The C++

Outline

- Constructor and Overloading
- Namespaces
- Inheritance
- Wrapper Class
- Templates
- Class Relation
- static, const
- friend function

Constructor and Overloading

Default Constructor: A constructor that takes no arguments. It is invoked when an object is created without any arguments.

Parameterized Constructor: A constructor that takes parameters to initialize the object's data members.

Copy Constructor: A constructor that initializes an object using another object of the same class. It is invoked when an object is passed by value or when an object is initialized with another object of the same type.

Move Constructor: Introduced in C++11, a move constructor is used to initialize an object from an rvalue reference. It allows for more efficient transfer of resources (such as memory) from one object to another.

Constructor and Overloading

Shallow Copy	Deep Copy
MyString(const MyString& other) : std::string(other) {}	// Deep copy constructor MyString(const MyString& other, bool deepCopy) : std::string(deepCopy ? other : "") {}

Namespaces

Preventing Naming Conflicts: Avoids clashes between identifiers in different parts of the program.

Logical Organization: Allows grouping of related code elements (classes, functions, variables) under a single name.

Readability and Clarity: Makes code more understandable by indicating the purpose or domain of the elements.

Ambiguity Resolution: Resolves naming ambiguities, especially when integrating libraries or modules.

Versioning and Maintenance: Facilitates maintenance and updates by isolating changes within specific namespaces.

Inheritance and Access Specifier

Inheritance is a fundamental feature of object-oriented programming (OOP) in C++. It allows a class (called the **derived** class or subclass) to inherit properties and behaviors from another class (called the base class or **superclass**).

Public: Members are accessible to everyone, including derived classes and objects outside the class hierarchy.

Protected: Members are accessible to derived classes and their own methods, but not to objects outside the class hierarchy.

Private: Members are only accessible within the class they are declared in; neither derived classes nor objects outside the class hierarchy can access them.

Namespaces and Inheritance

```
namespace mystringwrapper {

    class string : public std::string {
    public:

....
```

Wrapper Class

In C++, a wrapper class provides additional functionality, abstraction, or control over its usage.

It's often used to add features like error handling, memory management, or interface standardization.

size_t **find**(const MyString& substr) const;

std::vector<MyString> **split**(char delimiter) const;

MyString& **concat**(const MyString& str);

...

Templates

Templates in C++ provide a way to write generic functions or classes that can work with any data type. They allow you to define functions or classes without specifying the exact data types they will operate on. Instead, you use placeholders (typically typename or class) to represent generic types.

```
template <class T>
class Node {
private:
    T element;
    Node<T>* next_node;
    Node<T>* prev_node;
...
```

Class Relation :: Aggregation

Aggregation is a special form of association, where one class (the whole) has a relationship with another class (the part), but the part can exist independently of the whole.

In C++, aggregation is typically implemented by having a pointer or reference to another class.

Aggregation implies a weaker relationship compared to composition. The lifetime of the aggregated object is not dependent on the lifetime of the container object.

Class Relation :: Aggregation

```
template <class T>
class list {
private:
    Node<T>* list_head;
    Node<T>* list_tail;
....
```

Static and const

Static:

- When used inside a class, static keyword is used to declare class-level variables and functions. These variables and functions belong to the class itself, rather than to instances of the class.
- Static variables are shared among all instances of the class. They are initialized once (usually at the start of the program) and retain their value until the program ends.
- Static member functions can be called without needing an instance of the class.
- Static variables and functions can be accessed using the scope resolution operator :: or through an instance of the class.

Const:

- The const keyword in C++ is used to declare constants, i.e., variables whose value cannot be modified after initialization.
- It can be applied to variables, pointers, and member functions.
- When used with a pointer, it specifies that the pointer itself cannot be changed to point to another address.
- When used with a member function, it indicates that the function does not modify the state of the object on which it is called.

Static and const

Static	Const

template <class T>
class list {
private:
  Node<T>* list_head;
  Node<T>* list_tail;
  static int size;
…
 	
// Initialize the static member variable size
template<class T>
int list<T>::size = 0;
…
	// Function to return the head of the list
	Node<T>* Head() **const**;
	
	// Function to return the tail of the list
	Node<T>* Tail() **const**;
	
	// Function to return the last element of the list
	T back() **const**;
	
	// Overloaded subscript operator to access elements by index
	T operator[](int index) **const**;

friend function

- Allow access to private members of a class from outside.
- Break encapsulation in controlled situations.
- Facilitate operator overloading for user-defined types.
- Provide efficient access to private members without accessor methods.
- Offer flexibility in class relationships and design.
- Reduce coupling between classes by restricting access only to necessary components.

```
friend std::ostream& operator<<(std::ostream& os, const MyString& str);
```

YouTube

https://youtu.be/3NTEEjs0DY8