C++ Lab HW1 Ans By 1123527 陳淇祿

1.

(1)Encapsulation

(2)Inheritance

(3)Polymorphism

(4)Abstraction

2.

(1) The preprocessor is a phase of compilation in many programming languages, including C and C++, where directives prefixed with "#" are processed before actual compilation begins. These directives can include macro definitions, file inclusions, and conditional compilation statements. The preprocessor manipulates the source code before it's compiled, enabling tasks such as code reuse, conditional compilation, and debugging aids.

(2) A namespace is a feature in many programming languages, including C++ and C#, that organizes code elements into distinct scopes to prevent naming conflicts and to make code more modular and readable. Namespaces provide a way to group related classes, functions, variables, etc., under a unique identifier, ensuring that identifiers within one namespace don't clash with those in another. This helps in managing large codebases and facilitates code reuse and maintenance.

3. Inheritance in Object-Oriented Programming (OOP) allows a class to inherit properties and behaviors (methods) from another class, known as the base class or parent class. The class that inherits these properties and behaviors is called the derived class or child class. This concept promotes code reusability and establishes a hierarchical relationship among classes.

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

using namespace std;

int *factorial*(int input) {

    int ans = 1;

*for* (int i = 1; i <= input; i++) {

        ans \*= i;

    }

*return* ans;

}

int *main*() {

    int num = 0;

    cin *>>* num;

    cout *<<* *factorial*(num);

*return* 0;

}

5.(1) if Statement: The if statement allows for conditional execution of code based on whether a specified condition evaluates to true or false. It can be followed by an optional else statement for executing code when the condition is false.

(2)switch Statement: The switch statement provides a way to execute different blocks of code based on the value of a variable or expression. It contains multiple case labels that match specific values, along with an optional default case for handling values that don't match any of the specified cases.

(3)while Loop: The while loop repeatedly executes a block of code as long as a specified condition remains true. It evaluates the condition before each iteration, and if the condition is false initially, the loop body is not executed at all.

(4)for Loop: The for loop is used for iterating over a range of values or performing a specific number of iterations. It consists of three parts: initialization, condition, and iteration expression, enclosed within parentheses. The loop body is executed repeatedly until the condition becomes false.

(5)do-while Loop: The do-while loop is similar to the while loop, but it guarantees that the loop body is executed at least once before checking the loop condition. It evaluates the loop condition at the end of each iteration.

6.(1) &&:AND

(2)||:OR

(3)!:NOT

7.loop inside loop

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

using namespace std;

int *main*(){

*for*(int i = 1; i <= 3; i++){

*for*(int j = 1; j <= 3; j++){

            cout *<<* "(" *<<* i *<<* "," *<<* j *<<*")  ";

        }

    }

}

8. Class in C++ is a user-defined data type that serves as a blueprint for creating objects. It encapsulates data for the object and functions to manipulate that data.

The two primary components of a class are:

Data members: These are variables declared within the class, representing the attributes or properties of objects created from the class.

Member functions: Also known as methods, these are functions defined within the class to perform operations on the data members or to provide functionality related to the class.

9.

An object in C++ is an instance of a class. It's a concrete entity that represents a specific instantiation of the class blueprint. Objects have properties (data members) and behaviors (member functions) defined by the class. When a class is instantiated, memory is allocated to store the object's data members, and the object can then be used to interact with the program.

Access specifiers in C++ are keywords used to define the accessibility of class members (data members and member functions) from outside the class. There are three access specifiers in C++:

Public: Members declared as public are accessible from outside the class through object instances. They can be accessed by any function, whether it's a member function of the class, a friend function, or a function defined outside the class.

Private: Members declared as private are not accessible from outside the class. They can only be accessed by member functions of the same class. Data encapsulation is achieved using private access specifiers, ensuring that sensitive data is not modified accidentally or inappropriately.

Protected: Members declared as protected are similar to private members but have one key difference: they are accessible in derived classes. Protected members can be accessed by member functions of the class and by member functions of derived classes. They are primarily used to implement inheritance and provide controlled access to derived classes.

Access specifiers play a crucial role in defining the interface and encapsulation of a class, determining how its members can be accessed and manipulated from outside the class scope. They help in enforcing data hiding and encapsulation principles, promoting robust and maintainable code.

10.Instance variables refer to the variables that each class instance (object) possesses in C++. These variables have values independent of other objects, with each object having its own set of instance variables that store the object's state information.

Static variables, on the other hand, are associated with the class itself rather than with individual instances of the class. Their values are shared among all instances of the class, even if no objects of the class are created. Static variables are typically used for sharing the same data across multiple objects or tracking global information specific to a class.

11. A function in programming is a block of code that performs a specific task or action. It typically takes input parameters, processes them, and produces output. Functions can be reusable units of code, allowing programmers to write modular and organized programs.

Functions are used in programming for several reasons:

1. **Modularity**: Functions allow code to be broken down into smaller, manageable pieces, making it easier to understand, maintain, and debug. Each function can focus on performing a specific task, promoting code reusability and reducing redundancy.
2. **Abstraction**: Functions abstract away implementation details, allowing programmers to focus on the high-level logic rather than the internal workings of the function. This simplifies the programming process and enhances code readability.
3. **Encapsulation**: Functions encapsulate related operations, grouping them together and providing a clear interface for interacting with them. This helps in organizing code and keeping related functionality together, leading to more organized and maintainable codebases.
4. **Code Reusability**: Functions can be called multiple times from different parts of a program or even from different programs. By defining a function once, it can be reused wherever needed, reducing the amount of code duplication and improving development efficiency.
5. **Ease of Testing**: Functions make it easier to test individual components of a program in isolation, allowing for more effective unit testing. Since functions encapsulate specific functionality, they can be tested independently, simplifying the testing process and enhancing code quality.

Overall, functions play a crucial role in programming by promoting code organization, reuse, abstraction, and encapsulation, ultimately leading to more efficient and maintainable software development.

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

using namespace std;

int *add*(int a,int b){

*return* a+b;

}

int *main*(){

    int num1 = 0,

        num2 = 0;

    cin *>>* num1 *>>* num2;

    cout *<<* *add*(num1,num2);

}

12.one have ”~” ,another don’t

Constructors and destructors are special member functions of a class in C++.

**Constructors**:

* Constructors are member functions that are automatically called when an object of the class is created.
* They are used to initialize the object's data members or perform any necessary setup operations.
* Constructors have the same name as the class and can be overloaded to accept different sets of parameters.
* If no constructor is explicitly defined in the class, C++ provides a default constructor, which initializes data members to default values.

**Destructors**:

* Destructors are member functions that are automatically called when an object is destroyed, typically when it goes out of scope or is explicitly deleted.
* They are used to release resources allocated by the object, such as dynamic memory, file handles, or network connections.
* Destructors have the same name as the class prefixed with a tilde (~), and they cannot accept any parameters.
* If no destructor is explicitly defined in the class, C++ provides a default destructor, which does nothing.

I’m not sure if “As program to call the constructor and destructor.” Means to write a program or not. So here’s the program

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

using namespace std;

class MyClass {

public:

*MyClass*() {

        cout *<<* "Constructor called" *<<* *endl*;

    }

*~MyClass*() {

        cout *<<* "Destructor called" *<<* *endl*;

    }

};

int *main*() {

    MyClass obj;

*return* 0;

}

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

using namespace std;

int *main* (){

    int k = 0,

        num = *pow*(1,k)+*pow*(5,k)+*pow*(3,k);

*while*(num != 153){

        num = *pow*(1,k)+*pow*(5,k)+*pow*(3,k);

*if*(num == 153){

            cout *<<* "yes " *<<* k *<<* *endl*;

*break*;

        }

        k++;

    }

}