1. **What is a constructor?**

Constructors allow initialization of objects at the time of their creation. Constructor function is a special function that is a member of the class and has same name as that of the class. An object's constructor is automatically called whenever the object is created (statically or dynamically). Constructors are always public. They are used to make initializations at the time of object creation.

Consider following example of a stack class:

```
#include <iostream>
using namespace std;
class stack
{
        int top, bottom;
        int data[20];
        stack()         //constructor function
        {
           top = bottom;
           cout <<"Inside Stack Constructor: Stack
                   initialized";
        }
        int stackfull()
        {
              return ((top == max – 1)?1:0);
        }
        int stackempty()
        {
             return (top == bottom)?1:0);
        }
        void push(int no)
        {
                if(stackfull())
                        cout<<"Stack is full";
                else
                        data[++top] = no;
        }
        int pop()
        {
                if(stackempty())
                     cout<<"Nothing to pop.. Stack is Empty!";
                else
                       return(data[top--];
        }
};

int main()
{
        int i, no;
        stack st; //object is created; hence constructor is
              invoked-  stack initialization done
```

```
        cout <<" Entered Main\n";
        for(i = 0; i < 10; i++)
                st.push(i);
        no = s.pop();
        cout <<"The popped element is:"<
                return 0;
}
```

The o/p of the program would be:

```
Entered Main
Inside Stack Constructor: Stack initialized
The popped element is: 9
```

As seen above, the stack object is initialized automatically at the time of creation by the constructor. So we don't need to write and invoke initialization functions explicitly.

2. What are destructors?

Destructors are complements of constructors. When an object is destroyed, its destructor is automatically called. Destructors are mainly useful for doing the clean up job. E.g. an object may have allocated some memory during its lifetime; destructors are the place where this memory is deallocated. Or an object may need to close some files by releasing its handles which it had previously obtained.

Destructor function has same name as that of a constructor; but the name is preceded by a tilde ('~') sign.

We will expand the above example of a stack class to include a destructor:

```
#include <iostream>
using namespace std;
class stack
{
        int top, bottom;
        int data[20];
        stack()         //constructor function
        {
                top = bottom;
                cout <<"Inside Stack Constructor: Stack initialized\n";
        }
        int stackfull()
        {
                return ((top == max – 1)?1:0);
        }
        int stackempty()
        {
                return (top == bottom)?1:0);
        }
        void push(int no)
        {
                if(stackfull())
```

```
                cout<<"Stack is full";
            else
                data[++top] = no;
        }
        int pop()
        {
            if(stackempty())
                cout<<"Nothing to pop.. Stack is Empty!\n";
            else
                return(data[top- -];
        }
        ~stack() //destructor function
        {
            cout <<"Inside Stack Destructor: Stack Destroyed\n";
        }
};
int main()
{
    int i, no;
    stack st; //object is created; hence constructor is invoked- stack initialization done
    cout <<"Entered Main\n";
    for(i = 0; i < 10; i++)
        st.push(i);
    no = s.pop();
    cout <<"The popped element is:"<<no << "\n";
    return 0;
}// at the end of object's lifetime, destructor is invoked
```

The o/p of the program would be:
Entered Main
Inside Stack Constructor: Stack initialized
The popped element is: 9
Inside Stack Destructor: Stack Destroyed
As seen from the o/p the constructors and destructors are automatically called.

3. **What are the restrictions apply to constructors and destructors?**

The following restrictions apply to constructors and destructors

Constructors and destructors don't return values.
The addresses of constructors and destructors can't be taken so we can't use references and pointers on them.
Constructors cannot be declared with the keyword virtual.
Constructors and destructors cannot be declared static, const, or volatile.

4. **Explain the order in which constructors are called when an object of a derived class is created.**

The constructors of any virtual base classes are called first in the order of inheritance.
Non-virtual base class constructors are called next.
The derived class constructor is called last.

5. **Difference between a copy constructor and an assignment operator.**

Copy constructor creates a new object which has the copy of the original object .

On the other hand assignment operators does not create any new object. It instead, deals with existing objects.

6. **What is a virtual destructor? Explain the use of it.**

If the destructor in the base class is not made virtual, then an object that might have been declared of type base class and instance of child class would simply call the base class destructor without calling the derived class destructor.

Hence, by making the destructor in the base class virtual, we ensure that the derived class destructor gets called before the base class destructor.

```
class a
{
        public:
        a(){printf("\nBase Constructor\n");}
        ~a(){printf("\nBase Destructor\n");}
};

class b : public a
{
       public:
       b(){printf("\nDerived Constructor\n");}
       ~b(){printf("\nDerived Destructor\n");}
};
int main()
{
       a* obj=new b;
       delete obj;
       return 0;
}
```

Output:
Base Constructor
Derived Constructor
Base Destructor

By Changing
~a(){printf("\nBase Destructor\n");}
to
virtual ~a(){printf("\nBase Destructor\n");}

Output:
Base Constructor
Derived Constructor
Derived Destructor
Base Destructor

7. **Virtual constructor:**

A constructor of a class can not be virtual and if causes a syntax error.

8. **What is private, public and protected Inheritance?**

Private Inheritance
The Public and protected members of Base class become private members of the derived class.

Public Inheritance
All the public members and protected members are inherited as public and protected respectively.

Protected Inheritance
Public and Protected members are derived as protected members.

9. **What is virtual function? Explain with an example.**

A virtual function is a member function that is declared within a base class and redefined by a derived class. To create virtual function, precede the function's declaration in the base class with the keyword virtual. When a class containing virtual function is inherited, the derived class redefines the virtual function to suit its own needs.

Base class pointer can point to derived class object. In this case, using base class pointer if we call some function which is in both classes, then base class function is invoked. But if we want to invoke derived class function using base class pointer, it can be achieved by defining the function as virtual in base class, this is how virtual functions support runtime polymorphism.

Consider following program code:

```
Class A
{
      int a;
      public:
      A()
      {
            a = 1;
      }
      virtual void show()
      {
              cout <<a;
      }
};
```

```
Class B: public A
{
        int b;
        public:
        B()
        {
              b = 2;
        }
        virtual void show()
        {
               cout <<b;
        }
};

int main()
{
         A *pA;
         B oB;
         pA = &oB;
         pA->show();
         return 0;
}
```

Output is 2 since pA points to object of B and show() is virtual in base class A.

10. What are pure virtual functions?

Pure virtual functions are also called ‘do nothing functions’.
e.g. virtual void abc() = 0;
When a pure virtual fnction is declared in the base class, the compiler necessitates the derived classes to define those functions or redeclare them are pure virtual functions. The classes containing pure virtual functions cannot be used to declare objects of their own. Such classes are called as abstract base classes.

11. Explain the use of Vtable.

Vtables are used for virtual functions. Its a shortform for Virtual Function Table.

It's a static table created by the compiler. Compiler creates a static table per class and the data consists on pointers to the virtual function definitions. They are automatically initialised by the compiler's constructor code.

Since virtual function pointers are stored in each instance, the compiler is
enabled to call the correct vrtual function at runtime.

12. What is a virtual base class?

An ambiguity can arise when several paths exist to a class from the same base class. This means that a child class could have duplicate sets of members inherited from a single base class.

C++ solves this issue by introducing a virtual base class. When a class is made virtual, necessary care is taken so that the duplication is avoided regardless of the number of paths that exist to the child class.

13. Explain the problem with overriding functions.

Overriding of functions occurs in Inheritance. A derived class may override a base class member function. In overriding, the function names and parameter list are same in both the functions. Depending upon the caller object, proper function is invoked.

Consider following sample code:

```
class A
{
	int a;
	public:
	  A()
	  {
		a = 10;
	  }
	  void show()
	  {
		cout << a;
	  }
};

class B: public A
{
	int b;
	public:
	B()
	{
	    b = 20;
	}
	void show()
	{
		cout << b;
	}
};

int main()
{
	A ob1;
	B ob2;
	ob2.show(); // calls derived class show() function. o/p is 20
	return 0;
}
```

As seen above, the derived class functions override base class functions. The problem with this is, the derived class objects can not access base class member functions which are overridden in derived class.

Base class pointer can point to derived class objects; but it has access only to base members of that derived class.

Therefore, pA = &ob2; is allowed. But pa->show() will call Base class show() function and o/p would be 10.

14. Describe static and dynamic binding of functions.

Static Binding:
By default, matching of function call with the correct function definition happens at compile time. This is called static binding or early binding or compile-time binding. Static binding is achieved using function overloading and operator overloading. Even though there are two or more functions with same name, compiler uniquely identifies each function depending on the parameters passed to those functions.

Dynamic Binding:
C++ provides facility to specify that the compiler should match function calls with the correct definition at the run time; this is called dynamic binding or late binding or run-time binding. Dynamic binding is achieved using virtual functions. Base class pointer points to derived class object. And a function is declared virtual in base class, then the matching function is identified at run-time using virtual table entry.

15. Overloading vs. overriding

- Overriding of functions occurs when one class is inherited from another class. Overloading can occur without inheritance.
- Overloaded functions must differ in function signature ie either number of parameters or type of parameters should differ. In overriding, function signatures must be same.
- Overridden functions are in different scopes; whereas overloaded functions are in same scope.
- Overriding is needed when derived class function has to do some added or different job than the base class function.
- Overloading is used to have same name functions which behave differently depending upon parameters passed to them.

16. What is Object slicing? Give an example.

When a Derived Class object is assigned to Base class, the base class' contents in the derived object are copied to the base class leaving behind the derived class specific contents. This is referred as Object Slicing. That is, the base class object can access only the base class members. This also implies the separation of base class members from derived class members has happened.

```
class base
{
    public:
        int i, j;
};
class derived : public base
{
     public:
          int k;
};
int main()
```

```
{
    base b;
    derived d;
    b=d;
    return 0;
}
```
here b contains i and j where as d contains i, j& k. On assignment only i and j of the d get copied into i and j of b. k does not get copied. on the effect object d got sliced.

17. What is inline function?

An inline function is a combination of macro & function. At the time of declaration or definition, function name is preceded by word inline.

When inline functions are used, the overhead of function call is eliminated. Instead, the executable statements of the function are copied at the place of each function call. This is done by the compiler.

Consider following example:

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

inline int sqr(int x)
{
        int y;
        y = x * x;
        return y;
}
int main()
{
            int a =3, b;
            b = sqr(a);
            cout <<b;
            return 0;
}
```

Here, the statement b = sqr(a) is a function call to sqr(). But since we have declared it as inline, the compiler replaces the statement with the executable stmt of the function (b = a *a)

Please note that, inline is a request to the compiler. If it is very complicated function, compiler may not be able to convert it to inline. Then it will remain as it is. E.g. Recursive function, function containing static variable, function containing return statement or loop or goto or switch statements are not made inline even if we declare them so.

Also, small functions which are defined inside a class (ie their code is written inside the class) are taken as inline by the compiler even if we don't explicitly declare them so. They are called auto inline functions.

18. What is the difference between inline functions and macros?

A macro is a fragment of code which has been given a name. Whenever the name is used, it is replaced by the contents of the macro. There are two kinds of macros: Object-like macros and function-like macros.

Inline function is a function that is expanded in line when the function is called. That is the compiler replaces the function call with the function code (similar to macros).

The disadvantage of using macros is that the usual error checking does not occur during compilation.

19. What are static member functions?

A static function can have an access to only other static members (functions or variables) declared in the same class.
A static member function can be called using the class name instead of its objects.
E.g. classname :: functionname;

20. What happens when recursion functions are declared inline?

The call to the body of the function is replaced by an inline function. This reduces the saving context on stack overhead. This process is efficient when the size of the function is small and invoked occasionally. Deep nesting of a method is done when a function is invoked recursively. The inline function is invoked recursively, and every call to itself is replaced with the body of the function, thus consumes a lot of code space.

21. Advantages and disadvantages of using macro and inline functions.

A textual substitution is provided by a macro as a constant, where as an inline function is procedure which is called at each time. Although the macros have few advantages over inline functions, the disadvantages are numerous. For example, a macro can not perform type checking and validation, as these operations are performed in a function at the most.

Everyone should decide for themselves to use them, but the use of inline functions over macros is advocated by Bjarne Struoustrup, the creator of C++. The imperative features of inline functions are frequently used with classes in C++. There is similarity between invoking normal functions and inline functions, except that, inline functions are never actually called. The inline functions, as their name suggests, are expanded in line at every time of invocation. All that is needed to invoke an inline function is to prefix the key word ‘inline’ to the function.

22. What is static class data?

Static data members of a class are declared by preceding the member variable’s declaration with the keyword static. Only one copy of static data members exist and all objects of the class share that variable. Unlike regular data members, individual copies of a static member variable are not made for each object. How many ever no of objects of a class are created, only one copy of static data member is shared amongst all of them. All static variables are initialized to zero before the first object is created.

When a member variable is declared static within a class, it is not defined (ie storage is not allocated for it) We must provide a global definition for it outside the class. This is done by redeclaring the static variable

using scope resolution operator (‘::’) to identify the class it belongs to. This causes storage for the class to be allocated.

23. What are static member functions?

A static function can have an access to only other static members (functions or variables) declared in the same class.
A static member function can be called using the class name instead of its objects.
E.g. classname :: functionname;

24. What is a friend function?

Private data members cannot be accessed from outside the class. However, situations arise where two classes need to share a particular function. For such situations, C++ introduces friend functions. These are functions that can be made friendly with both the classes, thus allowing these functions to have an access to the private data of these classes.

25. What is a namespace? What happens when two namespaces having the same name?

A conceptual space for grouping identifiers, classes etc. for the purpose of avoiding conflicts with the elements of an unrelated code which have the same names.

When two name spaces are having same name, the corresponding compiler uses our name spaces instead of the type.

26. Distinguish between new and malloc and delete and free().

Delete is assocated with new and free(0 is associated with malloc()

New
Its an operator
delete is associated with new
It creates an object
It throws exceptions if memory is unavailable
Operator new can be overloaded
You can state the number of objects to be created.

malloc()
It’s a function
free() is associated with malloc()
It does not create objects
It returns NULL
This cannot be overloaded
You need to specify the number of bytes to be allocated.