Name :- Pranav Mishra Registration No :- 12114762
Roll No :- RD0C09A55 Course Code : CAP444
Teacher : Dr. Yasir Iqbal Sir Course Title :- OOP using C++

Ans :-1 namespace std :-

It is container for set of identifiers. It provide a level of direction to specific identifiers, thus making it possible to distinguish between identifiers with the same exact name.

C++ has a standard library that contains common functionality you use in building your applications like containers, algorithms, etc. If name used by these were out in the open

The using namespace statement just means that in the scope it is present, make all the things under the std namespace available without having to prefix std:: before each of them.

Why we use namespace ?:-

If you are going to use bunch of code of A library then its going to boring to add prefix very time while calling the function of A library. Then what is the solution you can add "using namespace std" at the top of your code and then just call get_data without using A prefix for the rest of code

The namspace std is special, The built in C++ library routines are kept in the standard namespace std. That includes stuff like cout, cin, string, vector, map etc. Because these tool are used so commonly it's popular to add "using namespace std' at the top of the code so that you won't have to type the std:: prefix constantly. It only make our task eeasy,
It is not necessary.

Program showing the use of Namespace in C++ :-

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

namespace firstone
 {
  void fun()
  {
     cout<< "This is the first NS" << endl;
  }
}
 namespace secondone
  {
  void fun()
  {
    cout<< "This is the second NS" << endl;
  }
}
using namespace firstone;
 int main()
 {
     fun();
}
```

Output :-

This is the First NS

Ans:-2:- Friend Function :-

A friend function of a class is defined outside that class' scope but it has the right to access all private and protected members of the class. Even though the prototypes for friend functions appear in the class definition, friend are not member functions.

A friend can be a function, function template, or member function, or a class or class template, in which case the entire class and all of its members are friends.

To declare a function as a friend of a class, precede the function prototype in the class definition with keyword friend.

Declaration of friend function :-

```
class Boy {
        double width;
    public:
        double length;
    friend void show (Boy Boy);
     void print (double width);
};
```

Generally, non-member functions can not access the private members of a particular class. Once declared as a friend function, the function is able to access the private and the protected members of these classes.

Use of Friend function:

We use the friend function whenever we have to access the private or protected members of a class. This is only the case when we do not want to use the objects of that class to access these private or protected members.

Without the friend function, we will use the object of these classes to access all the members

Friend function helps us avoid the scenario where the function has to be a member of either of these classes for access.

Friend function are also used in operator overloading. The binary operator overloading in C++ using the friend function can be done.

All friend functions pass all their arguments through the arguments list, and each arguments value is subject to assignment-compatible conversions.

~~Syntax of friend function:~~

~~class Box~~ {

Ans:3:- Type Conversion:-

The type conversion is an operation that takes a data object of one type and creates the equivalent data object of multiple types.

conversion_op : type1 → type2

There are two types of type conversion in C++.

1:- Implicit Conversion

2:- Explicit Conversion

Implicit Conversion:-

It is done by the compiler on its own, without any external trigger from the user.

Generally take place when in an expression more than one data type is present. In such condition type conversion takes place to avoid loss of data.

All the data types of the variables are upgraded to the data type of the variable with largest data type.

Example:-

```
#include<iostream>
 using namespace std;
int main()
  {
        int x=10;
        char y='a';

            x= x+y;
        float z= x+1.0;

cout<<"x = "<<x<<endl
   <<"y = "<<y<<endl
   <<"z = "<<z<<endl;
return 0;
}
```

Output:-

x=107, z=108.000

Explicit type Conversion:

This process is also called type casting and it is user defined. Here the user can type cast the result to make it of a particular data type.

It can be done by two ways:

→ Converting by assignment.

→ Conversion using cast operator.

Example:

```
#include<iostream>
using namespace std;
int main()
 {
        float f= 3.5;
        int b= static_cast<int>(f);
        cout<<b;
 }
```

Output: 3