OOPS IN C++

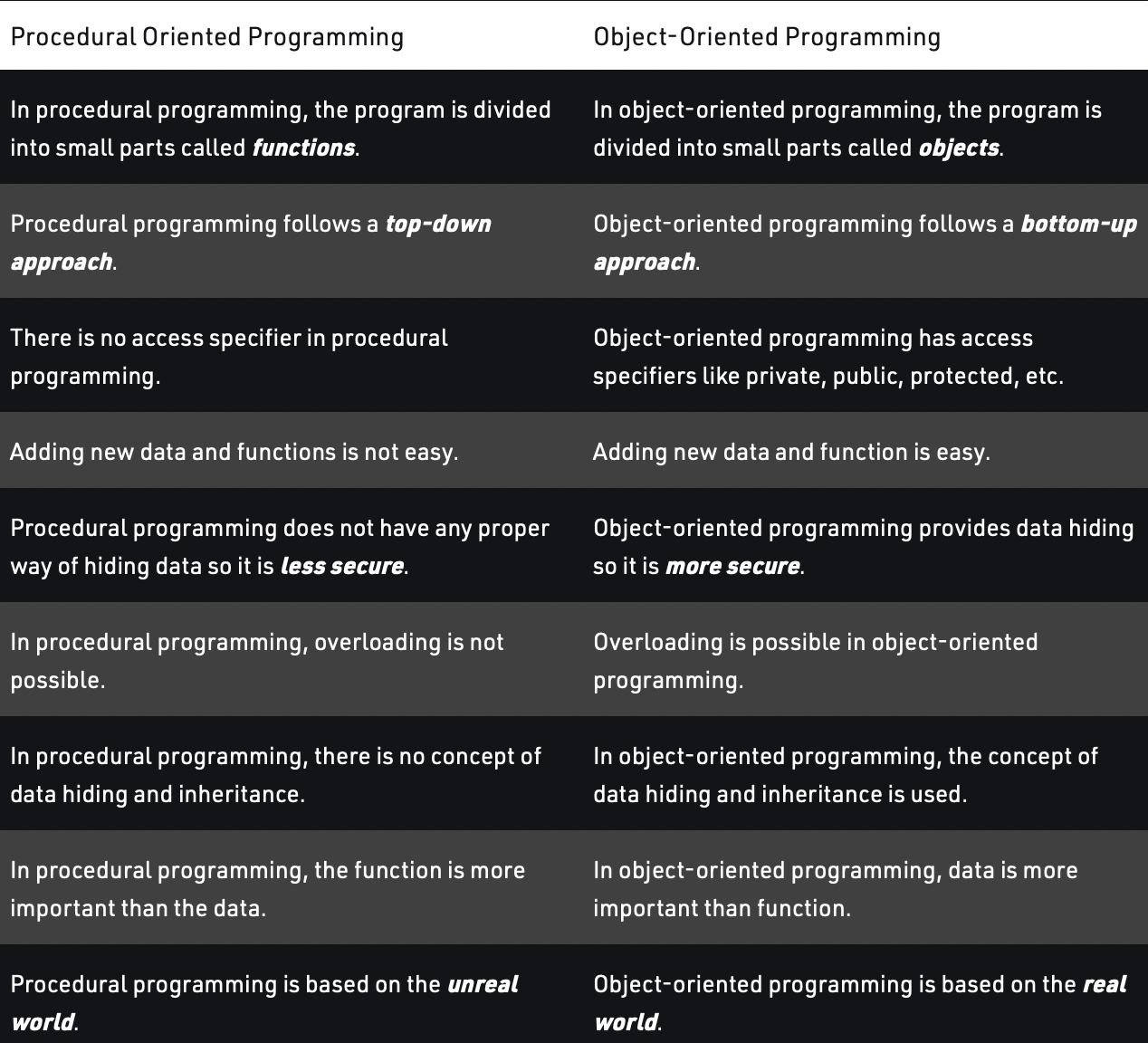
//reference 🡪 <https://whimsical.com/object-oriented-programming-cheatsheet-by-love-babbar-YbSgLatbWQ4R5paV7EgqFw>

Class is a user defined data type(template) and object is the property or implementation of the class.

\*\* Size of the object is the size of the properties declared but in case of no properties/empty class the size comes as 1 not 0. This is done to keep track.

In C++, a structure is the same as a class except for a few differences. The most important of them is security.

A Structure is not secure and cannot hide its implementation details from the end-user while a class is secure and can hide its programming and designing details.



**LIMITATIONS OF PROCEDURAL PROGRAMMING**

🡪 If the structure of the data is changed then all the functions need to be changed to work in accordance to it.

**class** person

{

**char** name[20];

**int** id; // Attribute

**public**: // Access specifier

**void** getdetails(){} // this is a member function

};

**int** main()

{

   person p1; // p1 is an object

}

Pointer of objects

\*\* It is allocated dynamically and then freed up when we are done with it by using delete operator. In this case the object is created in the heap memory.

Eg.

class student

Student \*chan = new student();

Delete chan;

Array of objects

Class students

Students [] = {s1,s2,s3};

We can initialize the values of the variables in the class so that all the objects that are created from that class get the same initial value.

Class members

These are the attributes of the objects .

We can access the class member by using ‘.’ Operator. Eg. chan.marks;

A **class pointer** is a **pointer variable** that stores address of an **object** of a class. In case of pointer to a class object which is dynamically allocated to the heap. We will dereference the pointer and then use dot operator.

Eg.

student \*chan = new student;

(\*chan).marks;

Or

Chan ->marks;

Class Methods

Methods are **functions** that belongs to the class.

We can either define the function (a) Inside class definition (b) Outside class definition

Class member access modifiers

It tells the compiler what class members are to be accessible outside the class and what members are to be the part of public interface.

Private 🡪 by default this is activated

Public 🡪

Protected 🡪

Constructors

A constructor in C++ is a **special method(function)** that is automatically called when an object of a class is created. To create a constructor, use the same name as the class, followed by parentheses (). It can be overloaded.

*#include* <bits/stdc++.h>

using namespace std;

class MyClass { *// The class*

public: *// Access specifier*

MyClass() { *// Constructor*

cout << "Hello World!";

}

};

int main() {

MyClass myObj; *// Create an object of MyClass (this will call the constructor)*

*return* 0;

}

Polymorphism

Different forms of the same thing is [polymorphism](polymorphism.cpp).

Eg. a man can be a son and a father at the same time.

Eg. a person has so many hobbies so at a time he can be a swimmer , writer , coder etc.

It can be implemented using

1. Compile time 🡪 a) function overloading b) operator overloading

2. [Run time](polymorphism_run_time.cpp) 🡪 virtual functions

[Operator overloading](polymorphism.cpp) 🡪 to specify more than one definition to a predefined operator. We can overload most of the build-in operators.

Abstraction

[Abstraction](virual_pure_virtual_functions.cpp) means only show relevant data and details rest of others are hide. This is the most important pillar in OOP. This is mostly done by*interfaces rather than abstract class*.

Eg.1 we use functions inside header files without knowing their functionality like pow, sine etc.

Eg.2 we have different cars but ignition button will start the car we don’t need to know the real implementation of the technology used inside.

Eg3. Let us assume we have a smartphone and it has a unique UI capable of operations, like to play song press play button and click on camera button to open camera. Now the new version of the same phone is used by the user which has newer set of advanced cameras better functioning ram and other new components but the UI remains the same so in this case the end user will not face any changes on the screen .

Then the new company arrives and sees that the user uses camera button to open camera hence it works with different fast algorithms but keeps the UI layer (abstraction layer) same and hence the complexity is unknown to the user and hence only the important data is known to the user.

This is implemented by using abstract classes. Abstract classes can have multiple pure virtual functions like camera() , music() , button\_actions() etc that are not implemented in this class but newer version of phones that inherit functionalities will have to implement all this in their own way and user will see no changes on his side.

It can be implemented using abstract classes and interfaces

🡪 Interface is a abstract class but it only contains purely virtual functions means all functions in it should be initialized to 0.

PURE VIRTUAL FUNCTIONS AND ABSTRACT CALSSES

VIRTUAL FUNCTIONS

🡪 A virtual function is defined in base class and redefined in derived class

🡪 It gives the ability of runtime polymorphism

🡪 It is created by adding virtual keyword at the start

Eg. **virtual** **void** func(){

//function body

}

If there is virtual function in base and derived class then the most derived class implementation of the function will work.

Pure Virtual functions and Abstract classes

A pure virtual function is a virtual function that must be overridden in the derived class else the derived class will become the base class.

Eg. virtual void chan(){

**virtual** **void** func() = 0;

}

🡪 A class is called abstract class if it has atleast one pure virtual function.

🡪 Sometimes implementation of all function cannot be provided in a base class because we don’t know the implementation.

🡪 Hence it make it compulsory for all derived class from the abstract class to have implementation of the pure virtual function else they too will become abstract class and this is not allowed.

ENCAPSULATION

🡪It is wrapping up data in a single unit .

🡪 combo of data hiding and abstraction.

🡪It says the data members (variables) should be kept private and member functions should be used to manipulate data by this way the access of data member is limited to those who have access of the member functions.

🡪 we can implement it using access specifiers.

🡪The process of implementing encapsulation can be sub-divided into two steps:

1. The data members should be labelled as private using the **private** access specifiers
2. The member function which manipulates the data members should be labelled as public using the **public**access specifier

Eg.1 for YouTube algorithm user has no access to the like count manipulation but can like or dislike which will change the value of total likes. Hence the data protection is covered.



Default copy constructor does shallow copy.

*// Copy Constructor  
Geeks Obj1(Obj);  
or  
Geeks Obj1 = Obj;*

*// Default assignment operator  
Geeks Obj2;  
Obj2 = Obj1;*

Depending upon the resources like dynamic memory held by the object, either we need to perform Shallow Copy or Deep Copy in order to create a replica of the object. In general, if the variables of an object have been dynamically allocated, then it is required to do a Deep Copy in order to create a copy of the object.

In shallow copy, an object is created by simply copying the data of all variables of the original object. This works well if none of the variables of the object are defined in the heap section of memory. If some variables are dynamically allocated memory from heap section, then the copied object variable will also reference the same memory location. This will create ambiguity and run-time errors, dangling pointer. Since both objects will reference to the same memory location, then change made by one will reflect those change in another object as well. Since we wanted to create a replica of the object, this purpose will not be filled by Shallow copy.

Des

The destructor is only one way to destroy the object created by the constructor.

destructor can-not be overloaded.

* It is automatically called when an object goes out of scope.
* Destructor release memory space occupied by the objects created by the constructor.
* In destructor, objects are destroyed in the reverse of an object creation.  The reason for reverse order is, an object created later may use the previously created object.

A a;

B b(a);

In the above code, the object ‘b’ (which is created after ‘a’), may use some members of ‘a’ internally. So, destruction of ‘a’ before ‘b’ may create problems. Therefore, object ‘b’ must be destroyed before ‘a’.

Destructors can be private.

If the object is created statically the destructor is automatically called but we need to call it manually in case of dynamically created.

We do this by delete keyword. Eg delete ob1;

Ie.  if the object is created by using new or the constructor uses new to allocate memory that resides in the heap memory or the free store, the destructor should use delete to free the memory.

We can declare constructor outside the class 🡪

<class-name> :: ~<class-name>() {

// some instructions

}

Static in C++

1. Static variable 🡪Variables in a function, Variables in a class.

When a variable is declared as static, space for it gets allocated for the lifetime of the program. Even if the function is called multiple times, space for the static variable is allocated only once and the value of the variable in the previous call gets carried through the next function call.

The static variables **in a class are shared by the objects.** There cannot be multiple copies of the same static variables for different objects. Also because of this reason static variables cannot be initialized using constructors.

A static variable inside a class should be initialized explicitly by the user using the class name and scope resolution operator outside the class using scope resolution operator(::).

Int class\_name :: var = 4;

1. Static member 🡪Class objects and Functions in a class.

scope of static objects is throughout the lifetime of the program.

 scope of normal object is inside the if block only where it is declared.eg. if an object is created inside a if condition then it lives till the scope of if.

The static variable in a class doesn’t belong to the object but to the class. Hence accessing a static member should be done as

Class\_name:: static\_var; not as object\_name.static\_var;

In case of static functions

We don’t depend on object to use these functions because same as in case of static members these functions belong to the class and we can acess them without creating objects of the class.

Also these functions can use only static variables(members).