C++ / OOPS Viva Question & Answer

1. Who developed C++ language ?

Ans :- Bjarne Stroustrup

2. What is keyword in c++ language?

Ans :- Keyword are the word that have a predefined meaning . These cannot be redefined by the programmer .For Example :- Int , char , for

3. What are the rules of varible name in C++ ?

1.Valid characters: Variable names can contain letters (both uppercase and lowercase), digits, and underscores (_).
2.The first character: It must be a letter (uppercase or lowercase) or an underscore (_). It cannot start with a digit.
3.Case sensitivity: C++ is case-sensitive, so "myVariable" and "myvariable" are considered different names.
4.Reserved keywords: You cannot use C++ reserved keywords (e.g., int, float, if, else, etc.) as variable names.
5. No spaces: Variable names cannot contain spaces.
6. Length: Variable names can be of any length, but it's recommended to keep them concise and meaningful for readability.

4. What type of data main() function Return ?

Ans. Int

5. Name Three Unary operator ?

Ans :- ++ , -- ,sizeof

6. Name Two Entry Control Loops ?

Ans :- While and for

7. What is function Overloading?

Ans :- When a program has more than one function with same name but different parameters that is know as function overloading . ex . :- sum(int num , int num1) , sum(float num , float num1)

8. What is recursion ?

ANS :- When a function call itself that is known as recursion . For example factorial .

9. How amny storage classes in c++ ?

ANS :- 5. Auto , static , extern , register and mutable

10 .Name different types of in heritance in C++ ?

ANS :- 1. Single inheritance
2. Multiple inheritance
3. Multilevel inheritance
4. Hierarchical inheritance
5.Hybrid inheritance

11. What is class ?

Ans :- A class is blueprint of object . It is user defined data type . For example Fruit is a class.

12. What is encapsulation ?

Ans :- The Wrapping of data and function into a single is known as encapsulation.

13. Define data abstraction ?

ANS :- It represents the essential features and excludes the details or explanations .

Abstraction is most important features of OOPS.

14. Define Polymorphism?

ANS :- Poly Means many . Polymorphism is the ability to use operator and function

Different ways.

15. What is typedef ?

ANS :- It is used to create a another name to existing type .

For example :- typedef int num :

16. What is Object ?

Ans :- Object is created from class . Apple is object of class fruit .

17. What is use of strcmp() function ?

Ans :- It is used to compare two string . It return 0, If both are equal

and return a negative value if both are not equal.

Imp . 18. Difference between C and C++ .

Ans :-

C	C++
C is a procedural programming language.	C++ is a multi-paradigm language that supports procedural, object-oriented, and generic programming.
C does not support object-oriented programming. It lacks features like classes, objects, inheritance, and polymorphism	C++ is designed to support object-oriented programming, making it easier to model real-world entities using classes and objects.
C uses header files (.h) to declare functions and data structures.	C++ also uses header files, but it allows defining and declaring functions within class definitions.
C does not support function overloading, where multiple functions can have the same name but different parameters.	C++ supports function overloading, enabling developers to create functions with the same name but different parameter lists
Both C and C++ have pointers for memory manipulation.	C++ introduces references, which are safer and more convenient alternatives to pointers for certain scenarios.
C does not have a built-in Standard Template Library.	C++ includes the STL, providing a collection of data structures and algorithms that simplify common programming tasks.
C primarily relies on error codes or return values for error handling	C++ introduces exceptions, allowing more robust and structured error handling.

19. what is inline Functions ?

Ans :- inline function is a special type of function that the compiler may choose to insert directly into the calling code instead of generating a separate function call. This process is known as "inlining." The purpose of using inline functions is to improve the performance of code by reducing the overhead of function calls.

Ex :- inline int add(int a, int b) {

```
    return a + b;
}
```

20. What is pointer ?

Ans :- In C++, a pointer is a special variable that stores the memory address of another variable. Pointers allow you to work directly with memory locations and manipulate data indirectly, providing a powerful feature for advanced memory management and dynamic data structures. For example:-

data_type* pointer_name;

21. What is class(Write a program) ?

Ans :- class is a user-defined data type that serves as a blueprint for creating objects. It encapsulates data (attributes) and functions (methods) that operate on that data, allowing you to model real-world entities with their behaviors and characteristics.

example of a class in C++:

```
class Dog {
public:
  string name;
  string breed;
  void bark() {
    cout << "Woof!" << endl;
  }
};
```

This class defines a dog object. The class has two properties: `name` and `breed`, and it has one behavior: `bark()`.

```
#include <iostream>

using namespace std;

class Dog {
public:
  string name;
  string breed;
  void bark() {
    cout << "Woof!" << endl;
  }
};

int main() {
  Dog my_dog;
  my_dog.name = "Spot";
  my_dog.breed = "Golden Retriever";
  my_dog.bark();

  return 0;
}
```

This program creates a dog object called my_dog. The program then sets the name and breed properties of the object, and calls the bark() method.

The output of the program is:

```
Woof!
```

22. What is Function Prototype ?

Ans :- A function prototype is a declaration of the function that informs the program about the number and kind of parameters, as well as the type of value the function will return.

In C++, a function prototype is declared using the following syntax:

returnType functionName(type1 argument1, type2 argument2, ...);

For example, the following is a function prototype for a function called `addNumbers()` that takes two integer arguments and returns an integer:

int addNumbers(int a, int b);

23. What is Operator (Write program) ?

ANS :- An operator is a symbol that performs a specific operation on one or more operands. Operators are used to perform arithmetic operations, logical operations, and bitwise operations.

Here is a list of some of the most common operators in C++:

- Arithmetic operators: +, -, *, /, %
- Logical operators: &&, ||, !
- Bitwise operators: &, |, ^, ~, <<, >>
- Assignment operators: =, +=, -=, *=, /=, %=, &=, |=, ^=, <<=, >>=

Here is a program that demonstrates how to use some of the most common operators in C++:

```
#include <iostream>
using namespace std;

int main() {
  int a = 10;
  int b = 20;
  int c;

  // Arithmetic operators
  c = a + b;
  cout << "a + b = " << c << endl;

  c = a - b;
  cout << "a - b = " << c << endl;

  c = a * b;
  cout << "a * b = " << c << endl;

  c = a / b;
  cout << "a / b = " << c << endl;

  c = a % b;
  cout << "a % b = " << c << endl;
```

```
    // Logical operators
    bool is_true = true;
    bool is_false = false;

    cout << "is_true && is_false = " << (is_true && is_false) << endl;
    cout << "is_true || is_false = " << (is_true || is_false) << endl;
    cout << "!is_true = " << !is_true << endl;

    return 0;
}
```

This program will print the following output:

```
a + b = 30
a - b = -10
a * b = 200
a / b = 0.5
a % b = 0
is_true && is_false = false
is_true || is_false = true
!is_true = false
```

24 . Scope resolution Operator ?

ANS :- The scope resolution operator (::) is used to access variables and functions that exist within a specific scope. The scope resolution operator is very useful when two variables in different scopes have the same name.

25. Type Casting in C++ ?

Ans :- Type casting in C++ is the process of converting a value from one data type to another. It allows you to change the interpretation of the data temporarily to perform specific operations. There are several types of type casting:

1. Implicit Type Casting (Automatic Type Conversion):

Occurs automatically when the compiler converts data from one type to another without the need for explicit syntax.
Generally, it is performed when there is no loss of data or precision.
For example, converting an int to a float or a float to a double.

2. Explicit Type Casting (Type Conversion):

Involves manually converting data from one type to another using explicit syntax.
Performed when there may be potential loss of data or precision.
You can use parentheses and the desired data type within them to perform explicit type casting.

26. Implicit and Explicit Type conversation with example ?

ANS :-

Implicit Type conversation

```
#include <iostream>
using namespace std;

int main() {
   int num = 10;
   float floatNum = num; // Implicit type conversion (int to float)

   cout << "Float Number: " << floatNum << endl; // Output: 10.0

   return 0;
}
```

Explicit Type conversation

```
#include <iostream>
using namespace std;

int main() {
   float floatNum = 3.14;
   int intNum1 = static_cast<int>(floatNum); // Explicit type casting (float to int)
   int intNum2 = int(floatNum); // Another way of explicit type casting

   cout << "Integer Number 1: " << intNum1 << endl; // Output: 3
   cout << "Integer Number 2: " << intNum2 << endl; // Output: 3

   return 0;
```

	}

27. Constructor and destructure (Write Program) ?

Ans :-1. A constructor is a special function that is called when an object of a class is created. The constructor is used to initialize the object's data members.

2. A destructor is a special function that is called when an object of a class is destroyed. The destructor is used to deallocate the object's memory.

Here is a simple program that demonstrates how to use constructors and destructors:

```
#include <iostream>
using namespace std;

class MyClass {
public:
  MyClass() {
    cout << "Constructor is called" << endl;
  }

  ~MyClass() {
    cout << "Destructor is called" << endl;
  }
};

int main() {
  MyClass my_object;

  return 0;
}
```

This program will print the following output:

```
Constructor is called
Destructor is called
```

28. Inheritance in C++ . Write code ?

ANS :- Inheritance is a feature of object-oriented programming (OOP) that allows you to create new classes from existing classes. The new class, called the **derived class**, inherits the properties and behaviors of the existing class, called the **base class**.

For example, let's say you have a class called Animal that has the properties name and age, and the behavior eat(). You can create a new class called Dog that inherits the properties and behaviors of the Animal class. The Dog class can also have its own properties and behaviors, such as the breed property and the bark() behavior.

In C++, inheritance is implemented using the : keyword. For example, the following code creates a Dog class that inherits from the Animal class:

C++

```
class Dog : public Animal {
public:
  string breed;
  void bark() {
    cout << "Woof!" << endl;
  }
};
```

29. Control Structure Statement ?

ANS :- Control structures in C++ are programming constructs that enable you to control the flow of execution in a program. They allow you to make decisions, repeat actions, and alter the program's sequence based on certain conditions. The three main types of control structures are:

1. Conditional Statements:-

`if`, `else if`, and `else`: Allows you to execute different blocks of code based on specified conditions.

`switch`: Allows you to select one of many code blocks to execute based on the value of an expression.

2. Loops:- `while`: Repeats a block of code while a condition is true.

`do-while`: Repeats a block of code at least once, then continues while a condition is true.

`for`: Executes a loop with initialization, condition, and iteration expression.

3. Jump Statements:- `break`: Terminates the nearest loop or switch statement.

`continue`: Skips the rest of the current iteration in a loop and starts the next iteration.

`return`: Exits the function and returns a value to the caller

30 . Difference between Procedural and OOP ?

ANS :- Procedural programming: Procedural programming is a programming paradigm that focuses on the procedures, or functions, that are used to solve a problem. In procedural programming, data is treated as a collection of variables, and the focus is on the steps that are needed to manipulate the data.

Object-oriented programming: Object-oriented programming is a programming paradigm that focuses on objects. In object-oriented programming, data is treated as objects, and the focus is on the interactions between objects.

31. Polymorphism and it's examples (write a Program) ?

ANS :- polymorphism and its examples in C++:

Polymorphism is the ability of an object to take on different forms. In C++, polymorphism can be achieved through **inheritance** and **virtual functions**.

Inheritance: Inheritance allows you to create new classes from existing classes. The new class, called the **derived class**, inherits the properties and behaviors of the existing class, called the **base class**. This means that the derived class can take on the form of the base class.

Virtual functions: Virtual functions are functions that can be overridden in derived classes. This means that the derived class can have its own implementation of the virtual function, which will take precedence over the implementation of the virtual function in the base class.

```
class Animal {
public:
  virtual void speak() {
    cout << "I am an animal" << endl;
  }
};
```

```
class Dog : public Animal {
public:
  void speak() {
    cout << "Woof!" << endl;
  }
};

int main() {
  Animal* animal = new Dog();
  animal->speak(); // Prints "Woof!"

  return 0;
}
```

32. Advantage of C++ ?

ANS :- C++ is a powerful and versatile programming language that has many advantages. Here are some of the most notable advantages of C++:

Speed: C++ is a compiled language, which means that it is converted into machine code before it is executed. This makes C++ very fast, especially when compared to interpreted languages like Python or JavaScript.

Control: C++ gives you a lot of control over your code. This allows you to optimize your code for performance and to create highly customized applications.

Portability: C++ code can be compiled and run on a variety of platforms, including Windows, macOS, Linux, and Android. This makes C++ a good choice for developing cross-platform applications.

Efficiency: C++ is a very efficient language. This means that your code will use fewer resources and will run faster than code written in other languages.

Scalability: C++ is a scalable language. This means that you can use it to create small, simple applications or large, complex applications.

Reusability: C++ code can be reused in other projects. This can save you time and effort when you are developing new applications.

33. Why do we use header file in C++ ?

Ans ;- A header file in C++ is a file that contains declarations of functions, classes, and variables that are used in other source files. Header files are used to avoid duplicating code in multiple source files.

Here are some of the reasons why we use header files in C++:

- **To avoid duplicating code:** Header files allow us to declare functions, classes, and variables once and then use them in multiple source files. This avoids duplicating code, which can make our code more maintainable and efficient.
- **To improve readability:** Header files allow us to group related declarations together, which makes our code easier to read and understand.
- **To improve compile-time errors:** Header files allow us to check for errors in our code at compile time, rather than at runtime. This can help us to catch errors earlier and to prevent them from causing problems in our applications.

```
#ifndef MY_HEADER_FILE
#define MY_HEADER_FILE

class MyClass {
public:
  int x;
  void my_function();
};
```

```
#endif
```

34. Write a Program using logical Operator?

ANS ;-

```
#include <iostream>

using namespace std;

int main() {
  int x = 10;
  int y = 20;

  // Logical AND operator
  cout << (x > 10 && y < 20) << endl; // 0
  cout << (x > 10 && y == 20) << endl; // 0
  cout << (x == 10 && y < 20) << endl; // 1

  // Logical OR operator
  cout << (x > 10 || y < 20) << endl; // 1
  cout << (x > 10 || y == 20) << endl; // 1
  cout << (x == 10 || y < 20) << endl; // 1

  // Logical NOT operator
  cout << (! (x > 10)) << endl; // 0
  cout << (! (x == 10)) << endl; // 1

  return 0;
}
```

This program will print the following output:

```
0
0
1
1
1
0
1
1
```

The `&&` operator is the logical AND operator. It returns `true` if both operands are true, and `false` otherwise.
The `||` operator is the logical OR operator. It returns `true` if either operand is true, and `false` otherwise.
The `!` operator is the logical NOT operator. It returns the opposite of the operand.