in the main because main is a function. Here the static member are defined outside the class because static members are shared by every object inside the class, so they cannot be defined in the class. Finally we can access the static member inside the main function. Moreover, when we are outside the class, we need the classname:: to be before the name of the static member, otherwise C++ does not know to which class does the static member belong to.

We also note that since static members are for the class and not the object, we do not actually need to create an object to use static methods on the class.

Static Methods

Just like static members, static methods are also independent of any object of the class. A static member function can be called even if no objects of the class exist and the static functions are accessed using only the class name and the scope resolution operator ::.

A static member function can only access static data member, other static member functions and any other functions from outside the class.

Static member functions have a class scope and you cannot use the this pointer on the static member function. (More on this later)

Static methods are declared and defined inside the class, and they can be called from outside the class if access to them is not blocked by access specifier.

Below I have created a dummystaticmethod that will return an integer and will return 10. In the main function, I have created a dummy variable to store the return of the dummystaticmethod and then console out the dummy variable, which consoles out 10.

```
class phone{
```

```
    public:
    static int phonecount;
    std::string brand = "Huawei";
    int worth = price * minimalist;

    static int dummystaticmethod(){
        return 10;
    };

    void inflateprice(float inflate_amount);

    private:
    int price = 500;
    bool minimalist = false;

};

void phone::inflateprice(float inflate_amount){
    price = price * (1. + inflate_amount);
}

int phone::phonecount = 0;

int main(){

    int dummyvariable = phone::dummystaticmethod();
    std::cout << dummyvariable;

    return 0;
}
```

We again note that since static methods are for the class and not the object, we do not actually need to create an object to use static methods on the class.

8.7 Constructor

We have seen how we can create an object from class in main, and then set the members of the class in main. We have also seen that we can give defaults to these values.

Constructors are special methods in C++ class. **The constrcutors must have the same name as the class name.** Constructors are methods which returns an object from the class. Since constructor are by definition returning an object of the class, constructors do not actually need return types, since its return type must be the class. Similar to other methods in C++, constructors need only be declared in the class. Constructors can be defined in the class, but they can also be defined outside the class, just not in other functions or members since C++ does not support nested functions. We note that for constrcutors defined in the main, the constructor only

needs the classname for its name, because since it is in the class, C++ understand that the constructor is a constructor for the class. However, if defined outside the class, then the constructor must also have a classname:: in front of it to tell to which class does the constructor belong to

The simplest constructor for C++ has syntax, noting that the constructor is the must have the same name as the class

```
constructorname(){};
```

In the below, I have implemented such a constructor for the phone class.

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

class phone{

    public:
    static int phonecount;
    std::string brand = "Huawei";
    int worth = price * minimalist;

    static int dummystaticmethod(){
        return 10;
    };

    void inflateprice(float inflate_amount);

    phone(){}; // Constructor

    private:
    int price = 500;
    bool minimalist = false;

};


void phone::inflateprice(float inflate_amount){
    price = price * (1. + inflate_amount);
}

int phone::phonecount = 0;

int main(){

    phone phone1 = phone();

    return 0;
}
```

I can also have the constructor declared in the class and defined elsewhere. I note that in the constructor declaration, I need to have (), while for the definition outside, I need to have () and {} and classname::

An example is shown belows with the class phone. Here the constructor has been declared in the phone class through the line

```
phone();
```

And the constructor has been defined outside the class through

```
phone::phone(){};
```

I note that for a constructor declared in the class and a constructor defined outside the class, the arguments in () of the constrcutor declared in the class and the constructor defined outside the class **MUST MATCH**

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

class phone{

    public:
    static int phonecount;
    std::string brand = "Huawei";
    int worth = price * minimalist;

    static int dummystaticmethod(){
        return 10;
    };

    void inflateprice(float inflate_amount);

    phone(); // Constructor Declared

    private:
    int price = 500;
    bool minimalist = false;

};

phone::phone(){}; // Constructor Defined

void phone::inflateprice(float inflate_amount){
    price = price * (1. + inflate_amount);
}
```

```
int phone::phonecount = 0;

int main(){

    phone phone1 = phone();

    return 0;
}
```

For now we have seen how to create a constructor, and how to define the constructor both inside the class and outside the class. You can think of constructor as a worker that buiilds an object from the materials given, i.e. the arguments. Our current constructor does not have any arguments, so it's like we have not given the worker any construction material, but in the world of C++, there are no worker's right or protection, so what will the worker do? The worker here does have the defaults for some members of the object, so the worker will use the defaults to build the object.

We can of course give the worker some construction material, i.e. arguments, and explicitly tell the worker to build the object from those construction materials. If we do not tell the worker explicitly to use the construction material given (i.e. arguments), the worker will still use the defaults to construct the object.

Below, I show an example, with the constructor definition and declaration inside the class, where the worker takes in some arguments to construct an object. We note that constructors only need to be declared inside the class, and can even be defined in another separate header file, provided that the header file is included.

```
class phone{

    public:
    static int phonecount;
    std::string brand = "Huawei";
    int worth = price * minimalist;

    static int dummystaticmethod(){
        return 10;
    };

    void inflateprice(float inflate_amount);

        // Declared and Defined Constructor
    phone(std::string input_brand, int input_price,
    bool input_minimalist){
        brand = input_brand;
        price = input_price;
        minimalist = input_minimalist;
        worth = price * minimalist;

    }

    private:
```

```
    int price = 500;
    bool minimalist = false;

};

void phone::inflateprice(float inflate_amount){
    price = price * (1. + inflate_amount);
}

int phone::phonecount = 0;

int main(){

    phone phone1 = phone("Samsung", 3000, false);

    return 0;
}
```

I can similarly do it outside of the class

```
class phone{

    public:
    static int phonecount;
    std::string brand = "Huawei";
    int worth = price * minimalist;

    static int dummystaticmethod(){
        return 10;
    };

    void inflateprice(float inflate_amount);

        // Declaring Constructor
    phone(std::string input_brand, int input_price,
     bool input_minimalist);

    private:
    int price = 500;
    bool minimalist = false;

};

// Defining Constructor
phone::phone(std::string input_brand, int input_price,
 bool input_minimalist){
        brand = input_brand;
        price = input_price;
        minimalist = input_minimalist;
        worth = price * minimalist;
```

```
}

void phone::inflateprice(float inflate_amount){
    price = price * (1. + inflate_amount);
}

int phone::phonecount = 0;

int main(){

    phone phone1 = phone("Samsung", 3000, false);

    return 0;
}
```

- We note here that here we have used an argument for each member of the class. So we have used the argument input_brand for the brand, we have used input_price for the price and we have used input_minimalist for the minimalist. You do not neccesarily have to use an argument for each member. You could no arguments whatsoever as shown above, or only arguments for some members. However, do note that the members that are not specified through the constructor **are going to take the defaults, and if no defaults, then seemingly random number for int, emptiness for string and true for bool etc.**
- It is usually considered a good idea to avoid confusion that even if there is a simple 1 to 1 relationship between the argument and the member, to have the argument be a different name from the member to avoid confusion in the code. Of course, some smarter people are very good and do not need such renaming to avoid confusion.
- We also note that for the member worth, I do not whether it is neccesary to recalculate it, but I will usually recalculate it just to be safe.
- We also note that list of arguments of the constructor declared in the class must match the list of arguments of the constructor defined outside the class and also match the arguments when we are calling the constructor in the main.

8.8 Destructor

While Constructors create the object, the destructor destroyes the object. **The destructor destructs the object. However, if the object contains as members pointers or references, the destructor will not also destroy the object that the member pointer or reference points to or references to, these must be destroyed separately if you want to free up the memory**

Destructors do not accept any arguments and Destructors does not do any return, since there is nothing to return as the whole purpose of the destructor is to destroy the object.

Similar to constructors, destructors must have the same name as the class; however, to distinguish the destructor from the constructor, the destructor has an additional prefix. Similar to constructors, the destructor can defined and declared inside the class, or it can be defined in the class and declared elsewhere.

While virtual destructors are a thing since C++20, it's not backward compatible and, well, we won't discuss this because it's way too new and not backward compatible.

We note that we cannot actually call the destructor on the object we want to destroy, we actually have to call the destructor on a pointer to the object to free up the memory.

The general syntax for the Destructor in C++ is

```
~classname(){}
```

Continuing from the code in the previous

```
class phone{

    public:
    static int phonecount;
    std::string brand = "Huawei";
    int worth = price * minimalist;

    static int dummystaticmethod(){
        return 10;
    };

    void inflateprice(float inflate_amount);

    phone(std::string input_brand, int input_price, bool input_minimal
        brand = input_brand;
        price = input_price;
        minimalist = input_minimalist;

    };

    ~phone(){};

    private:
    int price = 500;
    bool minimalist = false;

};

void phone::inflateprice(float inflate_amount){
    price = price * (1. + inflate_amount);
}
```

```
int phone::phonecount = 0;

int main(){

    phone * phonepointer;
    phonepointer = new phone("Samsung", 5000, false);

    return 0;
}
```

Here I have created a destructor inside of the class. In the main function, I have created a pointer to the object phonea and then used that pointer to point to an object of phone. I have then deleted that pointer, thus freeing the memory up.

While you cannot have arguments in the destructors, you could use C++ code such as std::cout etc. inside of the {} of the destructor.

Of course, this destructor can also be declared in the class and defined out of the class, as shown in the example below. We note that destuctors can also be only declared in the class and defined elsewhere inside a header file, provided that the header file is included.

```
class phone{

    public:
    static int phonecount;
    std::string brand = "Huawei";
    int worth = price * minimalist;

    static int dummystaticmethod(){
        return 10;
    };

    void inflateprice(float inflate_amount);

    phone(std::string input_brand, int input_price, bool input_minimal
        brand = input_brand;
        price = input_price;
        minimalist = input_minimalist;

    };

    ~phone();

    private:
    int price = 500;
    bool minimalist = false;

};
```

```
void phone::inflateprice(float inflate_amount){
    price = price * (1. + inflate_amount);
}

phone::~phone() {}

int phone::phonecount = 0;

int main(){

    phone * phonepointer;
    phonepointer = new phone("Samsung", 5000, false);
    delete phonepointer;

    return 0;
}
```

If you do not have a pointer, in the Pointers and References Section, you will see how you can get the memory address of an object, create an classofobject pointer, point it to the object, and then call the destructor on the newly created pointer, thus destroying the object.

8.9 Getter, Setter, Encapsulation

Getter and Setters are just ways we can classify the member functions of a class in C++. **Getters are used to get some info from the object of the class**, for example some info about the member of the class. **Setters are used to set the value for members of the class.** Getters and Setters are not special methods like the constructor or the destructor, they are just how we can classify the different methods of a class.

For example, the simplest getter method could simply just get a member of an object and console it out.

```
class phone{

    public:
    static int phonecount;
    std::string brand = "Huawei";
    int worth = price * minimalist;

    static int dummystaticmethod(){
        return 10;
    };

    void inflateprice(float inflate_amount);
```

```
    phone(std::string input_brand, int input_price,
    bool input_minimalist){
        brand = input_brand;
        price = input_price;
        minimalist = input_minimalist;

    };

    ~phone(){};

    //Getter
    void getprice(){
        std::cout << price;
    }

    private:
    int price = 500;
    bool minimalist = false;

};

void phone::inflateprice(float inflate_amount){
    price = price * (1. + inflate_amount);
}

int phone::phonecount = 0;

int main(){

    phone phone1 = phone("Apple", 5000, false);
    phone1.getprice();

    return 0;
}
```

Here I have created a method called getprice which returns nothing, and console out the price of the phone. The method getprice is public. The method getprice is a getter because it gets the price of the phone. I can call the method because the method is public, and the method can get the price, which is a private member, because the method is inside the class, and private does not control access from inside the class.

getprice is obviously not a setter because getprice cannot modify the price, or any other member.

getprice is used to ensure that while you can view the price, you cannot actually modify the price, since getprice simply console out the price. This is very useful when it comes to protecting stuff in objects that you really do not want to be tampered with, but you still want to view. For example, the user's credit card number. You want the user to be able to view their credit card number (maybe after some checks here) but you don't want the user randomly changing their credit card number.

Of course, there are ways to hack and break this protection, but since I am an ethical and good person with no personal experiences in such acts, there will be no info here.

Let's then see an example for setter. Here I have created a method called changeprice, which takes in as argument a new price and changes the price of the object, returning nothing. This method is a setter not because its name is changeprice, (i.e. I could name it notchangeprice and it would still be a setter, though I will be branded a heretic). What decides that it is a setter is the method's ability to change the price of the object, and not the objectname. Similarly, what decides that a method is a getter is the method's ability to pull some info about the object, and not the objectname.

```
class phone{

    public:
    static int phonecount;
    std::string brand = "Huawei";
    int worth = price * minimalist;

    static int dummystaticmethod(){
        return 10;
    };

    void inflateprice(float inflate_amount);

    phone(std::string input_brand, int input_price, bool input_minimal
        brand = input_brand;
        price = input_price;
        minimalist = input_minimalist;

    };

    ~phone(){};

    void getprice(){
        std::cout << price;
    }

        // Setter
    void changeprice(int price_new){
        if(price_new > 0){price = price_new;}
        else{}
    }

    private:
    int price = 500;
    bool minimalist = false;

};

void phone::inflateprice(float inflate_amount){
```

```
    price = price * (1. + inflate_amount);
}

int phone::phonecount = 0;

int main(){

    phone phone1 = phone("Apple", 5000, false);
    phone1.getprice();
    phone1.changeprice(10000);
    phone1.getprice();

    return 0;
}
```

In the main function, the first line calls the constructor and construct an object phone1 of the class phone with the brand Apple, the price 5000 and not being minimalist. The 2nd line calls the getprice method that console out the price of the object phone1, which will be 5000. The 3rd line then call the changeprice method that will check first check if the input is larger than 0. Since 10000 is larger than 0, the method will change the price of the object phone1 to 10000. The 4th line then calls the getprice method again that console out the price of the object phone1 to be 10000, the new value of the price I have set it to.

While the setter could have simply worked with me overwriting the price with the value of the new_price, I have included here an if else statement. The reason to include if else statement is to prevent some arbitrary change to the value of price, and force the new price to only be positive. For example, if it was a phone store, then I cannot possibly set the price of the phone to be negative, i.e. I don't get money but actually debt for every phone I sell. Here, I am talking about a simple phone store, and not some infamous financial institution which can tries some magikal, morally questionally restructuring. Here, we see a reason why to use setter. If I was to use public here, while changes to the price are possible, I cannot control how does the price change, i.e. someone could hack my system and make all my prices negative, so that I lose money everytime I sell a phone. However, if I instead use private for the member price, and used a setter method for changing the price, I can control how the price can be changed. Of course, this method of protection is not perfect and impervious to hacking, but this is just a simple example to show the usefulness of getters and setters.

Of course, you can hack and break such protection too.

Getters and Setters are an essential part of what is Encapsulation. **Encapsulation refers to the bundling of data with the methods that operate on that data, or the restricting of direct access to some of an object's components.** Here, we have encapsulation because we have limited the access to the member price of the object, and we have bundled the method getprice and changeprice to the price of the object. For example, here we have shown how we can encapsulate the member price of the class phone.

If you think of setting the price private means that you have made a big space capsule surrounding the the inside of the space capsule, and nothing goes in, nothing goes out. Then you have created the getter method getprice, i.e. a window on your space capsule, which allows people to view the capsule, but not change the space capsule. (No, we are not going to discuss blasting Gamma rays into the space capsule and thus altering change the atomic structure) Then you have created a setter method changeprice, which allows people to change the space capsule, but only make valid changes upon the capsule thorugh the if else that filters out people with nefarious intentions.

8.10 Inheritance and Polymorphism

So far, we have discussed the class in C++. In life, we often encounter objects that are child classes of a parent object. For example, the cabbage, broccoli, cauliflower, kale, Brussels sprouts, collard greens, Savoy cabbage, kohlrabi, and gai lan are actually all subclasses of Brassica oleracea. These different vegetables share same traits and characteristics, but do not share other traits and characteristics. While you can give a class to each one of these vegetables, it is much better if you can create a class call Brassica Oleracea which contains all the common traits and characteristics of all these vegetables, and actually make a child class for each of these vegetables to contain their unique traits and characteristics. We note that you are not restricted to parent and child, you can have grandparent, parent, child and grangrandparent, grandparent, parent, child etc. Also, please have sensible family hierarchy trees.

For example, some member for the parent class Brasscia Oleracea can be the number of leaves, the height of the plant etc. Some member function for the parent class could be givewater, givesunlight etc. For the child class cauliflower, unique member could be yellowness, size of flower, and unique member function could be cookcauliflower. I would expect that you are not a heretic that tries to cook the cabbage, broccoli, cauliflower, kale, Brussels sprouts, collard greens, Savoy cabbage, kohlrabi, and gai lan all in the same way.

Here, I will use BO to represent Brassica Oleracea and CF for Cauliflower to save me some spelling mistakes. We have already seen the general syntax for creating a class, so we need to understand now the general syntax for creating a subclass.

```
class nameofchildclass : public parentclass{};
```

An example code where I have a parent class OB and the child class CF. I have defined some members and some methods for both the parent and child class. For simplicity, I have made all members and methods of the child class to be public. For the parent class, the member leaves_numb and the method givesun and givewater are public, member height is protected and member pesticide is private.

```
class OB{

    public:
```

```
    int leaves_numb = 100;

    void givesun(){
        std::cout << "sun received" << "\n";
    }
    void givewater(){
        std::cout << "water received" << "\n";
    }

    protected:
    int height = 10;

    private:
    bool pesticide = true;

};

class CF : public OB{

    public:
    bool yellowness = true;
    int sizeofflower = 10;

    void cook_CF(){
        std::cout << "CF ready" << "\n";
    }

    int heightroof = height + 20;

};

int main(){

    CF plant1;
    std::cout << plant1.leaves_numb <<"\n";
    plant1.givesun();
    std::cout << plant1.sizeofflower << "\n";
    plant1.cook_CF();
    std::cout << plant1.heightroof << "\n";

    return 0;
}
```

In the 1st line of the main, I have created an object with name plant1 of the class CF. We know that CF is a child class of the parent class OB.

In the 2nd line, I have asked C++ to give me the number of leaves of plant1. Since CF is a child class of the parent class OB, and since CF will inherit the public members and methods of the class OB, CF will inherit the public member leaves_numb from the class OB, so the leaves_numb of CF will be 100, the default in the OB.

In the 3rd line, I have asked C++ to call the method givesun on the object plant1. Since CF is a child class of the parent class OB, and since CF will inherit the public members and methods of the class OB, CF will inherit the public method givesun from the class OB, so when I call the method givesun on the CF object plant1, it will console out the statement "sun received".

In the 4th and 5th line, I have called the unique member of the child class CF and called the unique method of the child class CF, so C++ does it, giving output and the output "CF ready".

Finally in the 5th line, I have called the heightroof member of the child class. If we look at declaration and definition of heightroof, it depends on the member height. Height is declared and defined in the parent class OB under access specifier protected.

Now, we have seen that public members of the parent class are accessible for the child class. Private members of the parent class are not accesible for the child class. Moreover, the child class is inheriting the public members and methods from the parent class and not the private members.

The class member declared as Protected are inaccessible outside the class but they can be accessed by any child class of that class.

Thus we see that while we will get an error when we try to access the private member pesticide for an object of the parent class, since pesticide is private. We **CANNOT** use the member pesticide if we are inside the scope of the child class, i.e. inside the {} of the child class because it is a private member of the class OB, and private blocks access from the class, i.e. here will be OB.

We will also get an error if we try to access the protected member height for an object of either the child class or parent class. However, unlike private, we **CAN** use the member height if we are inside the scope of the child class. Thus, we see that I can use height to find out the public member of the child class heightroof. When I ask C++ to give me the heightofroof of the object plant1 of class CF, C++ will tell me it's 30. heightroof is a public member of the child class CF, so my call in the main can access it. Then to find the heightroof, I will need member height. But now since I am inside of the child class, so I can access the member height of the parent class OB.

Here, we see an extremely simple example of polymorphism. We see that while object plant1 is a CF, it is also an OB, thus the plant can have the members and the methods of CF, but it can also have the members and the methods of CF that it is allowed to inherit.

Polymorphism is the provision of a single interface to entities of different types or the use of a single name to represent multiple different types. We currently have "a single name to represent multiple different type" so the object "plant1" is both of the class CF and the class OB. We will see "the provision of a single interface to entitities of different types" as we introduce virtual methods in the next subsection.

9 Reading and Writing Files in C++

You can read and write files in C++. Reading and writing files require the fstream library. While the iostream is for the input output stream to the console, fstream is for the file streaming. Before we read or write to the the file, we must first create some variable to represent (NOT STORE) the file that we are working on, and we also need to tell C++ what are we going to do to the file. The following keywords tell C++ what are we going to do to the file.

- *ofstream* Writes to the file
- *ifstream* Read from the file
- *fstream* Read and Write from the file

While fstream might seem convenient, it is much better to use ofstream or ifstream if you are sure that you are only going to read, or only going to write to the file. This is because some files might be set to read only, so you cannot use ofstream on the file.

9.1 Variable to Represent File

Before we open a file, you have to declare some variable in order to represent (not store) the file for any subsequent operations. The general syntax, assuming that the fstream and any other library, headers etc has been imported properly is

```
#include <iostream>
#include <fstream>

int main(){

std::typeoffilestream variablenametostorefile;
return 0;
}
```

Below, I have included an example where I tell C++ that I am going to read the file, and also created a variable that will represent the file in subsequent operations.

```
#include <iostream>
#include <fstream>

int main(){
```

```
        std::ifstream file1;
        return 0;
}
```

Here, I have created a variable called file1 that will represent the file that I will be working on. I have used ifstream to tell C++ that I am going to read from the file1.

Below, I have included an example where I tell C++ that I am going to write the file, and also created a variable that will represent the file in subsequent operations.

```
#include <iostream>
#include <fstream>

int main(){

        std::ofstream file1;
        return 0;
}
```

Here, I have created a variable called file1 that will represent the file that I will be working on. I have used ofstream to tell C++ that I am going to write to the file1.

By now, we have created a variable to represent the file. If we are reading the file, we need to tell C++ what is the filename of the file, and we have to link this filename with the variable that represents the file. If we are writing the file, we need to tell C++ what is the filename of the file that C++ will write to.

In C++, text files are read and written differently than binary files.

9.2 Opening the File

After we have created a variable to represent the file we are working on subsequent operations, we need to open the file so that we can work on the file. When you are doing read, the file must already exist; otherwise C++ does not know where to read the data from. When you are doing write, the file can already exist, or, if the file does not exist, C++ is understanding enough to create a new file for you with the filename you specify

The general syntax for operning a file in C++ is

```
variablenametostorefile.open("filename");
```

For example, if I wanted to open a file with the filename filename1.txt, and I want to use the variable file1 to represent the file with the filename filename1.txt in my subsequent operations, I would do

```
file1.open("filename1.txt");
```

We note that if you only give the filename to C++, then C++ will search for the file with that filename at the location where the cpp is located. It will not search for the filename in the subfolders or level above.

To open the file which is in a subfolder, you will do, being aware that C++ might fail because you might not have access to the subfolder, you will write

```
file1.open("foldername\\filename1.txt");
```

To open the file which is in a level above, having the same awareness of permissions as before, you will write

```
file1.open("..\\filename1.txt");
```

CAUTION!!!

The directory separating character is different depending on different operating systems, and identifying the directory separating character for different systems is actually not trivial, so you might have to replace the code above with the directory separator for your system. Thus, this becomes a problem if you want cross platform compatibility for your C++ code.

- If you are using C++17, and you are sure that no one using the code will try to compile the code using an older compiler, then you could use the following to try to get the line separator.

  ```
  std::experimental::filesystem::path::preferred_separator
  ```

  However, we also note that this feature is experimental and might not work properly in all cases.

- Another approach is to use some container like Docker.

- You could also try to define the character based on the operating system; however, you need to check your compiler specifications to make sure the compiler accepts. For example, Cygwin does not accept _WIN32

  The following snippet could be placed at the top of the code, and will create a read only constant called PathSeparator, which defines the path separator for a windows system. You can always add statements to tell C++ what are the respective separator for other operating systems. You can also do #error if you encounter some operating system you are not expecting.

  ```
  const char PathSeparator =
  #ifdef _WIN32
                                '\\';
  #endif
  ```

- Or you could use a 3rd party library like BOOST

9.3 Flags for Opening the File

When we open the file in C++, we can actually also put some extra info when we open the file, to tell C++ more specifically exactly how do we want to open the file. We refer to these extra info as flags. We note that the word flag is a rather generic word, and might mean something different outside of the context. The typical flags (i.e. here we mean extra info given to C++ when opening the file) include:

- *ios::binary* Open in binary mode and allows reading and writing of binary
- *ios::ate* Set the initial position at the end of the file.If this flag is not set, the initial position is the beginning of the file.
- *ios::app* All output operations are performed at the end of the file, appending the content to the current content of the file. This is neccesary if you are appending to the file
- *ios::trunc* If the file is opened for output operations and it already existed, its previous content is deleted and replaced by the new one.
- *ios::in* Open for input operations.
- *ios::out* Open for output operations.

We note that there are some flags that come by default with their respective type of file stream

- ofstream ios::out
- ifstream ios::in
- fstream ios::in | ios::out

We note that we can have multiple different flags, but please don't put conflicting flags or C++ might have weird and unexpected behaviour. The general syntax for flags in fstream is

```
variablenametostorefile.open("filename", flag1 | flag2 ...);
```

Here, we see that the filename is separated from the flags by a comma, while the flags are separated between each other by a vertical line, which is an OR operator in C++. A simple code example where I open the code in read mode, binary code, and at the end of line mode is

```
file1.open("filename1.txt", ios::in | ios::binary | ios::ate);
```

Another example where I open the file in write mode with append mode

```
file1.open("filename1.txt", ios::out | ios::app);
```

9.4 Check file is Opened

Unlike other programming languages that will throw an exception when the file is not opened, C++ does not throw an exception, so it is GOOD practice in C++ that you check if the file has been succesfully openened or not. A simple way to check it is included in the methods of the class of variables you have used to create the file.

The general syntax is

```
variablenametostorefile.is_open()
```

This method will return true if the file being linked to the variable on which the method is acting has been opened. It only checks that variable on which it is acting. The method will return if the file being linked to the variable on which the method is acting has not been opened.

A typical example is shown below, where we only execute the code if we know that the file has been opened succesfully

```
if(file1.is_open()){

// Some code here
}
```

9.5 Reading Text Files

You can read textfiles in C++ (Reading and writing binary files will be convered later). The typical way to read textfiles is to use the getline() method on the variable storing your file. Because you read your file line by line, you will need a loop to loop through the lines of the file. You will also need to create a string to store each line you have looped through in the string

The general syntax is as follows:

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

int main(){

    std::ifstream variabletostorefile;
    file1.open(filename);
    std::string linename;
    if(variabletostorefile.is_open()){
        while(std::getline(variabletostorefile, linename)){
            std::cout << linename << "\n";
```

```
        }
    }

    else{std::cout << "File failed to open \n";}

    return 0;
    }
```

An example is given here,

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

int main(){

    std::ifstream file1;
    file1.open("file1.txt");
    std::string line;
    if(file1.is_open()){
        while(std::getline(file1, line)){
            std::cout << line << "\n";
        }
    }

    else{std::cout << "File failed to open \n";}

    return 0;
    }
```

In the example, we have first created a variable called file1 which we will later use to represent to our file. Then we have linked fileone with a filen in the same folder as file1 with the name file1.txt We continued bby creating a string, which will be used to store each line that we are going to loop through. We have an if else statement to check that we have opened the file, and once we have finished checking that the file has been opened, we begin a while loop. The while loop will then get every line of the variable representing the file, so file1, and each line of the file will be stored inside the string called line. We then print the string line out at every loop. Here we should expect to print out every line within the file.

Here we are having the string line represent all the characters inside a line in the file. However, many times when we work with files, files are delimited using characters such as comma, backspace etc. We can easily tell C++ what our deliminting character is by simple including the delimiting character inside of ' (**NOT ""** Remeber, ' is for characters in C++, "" is for strings in C++, we are doing delimiting character or characters **NOT** delimiting string). An example is shown below for commas as delimiting character

```
while(std::getline(file1, line, ','))
```

There are many other methods to parse lines, include using the stringstream library or the BOOST library or simply doing

```
line >> variable1 >> variable2;
```

Which will parse the line based on BACKSPACE only.

We note that we must use a string for the purpose of parsing, i.e. getting the characters from the line. However, the data we have might be completely made out of numbers. Thus following getting them as strings, we should convert the data to int or float or their approapriate type, taking into account what are safe and unsafe conversion.

The string methods to convert strings to int or float are

```
std::stoi(string1); \\ Convert string1 to integer
std::stof(string2); \\ Convert string2 to float
```

We can of course use containers like arrays or vectors to store the data we have get from the file. The best practice for storing the data in arrays is to allocate sufficient memory to the vector or array, create an integer variable to count on which loop of the while loop you might be, and use this integer variable to update the element at the correct index.

A simple example is given below, where we are storing each line inside an array

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

float array1[10];

int main(){

    std::ifstream file1;
    file1.open("file1.txt");
    std::string line;
    int count = 0;


    if(file1.is_open()){
        while(std::getline(file1, line)){
            array1[count] = std::stof(line);
            count ++;
        }
    }
```

```
else{std::cout << "File failed to open \n";}

for(int i = 0; i < 10; i++){
    std::cout << array1[i] << "\t";
}

return 0;
}
```

Here I have created an array of floats of size 10. We have first created a variable called file1 which we will later use to represent our file. Then we have linked file1 with a file in the same folder as file1 with the name file1.txt We continued bby creating a string, which will be used to store each line that we are going to loop through. We have also created an integer called count to count on which loop we are currently on. We have an if else statement to check that we have opened the file, and once we have finished checking that the file has been opened, we begin a while loop. The while loop will then get every line of the variable representing the file, so file1, and each line of the file will be stored inside the string called line. We then take each line and then we called stof, which converts the string to the float. We then assign the element of array1 at the index count to be the result of converting the line to string, i.e. the 1st line will be the 1st element of the array, the 2nd line will be the 2nd element of the array etc. Finally after the loop, we do a for loop to print out all elemnets in the array.

9.6 Writing to a File

Writing to a file in C++ is similar to how we do cout. The general syntax for writing something to a file is

```
variabletostorefile << some string of characters;
```

However, just like how in cout we need a line separator to separate the console out into different lines, the escape characters for the cout are valid also for writing into the file.

Of course you could have multiple different write statements, for example

```
file1 << "This is something to be written on 1 line \n";
file1 << "This is something to be written on another line \n";
```

You can of course also put the write statements into a loop, just **MAKE SURE** there is a proper termination condition for the loop and you are not doing infinite loops.

9.7 Read and Write to Binary Files

For binary files, reading and writing data with the extraction and insertion operators ($<<$ *and* $>>$) and functions like getline is not efficient, since we do not need to format any data and data is likely not formatted in lines. The general syntax for binary files is

```
variabletostorefile.write ( memory_block, size );
variabletostorefile.read ( memory_block, size );
```

Where memory_block is the address (i.e. we pass a pointer to char here) of an array of bytes where the read data elements are stored or from where the data elements to be written are taken. The size parameter is an integer value that specifies the number of characters to be read or written from/to the memory block.

9.8 Returning back to the top of the file

We note that when we do the while getline loop, even if we break the loop, we are stuck at the last line in the file for the loop. To return back to the top of the file, we must do

```
file1.seekg (0, ios::beg);
```

9.9 Closing the File

An important thing I have not done for all the code about file streaming above is closing the file. While the file is automatically closed by C++ when you exit the progrma, or when we go out of scope of the {} encapsulating the .open method, we can manually close our file. This is very useful when you might be working on very large file, and closing the file can help free up the memory. We note that you should only close the file when you are sure that you are not going to use the file for a long time, or when you are sure that you are no longer going to use the file, because opening the file is actually quite an expensive operation, so while 1 opening might be neccesary to start working on the file, we want to have as few opening files as possible.

The general syntax for closing a file is simple

```
variabletostorefile.close();
```

In code, if I were to close a file which is represented by the variable file1, I will do

```
file1.close();
```

To reopen the file after closing it, if you think it is absolutely neccesary to do so, then just do

```
file1.close();
file1.open("filename.txt");
```

We note that while we have closed the file, we have not called any destructor on the variable file1 so even if we closed the file, file1 will still exists.

It is neccesary if we are opening the same file again to specify to file1 again the filename of the file. We note that file1 does not have to represent the same file. Remember, when we do open, we are simply saying that this variable that can represent file (i.e. file1) will represent the file with this specified filename(i.e. filename.txt). Thus when you use open method on file1 after you have closed it, you can actually point file1 to another file from the previous file that file1 has pointed towards. However, do make sure that you document the reassignment of file1 if there is reassignment to avoid confusion.

10 C++ Advanced

10.1 Intro to Pointers and References

10.1.1 What is a Pointer?

A Pointer in C++ is simply a variable that holds the memory address of of another variable. Different variabletypes in C++ will each have their own different pointers.

Because different variables in C++ use different amounts of memory, i.e. int usually uses 4 bytes, bool use 1 byte etc., thus, you need to have different types of pointers based on what is the variable that the pointer is supposed to point to, i.e. an integer pointer, so a pointer pointing to an integer variable, must know that it is actually pointing to the 4 bytes in the memory storing the integer and a boolean pointer must know that it is actually pointing to the 1 byte storing the boolean. If a pointer did not have a type, how would the pointer know how many bytes that it is pointing to?

We say that the pointer holds the memory address, and at the memory address that the pointer store, we can find the memory content containing the variable that the pointer is pointing towards.

10.1.2 Declaring a Pointer

First, let's begin through the general syntax for creating a pointer to C++.

```
variabletype_of_what_is_pointed_to * pointername;
```

For example, using the above syntax, I can declare an integer pointer with the name int_ptr

```
int * int_ptr;
```

If I have a boolean, I can also easily declare a boolean pointer with the name bool_ptr

```
bool * bool_ptr;
```

We note that I have just named the integer pointer to be int_ptr and the bool pointer to be bool_ptr. You could give any name to the pointer you want, so the name of the pointer is not what tells C++ what type of pointer it is.

When we declared the pointer, the first word tells about what is the variabletype that the pointer will be pointing towards. So if your pointer is supposed to hold the memory address of a integer, you would use int for the first word, and if your pointer is supposed to hold the memory address of a boolean, you would use bool for the

first word. You could also use your custom class name or struct name if the pointer is supposed to hold the memory address of your custom class or custom struct.

The next symbol * tells C++ that we are not declaring the variable, we are declaring a pointer to the variable. The symbol * **HERE** tells C++ that we are declaring a pointer to the variable. And finally, you give a name to the pointer and you're done.

We note that for int and bool, where the number of bytes is fixed, C++ knows how many bytes in memory it has to look up. However, for strings, where the number of bytes in memory depends on its length, the symbol * tells C++ to use some way to find the number bytes of memory that the string is taking. This is why in old C language, you did not have strings, but char, where each char is exactly 1 byte in size. If you needed to store a word, for example "cat", you would need to store each character inside a char array, so your char array will be sth like { 'c', 'a', 't' }. We know that arrays in C and C++ are fixed size, so C++ can understand just simply read the number of bytes for the array to figure out how many bytes in memory will it have to look up. For the word cat, there will be 3 characters in the char array, leading to the array having a fixed size of 3, so C++ reads this and understands it has to look up 3 bytes of memory, where the 1st byte is c, the 2nd byte is a, and the 3rd byte is t.

10.1.3 Declaring and Defining a Pointer

Currently, we have seen how we can declare a pointer. However, while this compiles succesfully, it's not really useful since the pointer does not point anywhere. To make a pointer point to some variable, we **CANNOT** just use the assignment operator directly to connect the pointer and the variable. Remember, pointers are supposed to hold the memory address of the variable and not the memory content of the variable. Thus we must first extract the memory address of the variable, and then use the assignment operator to assign the extracted memory address to our pointer.

The general syntax for declaring and defining a pointer to a variable is

```
variabletype_of_what_is_pointed_to * pointername = & variablename;
```

As an example

```
int x;
int * int_ptr = &x;
```

Here I have declared an integer x. I have not defined the value of x so C++ will just get some seemingly random number in there for the value of x. In the next line, I have declared an integer pointer with the name int_ptr to the LHS of the assignment operator. To the RHS, I have used the symbol & in front of the variable. **HERE**, the symbol & in front of the variable is used to get the memory address of the variable for the RHS. Thus, on the LHS of the assingment operator, I have an integer pointer that is supposed to hold the memory address for an integer, and on the RHS of the

assignment operator, I have the memory address of the variable x. Now, following this assignment, the integer pointer int_ptr will hold the memory address (**NOT MEMORY CONTENT**) of the variable x.

10.1.4 Getting Memory Content through Pointers

Ok, now that we have a pointer, well, how can we get the data at the memory address that the pointer is pointing towards? The general syntax for doing so is

```
* pointername;
```

And a code example continuing from our previous example

```
int x;
int * int_ptr = &x;
std::cout << *int_ptr;
```

So the 3rd line is a console out, and it is a console out of the * and the integer pointer. Here, we see that there is no variabletype preceding *, thus int this situation. * here alone actually means to get the memory content at the memory address the pointer holds. So here, the memory content at the memory address the pointer holds is what seemingly random value was give to x by C++. Thus it would console out that seemingly random value given to x.

10.1.5 What is a Reference?

Reference variable is an alias, that is, another name for an already existing variable. Once a reference is initialized with a variable, either the variable name or the reference name may be used to refer to the variable.

10.1.6 What are the Differences bettween Pointers and Reference?

Because Pointers and References are often talked together, and sometimes confused, it is important to understand the difference between Pointers and References.

- You cannot have NULL references. You must always be able to assume that a reference is connected to a legitimate piece of storage.
- Once a reference is initialized to an object, it cannot be changed to refer to another object. Pointers can be pointed to another object at any time.
- A reference must be defined when it is declared. Pointers can be only declared at any time.

10.1.7 Declaring and Defining a Reference

The general syntax for declaring and defining a reference is:

```
variabletype_of_what_is_referenced_to * referencename = & variablename;
```

```
int x;
int & x_reference = x;
std::cout << x_reference;
```

Here, I have declared x as an integer, and since I have not defined it, C++ will just assign some seemingly random value to x. In the 2nd line I have created a reference to x. The int & tells C++ that it is an integer reference and then I have assigned that reference to represent x. In the final line, I can console out x_reference. Since x_reference is simply an alias for x, the console will show the value of x. We note that since x_reference is simply an alias for x, any changes happening to x_reference will also change x in the same way, and any changes happening to x will also change x_reference in the same way. Essentially, currently, the reference is something like a shallow copy of the variable. It should be obvious that it is not neccesary to have the name x_reference for the reference of x; you can give your reference any name.

10.1.8 Avoid Confusion Symbols * and &

Here, we see that the symbols * and & are clearly very important for both Pointers and References. However, their use above probably confuses you a bit. To help you to understand what do * and & mean depending on the situation, I have created a simple C++ program with if else statements.

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

int main(){

    char character;
    bool front_variabletype;

    std::cout << "Do you have * or &?";
    std::cin >> character;

    if(character == '*'){
        std::cout << "Is there a variabletype in front of it? \n";
        std::cin >> front_variabletype;
        if(front_variabletype){
            std::cout << "You are declaring a Pointer \n";
        }
```

```
        else{
            std::cout << "You are getting the memory content \n";
        }
    }
    else if(character == '&'){
        std::cout << "Is there a variabletype in front of it? \n";
        std::cin >> front_variabletype;
        if(front_variabletype){
            std::cout << "You are declaring a Reference \n";
        }
        else{
            std::cout << "You are getting the memory address \n";
        }

    }
    else{std::cout << "Invalid Input \n";}

    return 0;
}
```

10.2 Multi D Data Structures

10.2.1 Multi D Array

The general syntx for a multi D Array in C++ is

```
variabletype  arrayname[size1][size2]....[sizeN];
```

An example

```
int arrayname [3][3][3];
```

Here I have created a 3x3x3 array containing integers with the name arrayname

To create an array where all the elements are the same, we can do

```
variabletype  arrayname[size1][size2]....[sizeN] = {value_to_be_filled};
```

An example

```
int arrayname [3][3][3] = {20};
```

Here I have created a 3x3x3 array containing integers with the name arrayname and with all values being 20.

10.2.2 Multi D Vectors

We can also have vectors of vectors in C++. The general syntax for a vector of vectors is

```
vector<vector<int> > vectorname;
```

An example

```
vector<vector<int> > vectorname;
```

Here I have created a vector of the vectorname, and inside the vectorname, each element of the vectorname is also a vector, and the vector inside has inside of it variabletype int. You can of course replace int with others such as bool, float etc, and even with your own class or struct, provided that the neccesary libraries, headers, declaration and definitions are alright.

I can also create a vector which are filled with a certain value, the general syntax is written as

```
vector<vector<int>> vectorname( n , vector<int> (m, 0));
```

And an example

```
vector<vector<int>> vetornamec( 20 , vector<int> (50, 60));
```

Here I have created a vector with the name vectorname. $vector < vector < int >>$ tells me that the vectorname vector will contain vectors inside that will contain the variabletype int. The $(20, vector < int > (50, 60))$ tells me that there will be 20 elements inside of vectornmae, and each element inside of vectorname will be a vector, which will hold the variabletype int and have 50 ints in each, with each int initalized to be 60.

If you have a nested vector containing custom class, and you want to fill it with an object from the custom class or struct, I would suggest creating the object you want to use to fill the nested vector beforehand, and then just use the example shown above, changing the variabletype from int to the classname or structname of your custom class or struct, and then changing the value 60 above with the name of the object you have created beforehand.

We note that while nested data structures might be nice, they are not as efficient as unnested data structures. You can of course use unnested data structures to represent 2d or 3rd or nd arrays using the indexing i*rows + j or i*rows + j*columns + k, but you do need some extra thinking to correctly implemented boundary conditions. Unnested data structures are faster since **C++ do not have to calculate how many bytes till next row or next column each time it passes through a row or column**

10.3 Dynamic Memory Allocation

10.3.1 For Variables

Now that we have gone through pointers and nested data structure, we can start with another powerful feature of C++ for HPC, so the Dynamic Memory Allocation.

Dynamically allocated memory is means allocating some memory on the Heap memory for something. For example, I can allocate some new memory on the heap for my data structure, I can also delete the allocated memory on the heap for my data structure. I have thus taken full control on whether to allocate or deallocate memory, and thus taken full repsonsibility too for anything that might happen due to improper memory allocation. If programmer doesnt deallocate memory, it causes **memory leak, meaning that memory is not deallocated until program terminates.**

C++ has two operators **new** and **delete** that perform the task of allocating and freeing the memory.

The general syntax for using the **new** operator is

```
int * pointername = new variabletype;
```

```
int * pointername = new int;
```

Here I have alllocated memory for a single integer, so 4 bytes, and I have assigned the pointer of the name pointername to hold the memory address for the new block of memory I have allocated to the int.

To free up the memory allocated, I will use the pointer pointing to the memory location of the allocated memory, and do

```
delete pointername;
```

Of course, we can also dynamically allocate memory for other classes than int, and also for custom struct and classes.

10.3.2 For Arrays

There is a difference between declaring and defining an array and allocating a block of memory using new. The memory of arrays which are declared and defined is controlled by the compiler, and the compiler should deallocate the memory for the array once the array is no longer useful.

However, dynamically allocated arrays always remain there until either they are deallocated by programmer or the program terminates, thus even if the array stops to be ever needed after some stage in your program's execution, the array will remain there and hog up memory which could be freed to do something else. This is because such arrays are created by you through dynamic memory allocation, and C++ respects

your sensibility and trusts that you won't be negligent in your duty of deleting the array once it is no longer useful.

If we want to allocated a block of memory for an array, we can simply do

```
int * pointername = new variabletype[numberofelements];
```

```
int * pointername = new int[10];
```

Here I have alllocated memory for a 10 integer in contiguous memory, so 40 bytes, and I have assigned the pointer of the name pointername to hold the memory address for the **FIRST** block of memory of 4 bytes in the 40 bytes I have allocated to the int.

To free up the memory allocated, I will use the pointer pointing to the memory location of the allocated memory, and do

```
delete[] pointername;
```

Here I **MUST** have [] in my delete operation. This is because my pointer points to the **FIRST** block of memory of 4 bytes in the 40 bytes, so if I do delete on the pointer, I only free up 4 bytes of the 40 bytes by using the delete on the pointer itself. You will have deleted the pointer and 4 bytes in the memory but still have 36 bytes of memory shown as allocated, that will not be deallocated until the program ends, thus resulting in a memory leak. You can of course call delete on the remaining 36 bytes; however, you have already deleted the pointer, so you actually don't know where the rest of the 36 bytes are in memory! [] tells C++ that you want to free up the entire array, so the entirety of the 40 bytes.

We also need to be very careful when dealing with pointers inside of the array, a.k.a an array of pointers, because when you are deallocating memory, you are deallocating memory for the pointers in the array, and not actually what the pointers are pointing towards.

Essentially, you can think of an array of pointers as the reference section in some journal article. When you call delete on the array, even using [], you are destroying the references in the references section, and not destroying what the reference is pointing towards.

If your array contains pointers, and you wish to deallocate also the memory allocated for the objects that the pointer inside the array point towards, you must first loop through the array and then call delete on each and every pointer inside the array, and **AFTER** you have finished your loop, then call delete to delete the array.

10.4 Pointer to Object with no Allocation

You can actually use new on a pointer to an object of a some class. Here, you will be allocating memory for the pointer, but you are not actually allocating memory for the object of some class the pointer is supposed to point towards. Remeber, pointers

are simply holders for memory addresses and they can be reassigned. This can be very useful when your data is already in memory somewhere from some previous operation, and for some reason you need for a new pointer to point at the data.

The general syntax for this is

```
classname * pointername;
```

And an example

```
class classname{};
classname * pointername;
delete pointername;
```

Here I have created a class with the name classname. Then I have created a pointer to the classname with the name pointername. Here, I have created a pointer which can point to an object from the class classname, but I have not actually created an object of the class classname. Thus, I have only allocated memory for the pointer that I have created, and I have not actually allocated memory an object of the class classname because I ahve not created an object of the name classname. The pointer created is currently pointing to nowhere, since I have not told C++ where is the pointer pointing towards; however, I can easily tell such things to the pointer, since pointers in C++ can be reassigned.

I can create a pointer and tell it what is the variabletype that the pointer is supposed to point to, but I DO NOT have to but I can create an object of the variabletype that the pointer is supposed to point to, and then point the pointer towards that object

And for a final, very important thing, that's why it is both in bold and red

We note that since in Dynamic Memory Allocation, the allocation of memory using new and the deallocation of memory using delete is your responsibility, so when you allocate a memory, remember to deallocate the memory at the first instance you are sure that you are never going to use that ever again.

WHEN YOU CALL new YOU ALWAYS CALL delete

10.5 Pointers and Linear Algebra

You should generally avoid pointers when you are just having simple snippets of code, i.e. if you just have a single simple object, then there is no point actually creating a pointer to the simple object because you are just complicating yourself. However, if you have something that is very large, like a very large matrix or vectors, then pointer and references are actually really really useful.

When you work with very large array or vectors, if you just have a single simple operation on the matrix or vector, you should just do that simple operation in your main function. However, if you are doing multiple different operations on the very

large matrix or vectors, you are likely to group each of these operations into a function, just to make code generally neater and more organized.

However, when you are passing object into a function in C++, what C++ actually does is to make a copy of the object, so you are using 2X the memory. This way, the object passed into the function is actually different from the object outside the function, meaning that operations done inside the function are not done on the object outside the function, unless of course you use an assignment operator and get the object outisde assigned the value from the return of the object. For more details on the concept, please go through C++ Memory Manipulation: Pointers and References.

Well, after going through C++ Memory Manipulation, you should see that passing a vector or an array or any large object or data structure into a function through pointers and references is far more memory efficient. While we understand that we are making changes on the heap memory and not the scope memory of the function, and since you have a single copy, you will be working on the original. Thus it is best recommended to have the initial object before passing it into function saved into some file. Your mass storage should have a lot more storage capacity than your RAM, else you should really pay for the extra mass storage. Since you are having a single copy of the object at all times, you will have to maybe call a writing function to write the generation desired to a file, so that you have a copy for plotting and debugging in the future, i.e. a simple if statement that if the generation is some generation, then write the entire object, array, vector, data structure into some file.

Thus, usually in numerical computation, for memory intensive workloads where you will use more than 1 Mb of memory, you would allocate memory on the heap, and then use a pointer to store the memory location of the first block of memory that you have allocated. Since arrays and vectors are by definition contiguous memory pieces, if you know the memory address of the first block of memory you can easily work out the address of the rest of the blocks of memory.

To allocated memory on the heap for an array, the general syntax will be

```
variabletype pointername = new variabletype[size of array];
```

And for an example

```
int * pointername;
pointername = new int[10];
```

Here I have used int * which tells C++ that I am declaring an integer pointer with the name pointername. In the next line, I am using this integer pointer to a newly allocated piece of memory, holding the variabletype of int, which is able to hold 10 integers.

I have of course do the above in a single line And for an example

```
int * pointername = new int[10];
```

We note that we can only point an integer pointer towards an integer or an integer array, we cannot say point an integer pointer to a string array

or a float or any other variabletype. This is because C++ gets confused. When you declare that it is an integer pointer, C++ expects that you are going to point the pointer to an integer or an integer array; however, if you point it to anything other than an integer, C++ gets confused since it is supposed to point to an integer, but it is not actually pointing to an integer. This could be extended to other variabletypes too; for example, C++ expects that if you have a boolean pointer to point to a boolean or a boolean array, if you have a string pointer to point to a string or a string array etc. You can of course point your integer pointer to another integer, that is not a problem for C++ since it is not violating its expectations that an integer pointer is supposed to point to an integer.

For vectors, you have sth slightly similar

```
vector<variabletype> * pointername;
pointername = vector<variabletype>();
```

And for a code example

```
std::vector<int> *pointername;
pointername = new std::vector<int>();
```

Here I have created a pointer with the name pointername, and it is a pointer to a vector that contains integer inside of it. I can of course do this in a single line too. In the brackets following the vector, I can specify stuff, just like I have shown in the C++ Basic Data Structures. You can even have a nested vector in there, it would not be a problem!

```
std::vector<int> *pointername = new std::vector<int>();
```

We note that in our 2 line version, we can totally just do the 1st line and C++ will accept it, since C++ is ok with you creating a pointer to sth, and not actually pointing the pointer to sth yet. You should; however, later on point the pointer to sth otherwise what would be the raison d'etre for the pointer?

We note that while the above code is correct, it is generally not a good practice to initialize a vector with empty brackets, instead you should try to initialize the vector with some size, preferably the maximum possible size needed to hold whatever data you have. The reasoning is that vector specific operations that grow or shrink the vector such as push back are really expensive. Moreover, here, if you do not tell C++ how much memory you allocate for your vector, C++ will simply allocate no memory whatsoever. If you try to access or set an element in the vector without growing the vector beforehand, C++ gets confused since the vector has no memory allocated to it, and you are trying to ask me for some element of a vector that has no memory???

10.6 Accesing Data Structures with Pointers

To access the elements inside an array/vector, and even work on them, we need to understand that pointername here is not the array, it is simply a pointer to the array.

Since you want to work with the array elements, you will have to tell C++ to look at the memory content at the memory address pointer points at. For arrays, there are many different ways to accesss the elements within the array

Since you can access them, you should also be able to work with them, and also do reassignment or any accepted mathematical operation with them. For nested vectors or arrays, you should understand by now that the elements of these nested vector or arrays are themselved vectors or arrrays. Thus, you can simply use the normal ways to access the vector or arrays inside the nested vectors or arrays.

I have included below examples of pointers to nested vectors or arrays. We note that if you are using nested vector or arrays, then the object that your pointer will point towards will be a nested vector or neste array respectively and the reference to the pointer will also be a reference to a nested vector or array respectively.

For an array

```
int rows = 10;
int cols = 10;
int **pointername = new int*[rows];
for(int i = 0; i < rows; ++i) {
    pointername[i] = new int[cols];
}

std::cout << pointername[5][5] << "\n";
```

For a vector

```
std::vector<std::vector<int>> *pointername;
pointername = new std::vector<std::vector<int>>
(10, std::vector<int>(10,0));

pointername->at(5).at(5);
pointername->operator[](5)[5];

std::vector<std::vector<int>> &referencename = *pointername;
referencename[5][5];
```

Now let's think about your **motivations**. The whole reason why you are using pointers and references and complicating yourself so much is because you want performance. However, 1D arrays/vectors are far faster than 2D arrays/vectors. The difference in performance between 1D arrays/vectors and 2D arrays/vector increase with the size of the array/vector. So using 2D arrays/vectors with pointer and reference is **complicated, difficult to do, prone to bugs, and generally have terrible performance, so why would you use 2D arrays/vectors** when maybe you can give some extra thinking about indices to save you the time spent debugging?

Of course, we remind ourselves that **WHEN YOU CALL new YOU HAVE TO CALL delete**. For both the nested array/vector, you have to loop through them, and then delete each vector/array from the bottom up.

10.7 arrow pointer

By now, you would have seen me sometimes use an arrow, so the $->$. This is known as an arrow operator in C++ and it is **used on a pointer** to the object in order to access the members or the methods in the class of the object.

For example, you would have seen me do

```
std::vector<std::vector<int>> *pointername;
pointername = new std::vector<std::vector<int>>
(10, std::vector<int>(10,0));

pointername->at(5).at(5);
```

In the line $pointername->at(5).at(5)$; of this code, I am using the arrow operator on the pointer with the name pointername. The pointer will point to a nested vector, and I am accessing the method called *at* declared and defined for a vector, which returns the element of the vector at the index specified. Here, it will return the element at index 5. Since it is a nested vector, the element returned from index 5 will still be a vector. Returning the element of the vector uses up the first at(5)

Now I have a vector, I am now no longer working on a pointer, so instead of the arrow operator, which is specfically for pointers to the object, I use the standard . dot operator to operate on the vector I have obtained. I am again accessing the emthod called *at* declared and defined for a vector, which returns the element of the vector at the index specified. Since I have here a doubly nested vector, the first at gives me the 5th element of the vector, and the next at gives me the 5th element of the 5th element of the vector.

To conclude, to access the member or methods inside a class or a struct,

- Arrow Operator $->$ is used to access the member or method from a pointer pointing to an object of that class
- Dot Operator . is used to access the member or method from an object of the class

10.7.1 this Pointer

How to call a class method from a class method in the same class? In the situation where you find that you have to use some member or function inside another method inside of the class, then you might consider using the this pointer, which allows you to access the member or methods inside the class while you are in the class. A simple example is given below

```
class classname{
        void method(some args){}
        void othermethod(some args){ this -> method(args);}

};
```

10.8 Function Overloading

We understand that in C++, what makes a function different from another function is not only the name of the function, but also the arguments of the function, i.e. C++ functions with the same name but different arguments will be treated by C++ to be different functions. C++ is smart enough to try to match the arguments you pass into it with which function matches the arguments you pass into it and use the function that matches the arguments you passed.

This is the concept of function overloading, where functions in C++ can have the same name, but be completely different functions because their arguments do not match. These functions which have the same name but different arguments are called overloaded functions of each other. In Visual Studio, if you hover over an overloaded function, Visual Studio, a.k.a. VS will tells you how many overloads does the function have, if you have the IntelliSense enabled. Other mainstay IDEs for C++ will provide such similar features.

Overloaded functions can have different return types; however, if the functions are completely the same, i.e. the functions have both the same functionname and the the same arguments, so **NOT OVERLOADED**, then you cannot have different return types, because C++ recognizes both functions as the same, and gets confused as to what to return from the function.

The general syntax for overloaded functions is

```
returnvariabletype function(args){whatever code}
returnvariabletype function(other args){whatever code}
```

An example is shown below

```
#include <iostream>

void functionname(){
        std::cout << "Function with return void called" << "\n";
}

int functionname(int argument1){
        std::cout << "Function with 1 argument called" << "\n";
        return argument1;
}

bool functionname(int argument, bool argument2){
    std::cout << "Function with 2 arguments called" << "\n";
    return argument2;
}
```

```
int main(){

    functionname();
    functionname(200);
    functionname(400, true);

}
```

This is very useful if you just have a small number of function that you want to have the same name, but their return type is different based on what arguments the user give to the function.

10.9 Operator Overloading

Operator Overloading, while a lot of times talked together with Function Overloading, should be treated slightly differently. In C++, we have seen how we can use different operators on different objects,i.e. we can use + to add ints, floats, etc and we can also use + to concatenate string. Similarly, we can use various operator to work on different objects. However, currently, our operators only work on the default objects of C++; Can we use such operators on user defined classes in C++?

The answer is of course YES!, otherwise why would we have this subsection? However, C++ cannot magically know how to use the operators on the class objects. For example, if you declare and define a class called phone, how would C++ know how to add or subtract a phone from one another? You have to clearly tell C++ what do the operation mean for the class.

Operator Overloading can be declared and defined as a class method for the class that we want the operator to act upon.

The general syntax for operator overloading inside the class:

```
class classname{
    public:
    returntype operator operatorsymbol
    (variabletype const &referencename)
    {
        // preferably some constructor too
        // Some whatsoever code
        return whattoreturn
    }
};
```

An example

```
#include <iostream>
```

```
class vector3d{

    public:
    int x, y, z;

    vector3d(int input_x, int input_y, int input_z){
        x = input_x;
        y = input_y;
        z = input_z;
    }

    vector3d operator + (vector3d const &v2add){
        vector3d result(0, 0, 0);
        result.x = x + v2add.x;
        result.y = y + v2add.y;
        result.z = z + v2add.z;
        return result;
    }

};

int main(){

    vector3d vector1 (1, 2, 3);
    vector3d vector2 (2, 3, 4);
    vector1 + vector2;

}
```

Here, inside the class vector3d, I was lazy and decided to leave everything under publc. The class vector3d has 3 members, so x, y and z. The class vector3d has a constructor that uses the input arguments to construct the x, y, z members.

The class vector3d also has an operator addition for it. **Here, the operator only needs a single argument, because the other one will be your class object.** We have said that the returntype of the operator operating on the object of the class(vector3d) and object in the argument(here is another vector3d, but **it can also be sth else**) will be vector3d(**can be something else also**). Inside the scope of the operator, I have created a new vector3d object to store the results, and set the x, y and z of the newly created vector3d object named result to be sum of the addition of the x, y, z of the vector3d that were being added. I then returned the result.

In the main, we have first constructed two vector3d objects vector1 and vector2 using the constructor, and then we have added them together.

Operator Overloading can also be declared and defined not as a method for the class we want the operator to be working upon, but as a separate function altogether too!

The general syntax in this situation is

```
returnvariabletype operator operatorsymbol
(const variabletype&, const variabletype&){

\\whatever code inside
return statement;

}
```

We note that instead of one argument like before, we now need to take in two arguments. This is because unlike before, when we declared and defined inside a class, in which one of the two operated upon was the object of the class, here we are no longer inside the class, so C++ needs you to tell it clearly what are the both the LHS of the operator and the RHS of the operator.

For example, our previous code could be easily adapted so that our declaration and definition are outside the class.

An example is shown below

```
#include <iostream>

class vector3d{

    public:
    int x, y, z;

    vector3d(int input_x, int input_y, int input_z){
        x = input_x;
        y = input_y;
        z = input_z;
    }

};

vector3d operator + (vector3d const &v1add, vector3d const &v2add){
    vector3d result(0, 0, 0);
    result.x = v1add.x + v2add.x;
        result.y = v1add.y + v2add.y;
        result.z = v1add.z + v2add.z;
    return result;
}

int main(){

    vector3d vector1 (1, 2, 3);
    vector3d vector2 (2, 3, 4);
    vector1 + vector2;

}
```

10.10 Function Template

Currently, for out C++ functions, we are currently having to declare each and every variabletype for the arguments and for what we create. This is not ok when we might want to return arrays or vectors from the function, since we must predefine our array or vector variabletype. Thus, is there someway you can try to have our array or vector return the same datatype as the datatype that the user gives us the array or vector in?

We can function template. In a function template, you can create a placeholder for the variabletype. When the code is compiled, the compiler internally generates the code based on what the variabletype given by the user, and then replaces everywhere you have set the placeolder with the specified variabletype.

The general syntax for a function template

```
template <typename placeholder>
returntype functionname(arguments)
{
   // Some code
   // return statement;
}

int main(){
    functionname <variabletypeintoplaceholder> (arguments);
}
```

An example is shown below

```
template <typename T>
T functionname(T arg1, T arg2)
{
    T result;
    result = arg1 + arg2;
    return result
}

int main(){
        functionname <int>(5, 6);
}
```

In the example, I have created a templated function with a placeholder typename called T. The function has name functionname, and will return a T, while it takes in a T as arg1 and a T as arg2. In the function I have created a T with the name result, which will be assigned the memory content of the result of the operator addition between the arg1 and arg2 Then the result is returned. In the main, I have called the function, and I am now telling C++ that we are T is simply a placeholder for int, i.e. what is inside of <>.

Now, what a human will do, is that it will generate a template function, and then replace every T in the previous paragraph with int, meaning integer. Thus, the equivalent sentences becomes

The function has name functionname, and will return a ~~T~~ int, while it takes in a ~~T~~ int as arg1 and a ~~T~~ as arg2. In the function I have created a ~~T~~ int with the name result, which will be assigned the memory content of the result of the oeprator addition between the arg1 and arg2 Then the result is returned.

For the compiler, it will do sth similar

```
int functionname(int arg1, int arg2)
{
    int result;
    result = arg1 + arg2;
    return result;
}
```

I can of course not only use int, I can also use float, short, even strings!

```
#include <string>

template <typename T>
T functionname(T arg1, T arg2)
{
    T result;
    result = arg1 + arg2;
    return result;
}

int main(){
        functionname <int>(5, 6);
        functionname <float>(2.5, 9.4);
        functionname <short>(7, 8);
        functionname <std::string>("string1", "string2");
}
```

We have used the letter T to represent our placeholder here, which is the popular choice for the placeholder. However, you can actually name your placeholder any valid name you want, it does no have to be capital letter, nor does it have to a single character, just respect the naming convention for class type in C++.

In some code, you might see sth like instead at the beginning of the function

```
template <class T>
```

Do not be alaramed, this is completely equivalent and indisticnt to

```
template <typename T>
```

We note that although the return type here was the placeholder, i.e. T, and the arguments are also placeholder, none of these need to be the placeholder. In fact,

you can have a templetized function where there is no placeholder whatsoever in the function itself. However, you should ask yourself what is the raison d'etre for the template function if you use no placeholder whatsoever?

I have to be careful that when I am using some variabletype to replace the placeholder, that I am actually using sensible variabletype. For example, if I tell C++ to replace T with int, but then I pass in two floats as argumetns, C++ will truncate the floats to become ints and give me a wrong result, unless you wanted the truncated result. I also have to make sure that the operations inside the function that I am doing are also sensible for the variabletype.

Of course, we can not only use string, floats, ints etc, we can very well pass in arrays, vectors, pointers and your own object and structs; however, we need to ensure that the operations are sensible for the variabletype.

You can of course not just have one, but more placeholders, as many as youh need. The general syntax is

```
template <typename placeholder1, typename placeholder2>
returntype functionname(arguments)
{
   // Some code
   // return statement;
}

int main(){
    functionname <variabletypeintoplaceholder1,
    variabletypeintoplaceholder2> (arguments);
}
```

An example

```
template<typename Alpha, typename Omega>
Alpha functionname(Alpha arg1, Omega arg2){

    Alpha result;
    if(arg2 == true){
        return arg1 * arg2;
    }
    else{return 0;}

}

int main(){
        functionname<int, bool>(20, 0);
}
```

Here, we have 2 placeholders, with the name Alpha and Omega. Through the main, we replace Alpha with int and Omega with bool.

10.11 Class Template

Similar to how functions can have templates, Classes in C++ can also have templates, just very so slightly different.

The general syntax for a template class in C++ is

```
template<typename placeholder>
class classname{
    \\ whatever is inside the class
};
```

An example is shown below

```
#include <string>
template<typename T>
class classname{

    public:
    T * array_pointer;

    classname(int width, int height){
        array_pointer = new T[width * height];
    }

    ~classname(){
        delete [] array_pointer;
    }
};

int main(){
    classname <int> object1(10, 10);
    classname <std::string> object2(20, 20);
    classname <bool> object3(40, 40);

}
```

Here we have created a templated class with the name classname. I was lazy so I made eveything public in the class, but you should give some good thought about what to be public and what to be private. I have created a pointer to the type T and called it the array_pointer. I have named it as such because it will be a pointer that will point to an array with elemnets of type T. I have then created a cosntructor and a destructor method for the classname. In the constructor, an array with elements of type T is created on the heap memory. In the destrcutor, the array created on the heap memory is destroyed together with the object of the class classname, abiding by our rule of **WHEN YOU CALL NEW YOU ALWAYS CALL DELETE**. We note that if we instead had here a nested vector or array, we will have work our way from the bottom up of the hierarchy through a loop.

In our main function, we demonstrate the flexibility of our template class by creating

an array of integers of the size 10 by 10 called object1. An array of strings of the size 20 by 20 called object2. An array of booleans of the size 40 by 40 called object3. We note that our row major index here is actually [i * height + j] since we actually do not have nested vectors or arrays. The array of integers, a.k.a object1 has index from 0 to 99, the array of strings, a.k.a the object2 has index from 0 to 399 and the array of booleans, a.k.a object3 has index from 0 to 1599.

For a human, we have for object1 where we told C++ that T is a placeholder for int

I have created a pointer to the type T int and called it the array_pointer. I have named it as such because it will be a pointer that will point to an array withe elemnets of type T int. I have then created a cosntructor and a destructor method for the classname. In the constructor, an array with elements of type T int is created on the heap memory. In the destrcutor, the array created on the heap memory is destroyed together with the object of the class classname, abiding by our rule of **WHEN YOU CALL NEW YOU ALWAYS CALL DELETE**.

For our compiler, we have the equivalent

```
template<typename int>
class classname{

    public:
    int * array_pointer;

    classname(int width, int height){
        array_pointer = new int[width * height];
    }

    ~classname(){
        delete [] array_pointer;
    }
};
```

You can easily imagine what would the compiler do for object2, where we told it that T is a placeholder for string, and for object3, where we told it that T is a placeholder for boolean.

We note that while we have made a destructor, what the destructor does now is simply to free up memory at the end. If we want to free up memory in the middle of the main, we can simply create a method to do so

An example method named freememory() is shown below that erases all memory allocated for the object and resets the pointer to be a NULL pointer. I demonstrate the functionality of freememory() using the object1

```
#include <string>
#include <iostream>
template<typename T>
class classname{
```

```
        public:
    T * array_pointer;

    classname(int width, int height){
        array_pointer = new T[width * height];
    }

    ~classname(){
        std::cout << "Destructor called \n";
        delete [] array_pointer;
    }

    void freememory(){
        std::cout << "Memory Begin Deleted \n";
        delete [] array_pointer;
        array_pointer = NULL;
        std::cout << "Complete Memory Deletion \n";
    }

};

int main(){
    classname <int> object1(10, 10);
    object1.freememory();
    std::cout << object1.array_pointer << "\n";
    classname <std::string> object2(20, 20);
    classname <bool> object3(40, 40);

}
```

10.12 Virtual Methods and Polymorphism

Polymorphism is the provision of a single interface to entities of different types or the use of a single name to represent multiple different types. We currently have "a single name to represent multiple different type" so the object "plant1" is both of the class CF and the class OB. We have already seen "the provision of a single interface to entitities of different types" through function and class template, i.e. templetization. Another final aspect, for now, of Polymorphism are the Virtual methods.

A virtual method is a member method which is declared and can be defined within a base class and but is overriden by a derived class. **When you try to access a derived class object using a pointer or a reference to the base class**, you can call a virtual function for that object and execute the derived classs version of the function.

In C++, you can have pointers pointing to the parent class and pointers pointing to the child class. In C++, if use a pointer to the parent class and the pointer is pointing to a parent class object, then C++ will do methdos from the parent class.

Similarly, if use a pointer to the child class and the pointer is pointing to a child class object, then C99 will do methods from the child class.

In C++, you can use a parent class pointer to point to a child class object, but you cannot use a child class pointer to point to a parent class object.

When you redirect a pointer pointing to the parent class, and point it to a child class, a question arises. **If there are two methods that are the same, i.e. same argument, same methodname in both the child and parent class**, which one should be compiler do? Well, by default, the compiler will do the method of the parent class since the method in the parent class has higher priority then the method of the child class. This can be problmeatic if we actually do not want it to do themethod in parent class, and we want it to actually do the method in the child class. Thus, we can use virtual methods.

If a method in the parent class is declared as virtual, we are explicitly telling C++ that this method in the parent class has lower priority than the method with the same methodname and arguments in the parent class, so C++ will be forced to execute the method of the child class instead of the parent class.

What you have achieved here is known as **Runtime Polymorphism**, meaning that it is during the running of the code that C++ will have to decide which method of the parent or child class will C++ use. We can also say that the **method is resolved at runtime**

Virtual methods are declared with a virtual keyword in parent class.

Purpose of Virtual Functions in C++?

- Virtual functions ensure that the correct method is called for an object, regardless of the type of reference or pointer used for, i.e. if you don't put virtual you will call the method in the parent and if you put virtual it will call the method in the child, thus you can control if it is the method in the parent or the method in the child that you will call.
- They are always defined in the parent class and overridden in a child class. If the child class has no method inside of it that has the same name and the same arguments as the virtual method in the parent class, then the parent class method is used

Rules about virtual methods in C++

- Virtual functions should be accessed using pointer or reference of base class type to achieve run time polymorphism.
- Virtual functions cannot be static. (It would be a super big mess if it was static, since which class should the method be static for? The parent or the child class?)

- The virtual functions should be the same in the base as well as derived class. (We need the virtual function in the parent and the correspnding function in the child class to have the same name and arguments for C++ to understand)
- A virtual function can be a friend function of another class. (Maybe more on this later)
- A class may have virtual destructor (virtual destructor is sth new in C++ 20 and generally not used due to lack of backwards compatibility) but it cannot have a virtual constructor. (Because C++ construct the parent class and then the child class on top of the parent, if you override the constructor, how would C++ how to construct the parent class? i.e. imagine you have a chicken and an egg. You need to first construct the chicken then you can use the Chicken to construct the egg. Now if you override the method to construct a chicken with the method to construct an egg, you can no longer construct a chicken, then how are you going to construct the egg?)

The general syntax for Virtual Methods is

```
#include <iostream>

class parentclassname {
public:
    virtual returnvariabletype functionname()
    {
        //do method stuff
        // maybe return statement
    }
};

class childclassname : public parentclassname {
public:
    returnvariabletype functionname()
    {
        //do method stuff
        //maybe return statement
    }
};

int main()
{
    parentclassname* parentclasspointer;

    childclassname childclassname;

    parentclasspointer = &childclassname;

    parentclasspointer->functionname();

}
```

And an example piece of code

```
#include <iostream>

class parentclassname {
public:
    virtual void functionname()
    {
        std::cout << "Parent class method called" << "\n";
    }
};

class childclassname : public parentclassname {
public:
    void functionname()
    {
        std::cout << "Child class method called" << "\n";
    }
};

int main()
{
    parentclassname* parentclasspointer;

    childclassname childclassname;

    parentclasspointer = &childclassname;

    parentclasspointer->functionname();

}
```

Here, I have created a parent class with the name parentclassname, that has as its public member a method with the name functionname that takes no arguments that has as returnvariabletype to be void. The method is virtual because there is a virtual keyword in front of the method.

I have then created a child class with the name childclassname, that will inherit everything public from the parent class with the name parentclassname. See the inheritance subsection for the syntax here. In the child class, it has a public method with the name functionname (same as the correpsonding virtual method in the parentclass) with no arguments (same as the corresponding virtual method in the parentclass) that has returnvariabletype to be void.

The method functionname in the parenclass and the childclass will give different outputs.

In the main, I have created a pointer to an object of the parentclass called parentclasspointer. I have then created an object of the childclass called childclassname. Then I have used the dereference operator to get the memory address of the child class object and then used assignment operator to assign the memory address of the

child class object to the the parent class pointer, effectively making the parent class pointer point towards the child class object.

In the final line, the pointer which was declared to be pointing to the parent class in the 1st line, and was redirected to point a child class object instead in the 3rd line, was used to access the method functionname() inside using the arrow operator.

Because we have used virtual, the child class method will override the corresponding base class method, resulting in the console out statement from the child class being executed. If we did not have the virtual keyword, then it will be the console out statement from the parent class being exeucted.

10.13 Smart Pointers

This could be useful, but you could also just use pointers and instead train up your ability to understand when to call new and delete.

11 Parallelization Conceptualization

Before we move to anything about parallelization of code, let's try to get the concept of parallelization across. It is far more important for you to undestand parallelization than to actually do parallelization.

Imagine you are the boss, or as some would call it, the upper management, and you have some workers under you. Because you are in the world of C++, where there is no worker protection and rights, you do not have to worry about such things. Your desire here is to extract as much blood and sweat and toll from your workers, with little consideration for their welfare or health. Your workers are very capable, but they are lacking any sort of human common sense.

You are very excited since climbing up the corporate which allows a greater opportunity to climb even higher in the corporate ladder. To reach your KPI, i.e. the key performance indicator, you want to make your workers work as much as possible. When there was a single worker, you just needed to exploit the single worker that you had, sometimes making the poor proletariat do twice the work for a marginal increase in payment, and sometimes overfeeding him all sorts of questionable stimulus that decrease the proletariat's lifespan and may cuause the bodily collapse but increases his efficiency in the present moment. You are greatly saddened that the physical limitations of your workers mean they cannot do thrice the work. Anyways, the worker is nothing but a cog in the corporate machine, and cogs are easily replaceable, so while you try to appear that you are sensible boss who is being forced to do so by your superiors, whether you truly care for the worker or not is something else entirely, and at least seems of no concern to your superiors. You have of course achieved equality in the workplace, because eveyrone in your workforce is equally exploitable and replaceable.

Being the brutally effective boss that you are, you know that now that you have more

workers under your control, the best way to proceed is to make each worker work as much as possible, instead of a single worker doing everything and the rest just watching that guy doing nothing. You will want all of your workers to work as much as possible, with no breaks whatsoever, i.e. no coffee break, no bathroom break, no smoking breaks etc. In real world, a typical way to do so is to simply remove the existence of bathrooms in the corporate building.

However, you know that you cannot make your workers work continuously actually, at least not for all jobs. There are some jobs where the work could be easily distributed between people. For example, you bring your team to construction shop to acquire the neccesary materials. You can easily tell one worker to get the cement, the other worker to get the steel beams, and another to get the pipes, and some other to get the wires etc. This situation you can easily share the work between people, since each one can do the work, and the work of each worker is not affected by another worker, i.e. the guy getting the cement does not have to wait for the guy getting the steel beams etc.

There are also jobs where one worker has to wait for another worker. For example, after you have purchased the material succcesfully and found out and punished the worker that decided to buy something else that what he was told to,you decided to proceed with your construction project. We are not doing modular construction here, but the classical construction. This time, you need to do your construction project in multiple different steps, and cannot just tell one worker to work on the roof while the foundation still has not been laid, since there is no roof without foundation. You will of course not try to make a single poor soul do the entire job, being the sensible boss that you are; you are going to exploit all of your workers. And being egalitarian, your exploitation will be equally harsh on all your workers, leaving no lazy soul. You can of course tell multiple different workers to work together on laying the foundation, then you can tell multiple different workers to construct the frame, then you can tell multiple different workers to construct the walss etc. However, this still presents a bottleneck, meaning that the people constructing the frame have to wait all the way until the foundation has been finished.

Another, more exotic, but prone to problem approach that might work sometimes, is to make the worker in the next step work on the finished parts of the previous step and only wait when all the parts left of the previous step are still unfinished and being worked upon. Remembering that your workers lack common sense, you have to make sure to instill common sense into them, i.e. telling them to be polite and wait for the previous worker to finish.

You have now gotten a really big headache trying to make your workers do everything effectively, and you realize that while you have a lot more workers, who can work simultaneously to make your job a lot faster, you need to spend considerable time and effort to try to coordinate them so that you can ensure that everyone is working 24/7. You decide that for jobs where the time taken is of little concern to you but the client wants a plan fast, you decide that this headache is not worth it, and just decide to leave it to a single worker. You decide that a headache is only worth it if the job is sufficiently large, the client is willing to wait and fund for a good efficient plan, and the time taken for the construction is significantly more than the time taken for your planning, and thus the time saved from an effective plan is worth your effort.

You also understand that an effective plan for each job is different, i.e. some jobs are really easy to plan for, and you get a significant time improvement from the plan. You also note that many other jobs are more difficult to plan for. You also note that many jobs, even with the best plan that you have created, might only have marginal time improvements. Thus, you have to make a decision on a per job basis whether you want to make a plan.

In the past, you started out as the boss of one worker; but now, since you have made some capital by exploiting the worker's labor, you have gotten some more workers. However, you notice that having a few more workers makes your management situation more complicated than when you had a single worker, especially when coordinating the work of your workers.

In the past, because you had a single worker, all the tools and neccesary info for the work just needed to be carried by a single worker, so each worker only needed a single toolbox.

However, as you have expanded the number of workers that you have under you, you find out that a single toolbox might not be enough for your worker anymore, since you have seen that when two or more workers need the same tool out of the toolbox, a problem appears because there is only a single tool that cannot easily be shared between the workers. Now you have two possible solutions to this problem. You either increase the production cost slightly by acquiring some more toolboxes for your tools. This is helpful if you are willing to spare the cost for increased efficiency. However, you can also make the workers wait for one another; however, one worker will be completely doing nothing while it waits for the other worker to finish the work. Whether you have the funds to buy a new toolbox, or the time to allow workers to wait for each other will decide which approach you will take.

12 Parallelization using OpenMP

13 Parallelization using MPI