1. Advantages of C Language

Ans:

1. C is portable.

2. C is efficient. It can interact with hardware efficiently.

3. C is flexible.

4. C is freely available. Wide variety of compiler available.

1. Any New Language Is Basically Design for Two Reasons

1. To overcome or avoid limitation of previous language.

2. To provide new feature which are not available on existing languages

1. Limitation C With Respect To C++

1. Language doesn’t provide security for data.

2. Using function can achieve code reusability but reusability is limited.

3. The programs are not extendible.

4. We can’t write function inside structure.

5. When program become complex, understanding and maintaining such

program is very difficult.

“Bjarne Stroustrup” design a new language C with classes in 1979 on DEC PD11

machine.

Reconstructed by ANSI on 1983

63 keywords are available on C++ language

27+5=32 ---------ANSI C

32(C) +31=63---------ANSI C++

Token in C:

Identifiers, keywords, operator, string, special symbol, constant

1. Difference between Procedure Oriented Language and Object Oriented Language

Procedure oriented language Object oriented language

1. Emphasis on steps or algorithm 1. Emphasis on data of the program

2. Programs are divided into small 2. Programs are divided into small

code units i.e. function data units i.e. classes

3. Most function shared global data 3. Data is hidden and not accessible

and can be modified. outside classes

4. Data move from function to 4. Objects communicate with each

Function other

5. Top down approach 5. Bottom up approach

1. Difference between Structure in C and C++

Ans: Structure in C

1. At the time of creating of structure variable, struct keyword is compulsory. Example: struct time t

2. By default all the members are accessible outside the structure. C languages doesn’t have concept of access specifier

Structure in C++

1. At the time of creating object of structure, struct keyword is not compulsory. Example: time t

2. By default all members of structure in C++ are public. We can make them private.

1. Access Specifiers in C++

Ans: By default all members in the structure are accessible in the program using dot and

arrow operator. Such access can be restricted by access specifier.

Private: Accessible only within the structure

Public: Accessible within and outside structure.

1. What is Function Overloading?

Function with same name but different in signature are called function overloading. Return type is not for function over loading. Function call is resolved according to the type of arguments passed.

A screenshot of a computer program

AI-generated content may be incorrect.

Function having same name but different either types of arguments, actual number

of arguments, ordered of arguments, such process of writing function is called

function overloading.

1. Function having same name but different in number of arguments

Example: void sum(int num1,int num2)

Void sum(int num1,int num2,int num3)

2. Function having same name and arguments but different in types of

arguments

Example: void sum(int num1,int num2)

void sum(int num1,float num2)

3. Function having same name and number of arguments but order of

arguments are different

Example: void sum(int num1,float num2)

void sum(float num1,int num2)

1. What is Name Mangling?

When we write function in C++, the compiler internally create unique name for

each and every function by looking towards name of function and type of

arguments pass that function. Such process of creating unique name is called name

mangling. That individual name is called mangled name.

Example: void sum(int num1,int num2)

sum@ num1,num2

void sum(int num1,int num2,int num3)

sum@ num1, num2, num3

**9.Class**

Building block that bind together data and code

Program is divided into different classes.

Class has

1. Variables(data member)

2. Functions(member function or method)

By default class member are private. Not accessible outside class.

Classes are standalone component and can be distributed inform of libraries.

Class is a blue print of an object.

1. What is Class, Object?
2. What is State, Behaviour, Identity, Data Member?
3. What is Constructor, Type of Constructor?
4. What is Destructor?
5. What is virtual Destructor, significant of it, Pure Virtual Destroctor?
6. What is Mutator, inspector & facilator?
7. What is constructor Constructor Overloading?
8. Why Constructor Doesn’t Have Any Return Values?
9. What is this pointer?
10. Can we do destructor overloading?
11. Why the size of empty object is 1 byte?
12. What is default argument?
13. What is Enum?
14. Why C++ is not pure object oriented language?
15. What is difference between function & Macro?
16. What is inline function?
17. What is anonymous object?
18. Allocating memory statically and dynamically?
19. Allocation of Memory for Array Statically
20. Allocating Memory Dynamically for an Array Using Malloc Function
21. Allocating Memory for a Single Variable Using new Operator
22. Allocating Memory for an Array Using new Operator
23. Dynamic Array of Object
24. Array of Pointer
25. Pointer to Pointer Object
26. Difference between malloc and new
27. Memory Leakage
28. Dangling Pointer
29. References
30. Swapping of Two Variable Using Call by Value Function, call by address, call by reference?
31. Difference between Pointer and Reference
32. Swapping of Two Number using 3rd variable, using + and \* operator?
33. Shallow Copy?
34. Deep Copy?
35. Condition for Deep Copy?
36. Copy Constructor?
37. Can a object be passed as value to the copy constructor
38. Operator Overloading
39. Overloading operator+ Function by Using Friend Function
40. What is the difference between Copy Constructor and Assignment Operator Function
41. Assignment Operator Overloading Function
42. List of Function that the Compiler provides by default to any Class if it is not available
43. Limitation of Operator Overloading
44. There are some functions which we can’t overload them as a friend function
45. Friend Function
46. Why Friend Function Doesn’t Have This Pointer?
47. Friend Class
48. Defining Class Member as a Friend into another Class
49. Defining Class as Friend into another Class
50. Static Variable
51. Size of the Object
52. Static Member Function
53. Static Member Function Doesn’t Have This Pointer
54. Member Initializer List
55. Constant Data Member
56. Constant Member Function
57. Principal of Object oriented language
58. Synonyms/ different Name of Compile and Runtime Polymorphism
59. Hierarchy
60. Composition
61. Types of Composition
62. Inheritance
63. Types of Inheritance
64. Diamond problem or Virtual Base Class or Virtual Inheritance
65. Mode of Inheritance
66. Object Slicing
67. Up Casting, Down casting
68. Virtual Function
69. Late Binding
70. Function Overriding
71. Difference between Function Overloading and Function Overriding
72. Virtual Function Table, VPTR
73. Pure Virtual Function
74. Template, Generic Programming Template
75. Casting Operator, significant of all casting operator?
76. Namespace, Sample Namespace Program
77. Exception Handling, Generic Catch Block
78. Run Time Type Information
79. Smart Pointer, difference between shared and unique
80. Does C++ class create padding similar to structure?
81. Does pure virtual function has entry in virtual function table
82. Virtual Function Pointer and Virtual Function Table
83. Constant object,
84. Do not throw an exception from a destructor
85. Can I throw an exception from a constructor? From a destructor?
86. Single tone Design Pattern, Factory Design Pattern, Proxy, Command

Single tone Design Pattern

#include <iostream>

using namespace std;

class Singleton {

   static Singleton \*instance;

   int data;

   // Private constructor so that no objects can be created.

   Singleton() {

      data = 0;

   }

   public:

  ~ Singleton() {

  }

   public:

   static Singleton \*getInstance() {

      if (!instance)

      instance = new Singleton;

      return instance;

   }

   int getData() {

      return this -> data;

   }

   void setData(int data) {

      this -> data = data;

   }

};

//Initialize pointer to zero so that it can be initialized in first call to getInstance

Singleton \*Singleton::instance = 0;

int main(){

   Singleton \*s = s->getInstance();

   //getInstance();

   cout << s->getData() << endl;

   s->setData(100);

   cout << s->getData() << endl;

   return 0;

}

Factory Design Pattern

#include <iostream>

using namespace std;

// Library classes

class Vehicle {

public:

    virtual void printVehicle() = 0;

};

class TwoWheeler : public Vehicle {

public:

    void printVehicle()

    {

        cout << "I am two wheeler" << endl;

    }

};

class FourWheeler : public Vehicle {

public:

    void printVehicle()

    {

        cout << "I am four wheeler" << endl;

    }

};

// Client (or user) class

class Client {

public:

    Client(int type)

    {

        // Client explicitly creates classes according to

        // type

        if (type == 1)

            pVehicle = new TwoWheeler();

        else if (type == 2)

            pVehicle = new FourWheeler();

        else

            pVehicle = NULL;

    }

    ~Client()

    {

        if (pVehicle) {

            delete pVehicle;

            pVehicle = NULL;

        }

    }

    Vehicle\* getVehicle() { return pVehicle; }

private:

    Vehicle\* pVehicle;

};

// Driver program

int main()

{

    Client\* pClient = new Client(1);

    Vehicle\* pVehicle = pClient->getVehicle();

    pVehicle->printVehicle();

    return 0;

}

Proxy

Command

#include <iostream>

// Receiver

class Receiver {

public:

    // Receiver class defines the action to be performed.

    void performAction()

    {

        std::cout << "Receiver is performing an action" << std::endl;

    }

};

// Command interface

class Command {

public:

    // The execute method is declared in the Command

    // interface.

    virtual void execute() = 0;

};

// Concrete Command

class ConcreteCommand : public Command {

private:

    Receiver &receiver;

public:

    // ConcreteCommand takes a reference to a Receiver

    // object in its constructor.

    ConcreteCommand(Receiver&rec)

        : receiver(rec)

    {

    }

    // The execute method calls the action on the Receiver.

    void execute() { receiver.performAction(); }

};

// Invoker

class Invoker {

private:

    Command\* command;

public:

    // The setCommand method allows setting the command to

    // be executed.

    void setCommand(Command\* cmd) { command = cmd; }

    // The executeCommand method triggers the execution of

    // the command.

    void executeCommand() { command->execute(); }

};

int main()

{

    // Create a Receiver instance.

    Receiver receiver;

    // Create a ConcreteCommand, passing the Receiver to it.

    ConcreteCommand command(receiver);

    // Create an Invoker.

    Invoker invoker;

    // Set the command to be executed by the invoker.

    invoker.setCommand(&command);

    // Execute the command.

    invoker.executeCommand();

    return 0;

}

1. Future and promise
2. Final Specifier
3. Can constructor be private
4. Class Tempale, function Template
5. Difference between composition & Inheritance
6. Interface in CPP
7. Diiference between vector & array, list & set
8. Explicit constructor in Cpp
9. Can Friend ship is mutual?
10. Function hiding in C++
11. Function overloading using Template
12. Class Template Copy Constructor
13. Insertion & Extraction operator overloading
14. Vector, Vector sort, List, Map, Multimap
15. Move constructor
16. Mutable
17. How to make a class non inheritable
18. Can we write protected or private constructor? Use case.
19. How to initialize reference?
20. Reference in class
21. RTTI class
22. Setw in Cpp
23. Solid principle in Cpp
24. Can static function be virtual, overloaded,
25. Can a static variable be const, volatile, const volatile
26. What is static member function?
27. Thread safety in singleton class
28. What is the size of virtual function?
29. Reverse a word