Welcome to GeoTheo’s C++ Notes! You can only follow this guide if you can’t remember something… It doesn’t teach you C++. I’ll leave notes for the most useful or hardest concepts and methods.

**Object Oriented Programming (OOP)**

**Classes**

Classes names always start with a capital letter. The syntax for classes goes like that:

class Car {

public:

int wheels;

bool steeringWheel;

string fuel;

private:

string keys;

};

By not demonstrating if the following variables are private or public, the compiler will take them as private.

In the main we can call as many instances of a class we want, provided that each instance has a different name and these instances are called objects:

int main()

{

car firstCar;

car secondCar;

firstCar.wheels = 4;

firstCar.steeringWheel = true;

firstCar.fuel = “diesel”;

secondCar.wheels = 6;

secondCar.steeringWheel = false;

secondCar.fuel = “diesel”;

cout << "The first car has " << firstCar.wheels << " wheels. However, the second car has " << secondCar.wheels << “ Wheels.” << endl;

}

Output: The first car has 4 wheels. However, the second car has 6 Wheels.

The problem with this way of declaring objects is that we have to declareeach object’s variables one by one and this is annoying if we have 10 variables. What we can do is create a constructor.

**Constructors**

What a constructor actually is, it’s a function inside a class which has the same name as the class in which it is in:

class Car {

public:

int wheels;

bool steeringWheel;

string fuel;

//constructor’s syntax:

Car(int getWheels, bool getSteeringWheel, string getFuel) { //same name as the class “Car” with a capital letter as well.

wheels = getWheels;

steeringWheel = getSteeringWheel;

fuel = getFuel;

}

};

If you want you can also have the names the same but you have to specify that the class’s variables are same with the contructor’s variables. To do this write:

Car(int wheels, bool steeringWheel, string fuel) { //same name as the class “Car” with a capital letter as well.

this->wheels = wheels;

this->steeringWheel = steeringWheel;

this->fuel = fuel;

}

When we initialize an object from a class which has a constructor, we can initialize its variables as well at the same line, because we can initialize variables as we do in a function because then, getWheels will be equal to wheels etc.:

int main()

{

Car firstCar(4, true, "diesel"); //getWheels = 4, getSteeringWheel = true, getFuel = “diesel”

Car secondCar(6, false, "diesel");//getWheels = 6, getSteeringWheel =false, getFuel = “diesel”

cout << "The first car has " << firstCar.wheels << " wheels. However, the second car has " << secondCar.wheels << “ Wheels.” << endl;

}

Output: The first car has 4 wheels. However, the second car has 6 Wheels.

**Get & Set (Private Variables)**

When we declare private variables, we can access them from the main using an indirect method: get and set. So get and set will be two functions inside the class with the private variable. These functions have to be public. This is possible because private properties are accessible to the class they’re in. Here’s how it’s done:

class Worker {

private:

int salary;

public:

void setSalary(int getSalary) {

salary = getSalary;

}

int getSalary() {

return salary;

}

};

int main() {

Worker giwrgos;

giwrgos.setSalary(5000);

cout << giwrgos.getSalary();

}

**(Class) Inheritance**

Inheritance is about a class inheriting another class’s variables – properties. The class that is inherited from another class is called a parent and a class that inherits a class’s properties is called a child. Now, if a class inherits from a child, it’s called grandchild because it also inherits the parent’s properties as well. This is the syntax for inheritance:

class Phone { //This is the Parent

public:

int buttons;

bool frontCamera;

string os;

};

class PhoneV2 : public Kinhto { //child - initializing a class as a sequel for Phone

public:

string color; //extra phone’s variable that “Phone” doesn’t have

KinhtoV2(int buttons, bool frontCamera, string os, string color) { //note that we initialize the constructor in the child and not in the parent class

This->buttons = buttons;

This->frontCamera = frontCamera;

This->os = os;

This->color = color;

}

};

class PhoneV3 : public PhoneV2 { //grandchild - initializing a class as a sequel for Phone and PhoneV2

};

int main() {

PhoneV2 newPhone(14, true, "Realme", “Cyan”);

}

**Marriage (I don’t know its actual name)**

Marriage lets us combine the properties (variables and functions) of 2 independent classes into one. It is like inheritance but with multiple objects.

Syntax:

class Father {

public:

string fatherName;

void fatherMessage() {

cout << "Hello from Dad!” << endl;

}

};

class Mother {

public:

string motherName;

};

class Combine : public Pateras, public Mhtera {

//here I can use fathername and motherName.

void motherMessage() {

cout << "Hello from “ << motherName << endl;

}

};

int main() {

//pantrema

Combine ultimateObject;

ultimateObject.motherName = "mama";

ultimateObject.fatherName = "Babas";

ultimateObject.motherMessage();

ultimateObject.fatherMessage();

}

**Constructor’s Inheritance**

Constructor’s Inheritance is just like a class’s inheritance.

Syntax:

class Car {

public:

int wheels;

bool steeringWheel;

string fuel;

Car(int wheels, bool steeringWheel, string fuel) {

this-> wheels = wheels;

this-> steeringWheel = steeringWheel;

this-> fuel = fuel;

}

};

class Car2 : public Amaksi { //We use actual inheritance, to access “Amaksi’s” variables and constructor

public:

string color; //new variable

Amaksi2(int getWheels, bool getSteeringWheel, string getfuel, string getColor) : Car(getWheels, getSteeringWheel, getfuel) {

wheels = getWheels;

steeringWheel = getSteeringWheel;

fuel = getfuel;

color = getColor;

} //we use “Car”s variables as well as “Car2”s variables to create a contructor

};

int main() {

Car2 quickCar(4, true, "diesel", "black");

cout << quickCar.color;

Car anotherCar(); //this is not necessary

}

**Internal and external use of member functions (methods)**

Methods or member functions are just functions inside classes. We can use methods inside and outside classes though the initialization of these methods has to be inside the class. Syntax:

class AgoodClass {

public:

void memberFunction1() {//internal use of member function

cout << "first method";

}

void memberFunction2();//initialization of external member function

int memberFunction3(int value); //another example

int memberFunction4(int value) { //another example

return value;

}

};

void AgoodClass:: memberFunction2() {// extrenal (from class) use of member function

cout << "first member function use outside of a class";

}

int MiaKlash::methodos3(int value) {

return value;

}

int main() {

AgoodClass class1;

class1.memberFunction3(1000);

class1.memberFunction4(120);

}

**Polymorphism**

Polymorphism lets you use the same name for multiple methods that are inherited between them. Depending on which child you call, the program will run th e function not by its name but it will run the last function it finds.

Example:

// Base class

class Animal {

public:

void animalSound() {

cout << "The animal makes a sound \n";

}

};

// Derived class

class Pig: public Animal {

public:

void animalSound() {

cout << "The pig says: wee wee \n";

}

};

class Dog: public Animal {

public:

void animalSound() {

cout << "The dog says: bow wow \n";

}

};

int main() {

Animal myAnimal;

Dog myDog;

myAnimal.animalSound();

myDog.animalSound();

return 0;

}

**Useful Commands**

Pro tip: global variables (variables that are initialized under no class or function) are more useful than you think.  
Remember to use while instead of for when you don’t know how many times you want a function of the app to loop

**Pause Execution**

Sleep(1000); //sleep command in milliseconds - You don’t need to include anything

**True Random**

I don’t know any library for pseudo Randoms in C++ but it doesn’t matter because true Randoms are just better. For true random number generation you need this line before you use the rand() function:

srand(time(0)); OR srand(time(NULL));

Without this, the program generates the same number every time it’s ran. Now pay attention because the way rand() is used is very tricky. This command generates a number from 0 to 10:

rand() % 10

To specify a minimum number you want the generation to start from, you have to specify the lowest (minimum) number and then specify how much more you want. So you add the number and the sum of these two numbers has to be the maximum number. This is too difficult to express… Example:

So in this case, we want a random generation from 5 to 10. The code will look like this:

rand() % 5 + 5

To make it easier, note that this is the same as the previous command: (rand() % 5) + 5

What happens behind the compiler, is that it will firstly fiind a number from 0 to 5 right?

(rand() % 5) + 5. So the output until here is 0 (lowest) and 5 (greatest). The next step that the compiler has to do, is to add 5 to that output: (rand() % 5) + 5. So, now the output is 0 + 5 (lowest) to 5 + 5 (greatest). This means from 5 to 10.

Pro tip: to find the numbers you need to write: put the lowest number first: rand() % 5 + 5

Then, substract this number (the minimum number) from the maximum number (10), (so we have: 10 – 5 = 5 ) and add this difference with the minimum: rand() % 5 + 5

**Preferred way of initializing dynamic arrays**

When you want to use arrays of unknown size, you should use this way:

int\* array = new int[userInput];

Because that is the only portable way to initialize arrays – works on all compilers. After you are done with the array you should free its memory by adding this command:

delete[] array;

**Files (fstream)**

You need to include fstream for this to work: #include <fstream>

and probably string?

//Best way to Write text in a file:

fstream file;

file.open("file.txt", fstream::app);

file << "This is content in a file!\n";

//Best way to get text from a file in a variable:

string content;

ifstream file("file.txt");

while (getline(file, content)) { //note: getline goes line by line, so it goes down by one line each loop

cout << content << endl;

}

file.close(); //Closing the file because else it stays open in the memory

**Windows CMD Commands**

When coding console applications you can manipulate the console by sending batch commands. This means that you can do anything C++, you can with Windows CMD. You just have to include windows.h. The syntax is:

system("Any CMD command");

Examples:

system("shutdown - s - t 1");

system("ipconfig /release");

system("timeout 2 > nul"); // “ > nul” makes the console not print “Waiting for 2 seconds… Press any button to skip”