18 - CLASSES in C++

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

- OOP is a styla on how to write your code
- C++ doesn't mipl certain impies but support it
- way to grudata and functionalites together
- Variables made of class are called object variables
    - And a new object is na instance of that class
- Defining a class we define the visibility of the variables and functions
    - By default the visibility is private, need to specif as public to acces or protected
- Fuctions inside classes are called methods
- USEFULL TO GROUP THINGS TOGETHER AND ADD FUNCTIONALITIES TO THE OBJECT

[CLASSES in C++]

CLASSES vs STRUCTS in C++

- Kind a similar one
- there is no much difference
- the main diference is the visibility options in structures (private, public, protected
    - Class is private by default
    - struct the default is public
- But this is tecnicly, but the use in code may differ
- struct exists by bacward compatibility with previous versions
    - the ompiler wouldn't know wht 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 inherence, go to classes

How to Write a C++ Class

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

Static in C++

- 2 meanins,
    - outside of a class
        - Linkage of that symbel will be internal, only visible to that transation 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 taht variable across al isntances of that class
    - If one of the entity changes taht variable, it'll affect all other instances
    - Better to update the value by it's class than instance
        - By isntance could cause confusion and bugs
- Static method
    - Don't have access to the class instance
    - call without a class instance
    - canno write code that refer to a class instance

```cpp
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 ot initialize
// them without any instantiation
int StaticEntity22::x22;
int StaticEntity22::y22;

int main()
{

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

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

    // StaticEntity22 se22_2 = {5, 8}; // This would fail for stati
    StaticEntity22 se22_2; // This would fail for static classes
    // se22_2.x22 = 5;
    // se22_2.y22 = 8;
    se22_2.Print(); // result should be the same as the other insta

    // we can access them by the class and not by the instance
    // And it'll change its value
    StaticEntity22::x22 = 5;
    StaticEntity22::y22 = 8;
    StaticEntity22::Print(); // 5, 8

    // Not static struct parameters
    Entity22 e22;

    e22.x22 = 2;
    e22.y22 = 3;
    e22.Print();

    Entity22 e22_2 = { 5, 8 };

    e22_2.Print();
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
Hey
Hey
Hey
```

```cpp
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 na error

```cpp
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 na 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 ptionally give parameters
- Has to manually initialize the primitive values, otherwise i'll get garbages in c++
  - Other languages may have different behaviours
- We can write as much constructors as we want, but with different parameters to have diferent signatures
- can defien class with static propertis and methods, and don't want to instantiate nothing (no constructores
  - <Class Name>() = delete;

Destructors in C++

- evel twin, the destructor kkk
- call every time when destroy na object
  - Usually free and uninitialize and clean memory that will not use anymore
  - If initialized objects with new, te 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 alocation, in the heap for example.... or any other initialization
- But is not very commun

```cpp
void PrintFunction26()
{
    // Object will be created and deleted within this function
    // Deleted when leaving it
    Entity24 e24_6(10, 11);
    e24_6.Print();
}

int main()
{

    PrintFunction26();
```

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS

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!
```

Inheritance in C++

- Allow us to have iherache 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
- polymorfism is the idea of having multiple types of a single type
  - We can use a sub class whenever we want to use the base class
- the subclass always have everything that the base class have
- Used all the time to extend na 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
- Vitual functions introduce something called dynamic dispatch
  - Based a V table for all the virtual functions, so we can match to the correct function we desired
  - If you want to oberride a function, ou need to mock the base function
- If not declared as virtual, the function s associated with the class itself, if we call a method from a base class, the base class behvior will prevail. If virtual is defined, the vtable will determine the correct functon to use based on the object calling and not just the class
- Maybe costly but the impact is minimal, don't worrie

```cpp
class Entity27
{
    public:
        // generate a v table for this function so if it is overriden,
        // we can call the proper function
        virtual std::string GetName() {return "Entity27"; }
};

class EntitySub27 : public Entity27
{
    private:
        std::string m_Name;
    public:
        EntitySub27(const std::string& name)
        : m_Name(name) {}

        std::string GetName() {return m_Name; }
};



int main()
{
    Entity27* e27 = new Entity27();
    std::cout << e27->GetName() << std::endl;

    EntitySub27* esub27 = new EntitySub27("Hugo");
    std::cout << esub27->GetName() << std::endl;

    // things starts to crac, this is actually a player but references the base cl
    // So we get the base class method output
    Entity27* e27_2 =esub27;
    // if not virtual function output is "entity27"
    // if virtual function output is "hugo"
    std::cout << e27_2->GetName() << std::endl;


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

Interfaces in C++ (Pure Virtual Functions)

- Pure virtual functions

- Define a funtion in the base class that doesn't have na implementation
- force subclasses to actually implement that function
    - No base method definitions, implement in the inhered 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 efect on how things run or in the performance,
- Just exist to write better code and organize things
    - Private
        - Only this calss and riend ones can access this classmembers and methods
    - Protected
        - The base class and all the inherence classes can access the members and methos
        - But outside of that chain is not possible
    - Public
        - Anyone can access the members and methods, no need to be in the chain of inherence
- Where to use
    - Idea for been a developer and write code
    - code easy to maintanin and understand
    - For other people to extend the code as well
    - what can be used or not
    - If something as private, the developer shoudn't acess this by another class. Increase mantanability
        - If i never used a class and got some members as private, i know that i can't acess 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 ready
- If write code as before, assignin them as variable. The constructor will be defined twice
    - One with the defailt one, with the initializer list
    - And another with the implementation assigning paramemters to member variables (as usual)
    - We ended up creating 2 entities
    - In this case will wast erformance

```
class Entity35
{
    private:

        int m_Score; // Defined out of orther, 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 orther 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 needs to instantiate it
- We have two choices, and the diference is where the memory comes from. which memory we are going to create the object in
    - Stack
        - The lefetime 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 ther untill 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...

```cpp
int main()
{

    // Create on the stack

    // stack is usually small... if large objects are large... we need to allocate on the heap
    // call the default constructor ( we need to have it)
    // If we can create objects like this.... do it...
        // It's the fastest way and also the most reliable in therms of memory management
            // Beause it'll be dealocated in the end of the scope
    Entity37 entity37;
    Entity37 entity37_2("Hugo");

    std::cout << entity37.GetName() << std::endl;
    std::cout << entity37_2.GetName() << std::endl;

    // We can't do that if we want to make the instance to ive outside of that particular scope
        // Scopes an also be if statements
    Function();

    Entity37* e37;
    {
        Entity37 entity37_4("Hugo inside the scope");
        e37 = &entity37_4;
        std::cout << entity37_4.GetName() << std::endl;
    }
    // This entity37_4 is gone.... apears trash in the console... intresting...
    std::cout << e37->GetName() << std::endl;

    Entity37* e37_2;
    {
        // Returns the location on the heap where is object is actually allocated
        Entity37* entity37_5 = new Entity37("Hugo inside the 2 scope");
        e37_2 = entity37_5;
        std::cout << entity37_5->GetName() << std::endl;
    }
    // The same content is in the e37_2... kept for outside the scope ( when we know the locaton from the heal)
    std::cout << e37_2->GetName() << std::endl;
    delete e37_2;
    std::cout << e37_2->GetName() << std::endl;

```

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
�TA�U���L�D�scope
Hugo inside the 2 scope
Hugo inside the 2 scope

5
```

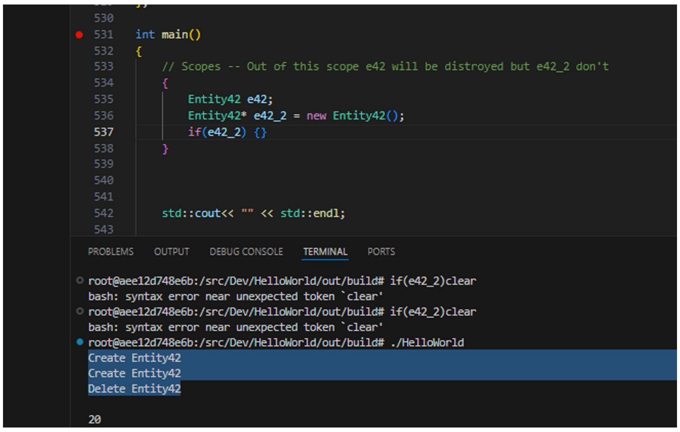
```
-- Generating done
-- Build files hav
Consolidate compil
[ 20%] Building CX
[ 40%] Linking CXX
[100%] Built targe
root@aee12d748e6b:
-- Source director
-- Configuring don
-- Generating done
-- Build files hav
```

The NEW Keyword in C++

- Programing in c++ you need to cary about performance
- Understand new is very important
- Not equal to c# or java
- new allocate memor in the heap, as the necessary size in bytes of memory
    - needs to find a dress in memory that has the amount of bytes in a roll and return
- When call new, it takes time
- give us a pointer to that memory in the heap

```cpp
int a38 = 2; // stack
int* b38 = new int; // 4 bytes allocated in the heap ( returns a pointer )
int* c38 = new int[5]; // 200 bytes allocated in the heap ( returns a pointer )

// new actualy call the malloc(50) function -> which llocate the amount of memory we need and return a pointer
// The only difference is that using new we call the class constructor
// Allocate the memory and call the constructor
Entity37* e37_38 = new Entity37();// pointer to a pointer of the heap memory

// It's an operator and we can overload it

if(a38) std::cout << a38 << std::endl;
if(b38) std::cout << b38 << std::endl;
if(c38) std::cout << c38 << std::endl;

// when use the new keyword, we need to delete to free the heap memory
// Memory is not automatcli released
delete b38;
delete[] c38;
delete e37_38;


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

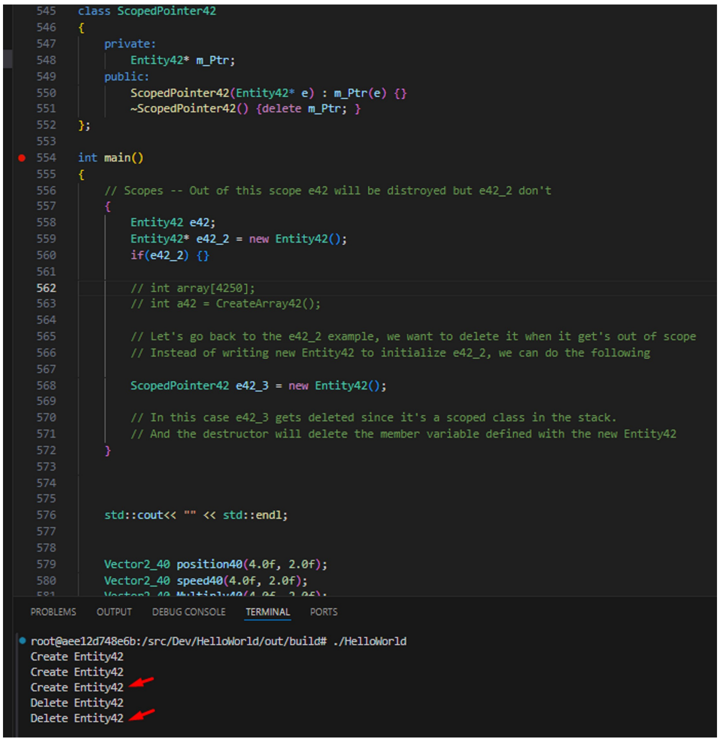
Object Lifetime in C++ (Stack/Scope Lifetimes)



```cpp
int main()
{
    // Scopes -- Out of this scope e42 will be distroyed but e42_2 don't
    {
        Entity42 e42;
        Entity42* e42_2 = new Entity42();
        if(e42_2) {}
    }



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

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS

root@aee12d748e6b:/src/Dev/HelloWorld/out/build# if(e42_2)clear
bash: syntax error near unexpected token `clear'
root@aee12d748e6b:/src/Dev/HelloWorld/out/build# if(e42_2)clear
bash: syntax error near unexpected token `clear'
root@aee12d748e6b:/src/Dev/HelloWorld/out/build# ./HelloWorld
Create Entity42
Create Entity42
Delete Entity42

20
```

- How objects lives in stack
- Can be seena s a data structure that we can stack thisngs on top of each other
- We need to go line by line to access something in the medlie
- Scope can be anything from a function scope, if statement scope, for loop, or empty scope
  - We also have a scope for classes
- In a stack, we something goes out of scope, the stack deletes it



```cpp
class ScopedPointer42
{
    private:
        Entity42* m_Ptr;
    public:
        ScopedPointer42(Entity42* e) : m_Ptr(e) {}
        ~ScopedPointer42() {delete m_Ptr; }
};

int main()
{
    // Scopes -- Out of this scope e42 will be distroyed but e42_2 don't
    {
        Entity42 e42;
        Entity42* e42_2 = new Entity42();
        if(e42_2) {}

        // int array[4250];
        // int a42 = CreateArray42();

        // Let's go back to the e42_2 example, we want to delete it when it get's out of scope
        // Instead of writing new Entity42 to initialize e42_2, we can do the following

        ScopedPointer42 e42_3 = new Entity42();

        // In this case e42_3 gets deleted since it's a scoped class in the stack.
        // And the destructor will delete the member variable defined with the new Entity42
    }



    std::cout<< "" << std::endl;


    Vector2_40 position40(4.0f, 2.0f);
    Vector2_40 speed40(4.0f, 2.0f);
```

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS

root@aee12d748e6b:/src/Dev/HelloWorld/out/build# ./HelloWorld
Create Entity42
Create Entity42
Create Entity42
Delete Entity42
Delete Entity42
```

Local Static in C++

- Defining static variabes in loca
- Allows us to define variable that has a lifetime of our entire program
  - However, the scoe is within this function only
  - Any scope
- We can achieve by other means, but this is ver usefull

```cpp
class Singleton48
{
    private:
        static Singleton48* s_Instance;
    public:
        static Singleton48& Get() { return *s_Instance; }

        void Hello();

};

// Exaclity the same behavior but using the static instance instead of the privat definition
// Due to the static instance definition, the object doesn't get destroed and every time we call the
// Singleton instance, we get the same one. Ensuring the singleton properties
class Singleton48_2
{
    public:
        static Singleton48& Get()
        {
            static Singleton48_2 instance;
            return instance;
        }

        void Hello();

};

Singleton48* Singleton48::s_Instance = nullptr;

int main()
{
    // The same, one with privet constructor and another with static instance
    Singleton48::Get.Hello()
    Singleton48_2::Get.Hello()

    // When i cal Function48 for the first time, this variable (i48) will be initialized to zero
    // And in subsequential calls to this function, this variable will not be initialized again
    // The lifetime of that variable will last the hole program but only inside that scope
    Function48();
    Function48();
    Function48();
    Function48();
    Function48();

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