State Whether True or False

1. Data items in a class must always be private.
2. A function designed as private is accessible only to the member functions of that class
3. A function designed as public can be accessed like any other ordinary function.
4. Member functions defined inside a class specifier becomes inline functions by default.
5. Class members are public by default.
6. Friend functions have access to only public members of a class.
7. An entire class can be made a friend of another class.
8. Functions cannot return class objects.
9. Data members can be initialized inside a class specifier.
10. Constructors must be explicitly invoked.
11. Constructors defined in private section are useful.
12. Constructors can return values.
13. Destructors invoked automatically.
14. Destructors take input parameters.
15. Destructors can be overloaded.
16. Constructors cannot be overloaded.
17. Constructors can take default Arguments.
18. Data members of nameless objects can be initialized using constructors only.
19. Constructors can allocate memory at runtime.
20. A class member function can take its class objects as value argument.
21. The objects are always destroyed in the reverse order of their creation.
22. The destructors are always called in the reverse order of constructors

Nameless objects (scope is within a single statement in which it is created)-> Data members to be intiatized at the time creation of object -> Constructors

Difference between C structures and C++ structures

In C++, struct and class are exactly the same things, except for that struct defaults to public visibility and class defaults to private visibility.  
**Some important differences between the C and C++ structures:**

1. **Member functions inside structure**: Structures in C cannot have member functions inside structure but Structures in C++ can have member functions along with data members.
2. **Direct Initialization:** We cannot directly initialize structure data members in C but we can do it in C++.

|  |
| --- |
| // C program to demonstrate that direct  // member initialization is not possible in C  #include <stdio.h>    struct Record {      int x = 7;  };    // Driver Program  int main()  {      struct Record s;      printf("%d", s.x);      return 0;  }  /\* Output :  Compiler Error     6:8: error: expected ':', ', ', ';', '}' or    '\_\_attribute\_\_' before '=' token    int x = 7;          ^    In function 'main': \*/ |

**Output:**

7

1. **Using struct keyword:** In C, we need to use struct to declare a struct variable. In C++, struct is not necessary. For example, let there be a structure for Record. In C, we must use “struct Record” for Record variables. In C++, we need not use struct and using ‘Record‘ only would work.
2. **Static Members:** C structures cannot have static members but is allowed in C++.

|  |
| --- |
| // C program with structure static member  struct Record {      static int x;  };    // Driver program  int main()  {      return 0;  }  /\* 6:5: error: expected specifier-qualifier-list     before 'static'       static int x;       ^\*/ |

This will generate an error in C but no error in C++.

1. **Constructor creation in structure**: Structures in C cannot have constructor inside structure but Structures in C++ can have Constructor creation.

|  |
| --- |
| // C program to demonstrate that Constructor is not allowed  #include <stdio.h>    struct Student {      int roll;      Student(int x)      {          roll = x;      }  };    // Driver Program  int main()  {      struct Student s(2);      printf("%d", s.x);      return 0;  }  /\* Output :  Compiler Error     [Error] expected specifier-qualifier-list      before 'Student'     [Error] expected declaration specifiers or     '...' before numeric constant     [Error] 's' undeclared (first use     5555555555in this function)     In function 'main': \*/ |

**Output:**

2

1. **sizeof operator:**This operator will generate **0** for an empty structure in C whereas **1** for an empty structure in C++.

|  |
| --- |
| // C program to illustrate empty structure  #include <stdio.h>    // empty structure  struct Record {  };    // Driver program  int main()  {      struct Record s;      printf("%d\n", sizeof(s));      return 0;  } |

Output in C:

0

Output in C++:

1

1. **Data Hiding:** C structures do not allow concept of Data hiding but is permitted in C++ as C++ is an object oriented language whereas C is not.
2. **Access Modifiers:** C structures do not have access modifiers as these modifiers are not supported by the language. C++ structures can have this concept as it is inbuilt in the language.

Polymorphism

* + - Ability to take more than one form
    - Single name many forms

Two Types

* + - Compile Time Polymorphism –
    - (Function Overloading, Operator Overloading)
      1. Which function is to be called is to be decided at the time of compilation.

Function overloading- We can define multiple functions with the same name but having different no. and type of arguments.

Add(int x, int y)

Add(Complex c1, complex c2)

Add(char a, char b)

Add(int x, int y, int z)

Operator Overloading:

+, - , \*, /

These operators can work with integer and float values.

2+2

2-2

2\*2

2/2

These are already defined by complier and we are using them directly.

+(int,int)

/(float,float)

+(float,float)

Complex

{

Complex + (Complex C2)

}

C3=c1.add(c2);

C3=c1+c2;

C3=c1-c2; subtract

C3=c1\*c2; Multiplication

C3=c1/c2;

Matrix

{

Matrix \* (matrix m2)

}

Matrix m1,m2,m3;

M3=m1.multiplication(m3);

M3=m1\*m2;

A+b+c;

2+3+4

5+4

* + - Run Time Polymorphism