

## General Reminders

<code>#include "myfile.h"</code>	Include file.
<code>rand()</code>	Random int, <code>#include&lt;cstdlib&gt;</code> .
<code>int(var)</code>	Convert a data type var to int.
<code>float(var)</code>	Convert a data type var to float.
<code>double(var)</code>	Convert a data type var to double.
<code>static_cast&lt;t&gt;(var)</code>	Convert var to the type t.
<code>void myF() const</code>	read only fonction.
<code>inline</code>	the whole code of the inline function is inserted or substituted at the point of its call during the compilation.
<code>constexpr</code>	that specify that an expression must be evaluated at compile time.
<code>sizeof(var)</code>	return the number of bytes used by the variable. <code>sizeof</code> runs at compile-time.
<code>move(obj)</code>	During assignation move the already existing object instead of creating a copy of it (memory optimizations).
<code>ternary Operator</code>	condition ? ifTrue : ifFalse

## Strings

<code>str[i]</code>	Get or set the char at the index i.
<code>.length()</code>	Return the number of characters.
<code>.substr(a,b)</code>	Returns the substring starting at index a with length b.
<code>.find(subStr)</code>	Return the start index of the substring
<code>.replace(i,l,str)</code>	Replace l characters starting at index i with str
<code>#include &lt;string&gt;</code>	<b>all the above must include string.</b>
<code>stoi(str)</code>	Convert a string to int.
<code>stof(str)</code>	Convert a string to float,
<code>stod(str)</code>	Convert a string to double,
<code>to_string(var)</code>	Convert var to a string,

## Arrays

0	1	...	n
"Max"	"Tom"	...	arr[n]

This table illustrate the structure of an array of strings. Considering that n is equal to the number of element minus one. Arrays are a static data type.

<code>int arr[4];</code>	Create a array of int and with 4 element.
<code>int arr[4]={6,3};</code>	
<code>arr[i]</code>	Get or set the element at the index i.

## Vectors

<code>#include&lt;vector&gt;</code>	Include vector library.
<code>vector&lt;type&gt; V;</code>	Instantiate a vector.
<code>vector&lt;type&gt; V(size);</code>	Instantiate a vector from Array obj.
<code>vector&lt;type&gt; V{6,3,3};</code>	Instantiate a vector from Array.
<code>V=vector&lt;type&gt;();</code>	Re-instantiate V.
<code>.at(i)</code>	Returns the element at index i.
<code>.size()</code>	Return the number of elements.
<code>.push_back(Value)</code>	Add the new element at the end.
<code>.pop_back()</code>	Remove the last element.
<code>.clear()</code>	Empty the vector.
<code>.insert(i, Value)</code>	Insert Value at i.

## Structures

```
struct myStruct {
    string param1;    // attribute 1
    double param2;    // attribute 2
}s1, s2;              // myStruct instances
```

`myStruct Obj;` instantiate structure object.  
`Obj.param1` Access param1 of Obj.

## Streams

<code>#include&lt;fstream&gt;</code>	Include stream library.
<code>#include&lt;sstream&gt;</code>	Include string stream library.
<code>ifstream fin;</code>	Instantiate a input stream.
<code>ofstream fout;</code>	Instantiate a output stream.
<code>stringstream s(str);</code>	Instantiate a string stream.
<code>myS.open("file.txt")</code>	Open txt file with the stream.
<code>myS.close()</code>	Close the stream file.
<code>getline(fin, line)</code>	Get the next line from fin.
<code>fout&lt;&lt;"hello"</code>	Output in stream "helloWorld".
<code>fin&gt;&gt;var</code>	Input from stream to var.
<code>&lt;&lt;setprecision(n)&lt;&lt;</code>	Set decimal points, <code>#include&lt;iomanip&gt;</code> .
<code>&lt;&lt;setw(n)&lt;&lt;</code>	Establishes a print field of n spaces.
<code>&lt;&lt;fixed&lt;&lt;</code>	Display floating point numbers in fixed. point notation.
<code>&lt;&lt;showpoint&lt;&lt;</code>	Enables or disables the unconditional inclusion of the decimal point character
<code>&lt;&lt;noshowpoint&lt;&lt;</code>	in floating-point output.
<code>&lt;&lt;left&lt;&lt;</code>	output the string on the left.
<code>&lt;&lt;right&lt;&lt;</code>	output the string on the right.

## clear buffer

The buffer must be cleared after after getting an input from a stream if you input and output in the same file at the same time.

```
if(cin.fail() == true) {
    cin.clear();
    cin.ignore(1000, '\n');
}
```

## cmath

<code>#include&lt;cmath&gt;</code>	Include cmath library.
<code>sqrt(x)</code>	Square root of x.
<code>pow(x, y)</code>	x raised to the power y.
<code>abs(x)</code>	Absolute value overloads.
<code>floor(x)</code>	Greatest integer $\leq x$ .
<code>ceil(x)</code>	Smallest integer $\geq x$ .
<code>fmod(x, y)</code>	Floating-point remainder of x/y.

## Error Handling

```
try {
    // risky operation
} catch (exceptions) {
    // runs if an exception of type Ex is thrown
}
```

<code>#include&lt;cassert&gt;</code>	Include assert library.
<code>#include&lt;stdexcept&gt;</code>	Common standard exceptions.
<code>throw myException</code>	Throw an error of type myException.
<code>exception::what()</code>	Retrieve diagnostic message.
<code>catch (const auto&amp; e)</code>	Catch exceptions by const reference.
<code>catch(...)</code>	Fallback handler; rethrow if unsure.
<code>exception</code>	Parent of all exceptions class.

## Object Oriented Programing(OOP)

```
class myClasses :public parentClass{
private:
    // private methods and variables
public:
    // public methods and variables

    myClasses(int p1, int p2){...} // Constructor

    ~myClasses(){...} // Destructor

    // Override the inherited method parentMethod()
    void parentMethod() override { ... }

    //Example Operator Overloading
    Number operator+(const Number &n){
        return Number(value + n.value);
    }
};
```

<code>myClasses myObj(3,5);</code>	Instantiate an myClasses type obj.
<code>myClasses myObj;</code>	Call the default constructor.
<code>protected:</code>	similar to private, but it can also be accessed in the inherited class.
<code>virtual</code>	Specify that a method can be overridden in a derived class.

## OOP With header file

If you use a header the file wich contain the main function must include the header file.

### Header file(myHeader.h)

```
#ifndef MYCLASS_H //if no def for MyClass
#define MYCLASS_H //else

using namespace std;

class MyClass{
public:
    :
private:
    :
};
#endif
```

### Class file(.cpp)

```
#include "myHeader.h"

MyClass::MyClass(int p1, ...){
    publicAttribute = p1;
    :
}
```

## Genericity

```
template <typename T_1, ..., typename T_n>
class myClasse{
    :
}
```

```
myClasse<T_1, ..., T_n>(...);
```

## Switch case

```
switch (x){
    case 0:
        /*Code in case x = 0*/
        break;
        :
    case n:
        /*Code in case x = n*/
        break;
    default:
        /*Code if no case match*/
}
```

## Pointer & References

<code>int*</code>	<code>myInt;</code>	* means myInt work form a pointer.
<code>new</code>		dynamically allocate a block of memory.
<code>delete</code>		release dynamically allocated memory.
<code>NULL</code>		Macro that referens to null pointer.
<code>*var</code>		Get var value, where var is a pointer.
<code>&amp;var</code>		Get memory addresse of <b>var</b> .
<code>void*</code>	<code>var</code>	Pointer with no associated data type.

## Lambda Expression

```
... = [captureClause] (parameters) -> returnType {
    // definition}
```

<code>[&amp;]</code>	capture all external variables by reference.
<code>[=]</code>	capture all external variables by value.
<code>[a,&amp;b]</code>	capture 'a' by value and 'b' by reference.

## Low Level Data Types

### signed fixed width integer types

<code>int8_t</code>	<code>int16_t</code>	<code>int32_t</code>	<code>int64_t</code>
<code>int_fast8_t</code>	<code>int_fast16_t</code>	<code>int_fast32_t</code>	<code>int_fast64_t</code>
<code>int_least8_t</code>	<code>int_least16_t</code>	<code>int_least32_t</code>	<code>int_least64_t</code>

### unsigned fixed width integer types

<code>uint8_t</code>	<code>uint16_t</code>	<code>uint32_t</code>	<code>uint64_t</code>
<code>uint_fast8_t</code>	<code>uint_fast16_t</code>	<code>uint_fast32_t</code>	<code>uint_fast64_t</code>
<code>uint_least8_t</code>	<code>uint_least16_t</code>	<code>uint_least32_t</code>	<code>uint_least64_t</code>

### other integer types

<code>intmax_t</code>	<code>&amp; uintmax_t</code>	Maximum-width integer type.
<code>intptr_t</code>	<code>&amp; uintptr_t</code>	Integer types capable of storing a pointer value.
<code>size_t</code>		An unsigned integer data type to represent the size of objects in bytes; commonly used for array indexing and loop

## Preprocessing

<code>#define NAME value</code>	Define a macro.
<code>#define F(x) x*x</code>	Define a function-like macro.
<code>#ifdef NAME</code>	If the macro <code>NAME</code> is defined.
<code>#ifndef NAME</code>	If the macro <code>NAME</code> is not defined.
<code>#else</code>	Alternative case for <code>ifdef/ifndef</code> .
<code>#endif</code>	End conditional directive.
<code>#include</code>	Include a file.
<code>__FILE__</code>	Current file name.
<code>__LINE__</code>	Current line number.
<code>__DATE__</code>	Compilation date.
<code>__TIME__</code>	Compilation time.
<code>#pragma</code>	Implementation-specific instruction.

• `once` — simple include guard for header files. • `pack(push, n)` / `#pragma pack(pop)` — set and restore struct packing/alignment to `n` bytes. • `pack(n)` — set struct member alignment to `n`. • `GCC optimize("...")` — enable compiler-specific optimizations (GCC/Clang). • `#pragma warning(push)` / `#pragma warning(pop)` / `#pragma warning(disable:NNNN)` — control MSVC warnings. • `#pragma message("text")` — emit a compile-time message. • `#pragma comment(lib, "name.lib")` — instruct MSVC linker to link a library. • `#pragma omp parallel`

— OpenMP parallel region (requires OpenMP enabled by compiler flags). • `#pragma intrinsic(func)` — request compiler intrinsic (MSVC).

## Compiler Commends

<code>clang++ fileName</code>	commend to compile c++ code with <b>clang</b> , clang is a LLVM compiler.
<code>-o name</code>	define the name of the compiled object.
<code>-v</code>	Makes the compiler print detailed information. "v" stands for "Verbose".
<code>-E</code>	Prints the preprocessor output.
<code>-Wall</code>	activates all warnings
<code>-Wextra</code>	Enable extra warnings beyond <code>-Wall</code> .
<code>-c fileName</code>	generate an object file. To add .o filse to the compilation simply add those like a regular file.
<code>-g</code>	Include debug symbols for gdb/lldb.
<code>-gline-tables-only</code>	Minimal debug info for profilers.
<code>-O0, -O1, -O2, -O3, -Ofast</code>	Optimizations levels, where <code>-O0</code> is no optimization