

18 - CLASSES in C++

sexta-feira, 7 de fevereiro de 2025 07:01

[CLASSES in C++](#)

- OOP is a style on how to write your code
- C++ doesn't imply certain things but support it
- way to group data and functionalities together
- Variables made of class are called object variables
 - And a new object is an instance of that class
- Defining a class we define the visibility of the variables and functions
 - By default the visibility is private, need to specify as public to access or protected
- Functions inside classes are called methods
- USEFUL TO GROUP THINGS TOGETHER AND ADD FUNCTIONALITIES TO THE OBJECT



CLASSES vs STRUCTS in C++

- Kind of a similar one
- there is no much difference
- the main difference is the visibility options in structures (private, public, protected)
 - Class is private by default
 - struct the default is public
- But this is technically, but the use in code may differ
- struct exists by backward compatibility with previous versions
 - the compiler wouldn't know what it was in old codes
- The usage differs
 - That is no right or wrong answer, differ by opinion
- struct used just to represent variables
- Never use a structure with inheritance, go to classes

How to Write a C++ Class

- Log class to manage the log messages, used for debug process
- console is like an information dump
- Defined simple functions, member variables (public and private)
- Instantiated in main and also used the public functions

Static in C++

- 2 meanings,
 - outside of a class
 - Linkage of that symbol will be internal, only visible to that translation unit that you are working with (translation unit = file)
 - Inside of a class
 - All instances of that class will share the same memory, will only be one instance of that static variable across all instances of the class
- Focus on static outside of a class

Static for Classes and Structs in C++

- If used with a variable
 - Only one instance of that variable across all instances of that class
 - If one of the entity changes that variable, it'll affect all other instances
 - Better to update the value by its class than instance
 - By instance could cause confusion and bugs
- Static method
 - Don't have access to the class instance
 - call without a class instance
 - cannot write code that refer to a class instance

```
struct StaticEntity22
{
    static int x22, y22;

    void Print(){
        std::cout << "Entity 22 x22 " << x22 << " y22 " << y22 << std::endl;
    }
};

// When making this variables static, we need to initialize
// them without any instantiation
int StaticEntity22::x22;
int StaticEntity22::y22;

int main()
{
    StaticEntity22 se22;
    se22.x22 = 2;
    se22.y22 = 3;
    se22.Print();
}
```

```
94
95     StaticEntity22 se22;
96     se22.x22 = 2;
97     se22.y22 = 3;
98     se22.Print();
99
100     // StaticEntity22 se22_2 = {5, 8}; // This would fail for static
101     StaticEntity22 se22_2; // This would fail for static classes
102     // se22_2.x22 = 5;
103     // se22_2.y22 = 8;
104     se22_2.Print(); // result should be the same as the other instance
105
106     // we can access them by the class and not by the instance
107     // And it'll change its value
108     StaticEntity22::x22 = 5;
109     StaticEntity22::y22 = 8;
110     StaticEntity22::Print(); // 5, 8
111
112     // Not static struct parameters
113     Entity22 e22;
114
115     e22.x22 = 2;
116     e22.y22 = 3;
117     e22.Print();
118
119     Entity22 e22_2 = { 5, 8 };
120
121     e22_2.Print();
122
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

Hey
Hey
Hey

```
StaticEntity22 se22;
e22.x22 = 2;
e22.y22 = 3;
e22.Print();

// StaticEntity22 se22_2 = {5, 8}; // This would fail for static classes
StaticEntity22 se22_2; // This would fail for static classes
e22.x22 = 5;
e22.y22 = 8;
e22.Print();
```

```
Hey
Hey
Hey

Hey
root@aee12d748e6b:/src/Dev/HelloWorld/out/build# ./HelloWorld
Static Entity 22 x22 2 Y22 3
Static Entity 22 x22 2 Y22 3
Static Entity 22 x22 5 Y22 8
Entity 22 x22 2 Y22 3
Entity 22 x22 5 Y22 8
```

Can access a non-static variable within a class, t generates no error

```
struct StaticEntity22
{
    // static int x22, y22;
    int x22, y22;

    static void Print(){
        std::cout << "Static Entity 22 x22 " << x22 << " Y22 " << y22 << std::endl;
    }
};
```

Constructors in C++

- Special type of method that runs each time we instantiate an object
- When we instantiate a class without initializing the parameters, there is no actual value and they would receive garbage
- To declare it, there is no return type and needs to match the name of the class
 - Can optionally give parameters
- Has to manually initialize the primitive values, otherwise it'll get garbage in C++
 - Other languages may have different behaviours
- We can write as many constructors as we want, but with different parameters to have different signatures
- We can define a class with static properties and methods, and don't want to instantiate anything (no constructors)
 - <Class Name>() = delete;

Destructors in C++

- Even twin, the destructor kkk
- Called every time when destroy an object
 - Usually free and uninitialized and clean memory that will not be used anymore
 - If initialized objects with new, the destructor will delete them
- Destroyed in the end of the scope... if in a function, will be destroyed when leaving the function
- Used to delete memory allocation, in the heap for example... or any other initialization
- But is not very common

Inheritance in C++

- Allow us to have a hierarchy of classes that relates with each other
- Create subclasses from a parent class
- Avoid code duplication
 - Put duplicated code into a base class
 - So we don't need to keep implementing that
- Polymorphism is the idea of having multiple types of a single type
 - We can use a subclass whenever we want to use the base class
- The subclass always has everything that the base class has
- Used all the time to extend an existing class
 - Separate responsibilities

Virtual Functions in C++

- Allow us to override methods in our derived method
- If created a virtual function in class A, we have the option to override them in the base class B
 - To do something else
- Virtual functions introduce something called dynamic dispatch
 - Based on a V table for all the virtual functions, so we can match to the correct function we desired
 - If you want to override a function, you need to mock the base function
- If not declared as virtual, the function is associated with the class itself, if we call a method from a base class, the base class behavior will prevail. If virtual is defined, the vtable will determine the correct function to use based on the object calling and not just the class
- Maybe costly but the impact is minimal, don't worry

Interfaces in C++ (Pure Virtual Functions)

- Pure virtual functions

```
172 void PrintFunction26()
173 {
174     // Object will be created and deleted within this function
175     // Deleted when leaving it
176     Entity24 e24_6(10, 11);
177     e24_6.Print();
178 }
179
180 int main()
181 {
182
183     PrintFunction26();
```

```
Destroyed Entity!
Destroyed Entity!
Destroyed Entity!
root@aee12d748e6b:/src/Dev/HelloWorld/out/build# ./HelloWorld
Created Entity!
Entity 24 x24 10 Y24 11
Destroyed Entity!
Created Entity!
```

```
219 class Entity27
220 {
221 public:
222     // generate a v table for this function so if it is overridden,
223     // we can call the proper function
224     virtual std::string GetName() {return "Entity27"; }
225 };
226
227 class EntitySub27 : public Entity27
228 {
229 private:
230     std::string m_Name;
231 public:
232     EntitySub27(const std::string& name)
233         : m_Name(name) {}
234
235     std::string GetName() {return m_Name; }
236 };
237
238
239
240
241 int main()
242 {
243     Entity27* e27 = new Entity27();
244     std::cout << e27->GetName() << std::endl;
245
246     EntitySub27* esub27 = new EntitySub27("Hugo");
247     std::cout << esub27->GetName() << std::endl;
248
249     // things starts to crac, this is actually a player but references the base cl
250     // So we get the base class method output
251     Entity27* e27_2 = esub27;
252     // If not virtual function output is "entity27"
253     // If virtual function output is "hugo"
254     std::cout << e27_2->GetName() << std::endl;
255
256     std::cout << "" << std::endl;
257 }
```

- Define a function in the base class that doesn't have an implementation
- Force subclasses to actually implement that function
 - No base method definitions, implementation in the inherited class is not optional
- Need to define the method as virtual and add a =0 to the end of the declaration, instead of the function body
 - Also, we can't instantiate that class

Visibility in C++

- Belongs to OO
- How visible some members or methods under a class are
- No effect on how things run or in the performance,
- Just exist to write better code and organize things
 - Private
 - Only this class and friend ones can access this class members and methods
 - Protected
 - The base class and all the inheritance classes can access the members and methods
 - But outside of that chain is not possible
 - Public
 - Anyone can access the members and methods, no need to be in the chain of inheritance
- Where to use
 - Idea for been a developer and write code
 - code easy to maintain and understand
 - For other people to extend the code as well
 - what can be used or not
 - If something as private, the developer shouldn't access this by another class. Increase maintainability
 - If I never used a class and got some members as private, I know that I can't access or call it directly

Member Initializer Lists in C++ (Constructor Initializer List)

- Way to initialize out member from a class in a constructor
- Some way to initialize those members
- We use to do this for make it easier to write code, and make it cleaner and make it easier to read
- If we write code as before, assign them as variable. The constructor will be defined twice
 - One with the default one, with the initializer list
 - And another with the implementation assigning parameters to member variables (as usual)
 - We ended up creating 2 entities
 - In this case will waste performance

```
class Entity35
{
private:
    int m_Score; // Defined out of other, compared to the initialization
    std::string m_Name;

public:
    // same thing...
    // Some compilers will complain about this out of other thing
    // Always initialize them in the same order as we declare it
    // Entity35() : m_Name("Unknown"), m_Score(0) {}
    Entity35() : m_Score(0), m_Name("Unknown") {}
    // {
    //     m_Name = "Unknown";
    // }

    Entity35(const std::string& name) : m_Score(0), m_Name(name) {}
    // {
    //     m_Name = name;
    // }
    const std::string& GetName() const {return m_Name; }
};

int main()
{
    Entity35 e35;
    std::cout << e35.GetName() << std::endl;

    Entity35 e35_2("Hugo");
    std::cout << e35_2.GetName() << std::endl;

    std::cout << "" << std::endl;
}
```

How to CREATE/INSTANTIATE OBJECTS in C++

- When we create a class and comes the time to use it, we usually need to instantiate it
- We have two choices, and the difference is where the memory comes from. Which memory we are going to create the object in
 - Stack
 - The lifetime is defined by the scope it's in. When the scope ends, the memory gets free

- Heap
 - Once we allocate something there, it's gonna sit there until the end of the program
 - Only use heap if the object is really really big or if you need outside of the scope
 - This can cause memory leak... and this is bad...

```

418
419 int main()
420 {
421     // Create on the stack
422
423     // stack is usually small... if large objects are large... we need to allocate on the heap
424     // call the default constructor ( we need to have it)
425     // If we can create objects like this.... do it...
426     // It's the fastest way and also the most reliable in terms of memory management
427     // Beause it'll be deallocated in the end of the scope
428     Entity37 entity37;
429     Entity37 entity37_2("Hugo");
430
431     std::cout << entity37.GetName() << std::endl;
432     std::cout << entity37_2.GetName() << std::endl;
433
434     // We can't do that if we want to make the instance to live outside of that particular scope
435     // Scopes can also be if statements
436     Function();
437
438     Entity37* e37;
439     {
440         Entity37 entity37_4("Hugo inside the scope");
441         e37 = &entity37_4;
442         std::cout << entity37_4.GetName() << std::endl;
443     }
444     // This entity37_4 is gone.... appears trash in the console... interesting...
445     std::cout << e37->GetName() << std::endl;
446
447     Entity37* e37_2;
448     {
449         // Returns the location on the heap where is object is actually allocated
450         Entity37* entity37_5 = new Entity37("Hugo inside the 2 scope");
451         e37_2 = entity37_5;
452         std::cout << entity37_5->GetName() << std::endl;
453     }
454     // The same content is in the e37_2... kept for outside the scope ( when we know the location from the heap)
455     std::cout << e37_2->GetName() << std::endl;
456     delete e37_2;
457     std::cout << e37_2->GetName() << std::endl;
458
459
460

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

○ root@aee12d748e6b:/src/Dev/HelloWorld/out/build#
 ○ root@aee12d748e6b:/src/Dev/HelloWorld/out/build#
 ● root@aee12d748e6b:/src/Dev/HelloWorld/out/build# ./HelloWorld
 Unknown
 Hugo
 Hugo inside the scope
 Hugo inside the 2 scope
 Hugo inside the 2 scope
 5

-- Generating done
 -- Build files have been generated.
 Consolidate compiler options into target-specific flags.
 [20%] Building CXX object CMakeFiles/HelloWorld.dir src/main.cpp.o
 [40%] Linking CXX executable HelloWorld
 [100%] Built target HelloWorld
 ● root@aee12d748e6b:
 -- Source directory: /src/Dev/HelloWorld
 -- Configuring done
 -- Generating done
 -- Build files have been generated.