C++ Cheat Sheet

Class: CSCI 1300

General Reminders

`#include "myfile.h"`	Include file.
`rand()`	Random int, `#include<cstdlib>`.
`int(var)`	Convert a data type var to int.
`float(var)`	Convert a data type var to float.
`double(var)`	Convert a data type var to double.
`static_cast<t>(var)`	Convert `var` to the type `t`.
`void myF() const`	read only fonction.
`inline`	the whole code of the inline function is inserted or substituted at the point of its call during the compilation.
`constexpt`	that specify that an expression must be evaluated at compile time.
`sizeof(var)`	return the number of bytes used by the variable. `sizeof` runs at compile-time.
`ternary Opearator`	condition ? ifTrue : ifFalse

Strings

`str[i]`	Get or set the char at the index `i`.
`str.length()`	Return the number of characters.
`str.substr(a,b)`	Returns the substring from a to b.
`str.find(subStr)`	Retrun the start index of the substring
`str.replace(i,l,str)`	Replace substring from `i` to `l` with `str`
`stoi(str)`	Convert a string to int, `#include<string>`.
`to_string(var)`	Convert var to a string, `#include<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.

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

Vectors

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

Structures

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

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

Streams

`#include<fstream>`	Include stream library.
`#include<sstream>`	Include string stream library.
`ifstream fin;`	Instantiate a input stream.
`ofstream fout;`	Instantiate a output stream.
`stringstream s(str);`	Instantiate a string stream.
`myS.open("file.txt")`	Open txt file whith the stream.
`myS.close()`	Close the stream file.
`getline(fin, line)`	Get the next line from fin.
`fout<<"hello"`	Output in stream "helloWorld".
`fin>>var`	Input from stream to var.
`<<setprecision(n)<<`	Set decimal points, `#include<iomanip>`.
`<<setw(n)<<`	Establishes a print field of n spaces.
`<<fixed<<`	Display floating point numbers in fixed. point notation.
`<<showpoint<<` `<<noshowpoint<<`	Enables or disables the unconditional inclusion of the decimal point character in floating-point output.
`<<left<<`	output the string on the left.
`<<right<<`	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) {
    cout << "cin failed state";
    cin.clear();
    cin.ignore(1000, '\n');
}
```

cmath

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

Error Handling

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

`#include<cassert>`	Include assert library.
`#include<stdexcept>`	Common standard exceptions.
`throw myException`	Throw an error of type myException.
`exception::what()`	Retrieve diagnostic message.
`catch (const auto& e)`	Catch exceptions by const reference.
`catch(...)`	Fallback handler; rethrow if unsure.
`exception`	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){
            // Body of constructor
        }

        ~myClasses(){
            // Body of destructor
        }
};
```

`myClasses myObj(3,5);`	Instantiate an myClasses type obj.
`myClasses myObj;`	Call the default constructor.
`protected:`	similar to private, but it can also be accessed in the inherited 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:
        MyClass(); //default constructor
        MyClass(p1, p2); //parameterized constructor
        int publicAtribute;
        void myFunction() const;
    private:
        int privAtribute;
};
#endif
```

Class file(.cpp)

```
#include<iostream>
#include "myHeader.h"

MyClass::MyClass(){
    publicAtribute = 0;
    privAtribute = 0;
}

MyClass::MyClass(int p1, int p2){
    publicAtribute = p1;
    privAtribute = p2;
}

void MyClass::myFunction() const{
    // my code
}
```

Switch case

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

Pointer & References

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

Lambda Expression

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

$[\&]$	capture all external variables by reference.
$[=]$	capture all external variables by value.
$[a, \&b]$	capture 'a' by value and 'b' by reference.

Low Level Data Types

signed fixed width integer types

`int8_t`	`int16_t`	`int32_t`	`int64_t`
`int_fast8_t`	`int_fast16_t`	`int_fast32_t`	`int_fast64_t`
`int_least8_t`	`int_least16_t`	`int_least32_t`	`int_least64_t`

signed fixed width integer types

`uint8_t`	`uint16_t`	`uint32_t`	`uint64_t`
`uint_fast8_t`	`uint_fast16_t`	`uint_fast32_t`	`uint_fast64`
`uint_least8_t`	`uint_least16_t`	`uint_least32_t`	`uint_least6`

other integer types

`intmax_t & uintmax_t`	Maximum-width integer type.
`intptr_t & uintptr_t`	Integer type capable of holding a pointer to void.
`size_t`	An unsigned integer data type to represent the size of objects in bytes.