Prasanna Wadekar 9970248407 sunbeam notes CPP 51 =00

SUNBEAM

Institute of Information Technology

Placement Initiative

C++ Notes

Language Basics :

Generally languages are classified as follows:

1. Procedure oriented programming language.
   e.g. C, Pascal etc.
2. Object Oriented Programming language.
   e.g. C++, Java, C# etc.
3. Object Based Programming Language.
   e.g. ada, modula-2, Bisual Basic
4. Logic Oriented Programming Language.
5. Rule Oriented Programming Language.
6. Constraint Oriented Programming Language.

Some of the advantages of object oriented programming are as follows:

1. Code Reusability

Objects created for object oriented programs can easily be reused in other programs.

2. Lower Cost Of Development

More effort is put into the object oriented analysis and design which lowers overall cost of development.

3. Improved Software Maintainability

Since design is modular, part of the system can be updated in case of issues without a need to make large scale changes.

4. Higher Quality Software

Faster development of software and lower cost of development allows more time and resources to be used in the verification of the software which tends to result in higher quality software.

PRASANNA

Parts of OOP model:

There are four major and three minor elements of object oriented programming model.

Major means a language without any one of these elements is not considered as object oriented.

1. Abstraction
2. Encapsulation
3. Modularity
4. Hierarchy

Minor means, each of these elements is useful but not essential.

1. Typing
2. Concurrency
3. Persistence

Brief history of C++:

The C++ programming language has a history going back to 1979, when Bjarne Stroustrup was doing work for his Ph.D. thesis. One of the language Stroustrup had the opportunity to work with was a language called Simula, which as the name implies is a language primarily designed for simulations. The Simula 67 language is regarded as the first language to support the object-oriented programming paradigm. Stroustrup found that this paradigm was very useful for software development, however the Simula language was far too slow for practical use.

Shortly thereafter, he began work on "C with Classes", which as the name implies was meant to be a superset of the C language. His goal was to add object-oriented programming into the C language, which was and still is a language well-respected for its portability without sacrificing speed or low-level functionality.

The first C with Classes compiler was called Cfront. It was a program designed to translate C with Classes code to ordinary C. Cfront would later be abandoned in 1993 after it became difficult to integrate new features into it, namely C++ exceptions. Nonetheless, Cfront made a huge impact on the implementations of future compilers and on the Unix operating system.
In 1983, the name of the language was changed from C with Classes to C++.

Data types:

Data type in any language described three things
1.How much memory is required to store the data.
2.Which kind of data that memory can hold.
3.Which operations we can perform on that data.
Basically there are only two types of datatypes but for our convenience we will classify it into three types
-Fundamental datatypes

1.void	unknown size	
2.bool	1 byte	[can contain either true or false value]
3.char	1 byte	[ASCII Compatible]
4.wchar_t	2 bytes	[Unicode Compatible]
5.int	4 bytes	
6.float	4 bytes	
7.double	8 bytes	

-Derived Datatypes
1.array
2.function
3.pointer
4.reference

-User Defined Data Types
1.Structure
2.Union
3.Class

Access Specifers in C++

There are three acess specifiers in c++
1.private: If we declare elements of the class as private then we can access these members inside same class in which it is declared. We cannot access private members inside nonmember function. Generally data members should be declared as private.

2.protected:If we declare elements of the class as protected then we can access these members inside same class in which it is declared as well as inside derived class. We can not access protected members outside the class.

3.public:If we declare elements of the class as public then we can access these members inside same class in which it is declared, inside derived class as well as inside non member function. Generally member function should be declared as public.

Structure in C++

-structure is collection of related elements which gets continueous memory space.
-in case of instantiation use of struct keyword is optional.

-we can write variable as well as function inside structure in c++.

-by default members of the class are treated as a public.

-to create light weight object we should use structure.

Data member:

-Variable declared inside class is called data member.

-Data member is also called field, property or attribute.

-If we create object then it gets single copy of the data member i.e. data member gets space inside object.

-As a convention name of data member should start with underscore(_).

Member function:

-Function implemented inside a class is called member function.

-Member function is also called method, operation,behaviour or message.

-Member function do not get space inside object rather all the objects of same type share single copy of member function.

-As a convention member function name should start with capital alphabet.

Class:

-syntactilly class is collection of data member and member function.

-In general 'Class is collection of such objects which represents common properties and behaviour'.

-Since object is always created by looking toward the, class is considered as a template/model/blueprint for an object.

-class is logical entity.

-class is a user defined data type.It is also called abstract data type.

Object:

-Syntactily, object is variable of class.

-An entity which is having physical existance is called object.

-In object oriented way, we can define object as a instance of class.

-At the time of creation of object use of class keyword is optional.

```
class CPoint pointInstance;        //valid
CPoint   pointInstance;            //valid
```

Some important terminology

1. Instantiation-Process of creating object from a class is called instantiation.
2. Concrete class-If class allows us to create an object of a class then it is called concrete class.
3. Concrte method-Member function of a class which is having a body is called concrete method.
4. Abstract class: If class do not allows us to create an object of a class then it is called abstract class.
5. Abstract method: Member function of a class which do not contain body is called concrete method.

Message Passing:

Process of calling member function on object is called message passing.consider the following example

```
CPoint pt1;
pt1.CPoint::PrintRecord( );        //valid : Message passing
pt1.PrintRecord( );                //valid : Message passing
```

Characteristics of an Object:

An object is an entity that has state, behavior, and identity. The structure and behavior of similar objects are defined in their common class. The terms instance and object are interchangeable.

1.State - The state of an object encompasses all of the properties of the object plus the current values of each of these properties.

2.Behaviour - Behavior is how an object acts and reacts, in terms of its state changes and message passing.

3.Identity - Identity is that property of an object which distinguishes it from all other objects.

Empty class and object:

A class which do not conain any member is called empty class.consider the following code.

```
class CEmpty
{
        //Empty implementation
};
```

According to Bjarne stroustrup, to differentiate object from a class, object must get space inside memory and it should be non zero. But compiler vendors do optimization because there is no use of such memory. So size of object of empty class is one byte.

this pointer:

- this is a keyword in C++.
- It is implicit pointer available inside every non static member function which is used to store address of current object.
- this is constant pointer and its general type is as follows

  ClassName* const this
- this represent address of current object while (*this)represents current object itself.
- Data member and member function can communicate with each other with the help of this pointer so this is considered as connection between data member and member function.
- If any member function accepts parameters then this pointer is always pass as a first parameter implicitly.
- We can not declare this as a function parameter explicitly.
- Use of this keyword is optional but we should use it.
- If name of data member and function parameter is same then it is mandatory to use this keyword before data member.
- Following function do not get this pointer:
  1. Global function
  2. Static member function
  3. Friend function.

Array of Objects:

```
CPoint arr[ 3 ];    //Here parameterless constructor will call three times
for( int index = 0; index < 3; ++ index )
        arr[ index ].PrintRecord();

CPoint arr[ 3 ] = { CPoint(10,20),CPoint(30,40),CPoint(50,60)};        //Here parameterized constructor will call three times
for( int index = 0; index < 3; ++ index )
        arr[ index ].PrintRecord();
```

Namespace is logical term, which is used to control scope of global members.In other words to avoid name ambiguity or name clashing or name collision we should use namespace.

```
int num1 = 10;
int num1 = 20;    //error: 'num1' : redefinition; multiple initialization
int main( void )
{
        printf("Num1    :       %d\n",num1);
        return 0;
}
```

As shown in above code we can not give same name to multiple members in same scope.Problem may occur, if name of elements refered from different library and name of element declared in same program is same. To avoid this problem we should implement global members inside namespace. we can write above program as follows

```
namespace NA
{
        int num1 = 10;
}
namespace NB
{
        int num1 = 20;    //Now no error
}
int main( void )
{
        printf("Num1    :       %d\n",NA::num1);        //Num1  :       10
        printf("Num1    :       %d\n",NB::num1);        //Num1  :       20
        return 0;
}
```

As shown in above code, we should use namespace name and scope resolution operator to access elements of namespace. We can implement namespace inside scope of another namespace.It is called nested namespace.First we will see nested namespace with different element names.

```
int num1 = 10;
namespace NA
{
        int num2 = 20;
        namespace NB
        {
                int num3 = 30;
        }
}
int main(void)
```

```
{
        printf("Num3    :       %d\n", NA::NB::num3);           //Num3  :       30
        printf("Num2    :       %d\n", NA::num2);               //Num2  :       20
        printf("Num1    :       %d\n", ::num1);                 //Num1  :       10
        return 0;
}
```

Generally we use the statement that to access global variable or function we should use :: operator.

In namespace topic we have discussed that to access elements of namespace we should :: operator.Consider above code. Since num3 is member of NB namespace & NB namespace is member of NA namespace we have written NA::NB::num3. num2 is member of NA namespace, we have written NA::num2.Even though num1 is not a member of any namespace we can use :: operator with it. Meaning of this statement is that, if we declare or implement global members without namespace it is considered as member of unnamed or global namespace.

If we want to use any member of namespace frequently then instead of specifying namespace name every time we can use using directive. The using directive allows the names in a namespace to be used without the namespace-name as an explicit qualifier. consider the following code.

```
int num1 = 10;
namespace NA
{
        int num2 = 20;
        namespace NB
        {
                int num3 = 30;
        }
}
int main(void)
{
        printf("Num1    :       %d\n", ::num1);                 //Num1  :       10
        using namespace NA;
        printf("Num2    :       %d\n", num2);                   //Num2  :       20
        printf("Num3    :       %d\n", NB::num3);               //Num3  :       30
        using namespace NB;
        printf("Num3    :       %d\n", num3);                   //Num3  :       30
        return 0;
}
```

We can write "using namespace namespace_name" statement any number of times.

We can give same name to the outer namespace as well as inner namespace.In this case we can not use 'using' directive to access namespace members.

```
namespace NA
{
        int num1 = 10;
        namespace NA
```

```
        {
                int num2 = 20;
        }
}
int main(void)
{
        printf("Num1    :       %d\n", NA::num1);           //Num1  :       10
        printf("Num2    :       %d\n", NA::NA::num2);       //Num2  :       20
        return 0;
}
```

We can give same namespace name to other namespace inside same scope.

```
namespace NA
{
        int num1 = 10;
}
namespace NA
{
        int num2 = 20;
}
int main(void)
{
        printf("Num1    :       %d\n", NA::num1);           //Num1  :       10
        printf("Num2    :       %d\n", NA::num2);           //Num2  :       20

        using namespace NA;
        printf("Num1    :       %d\n", num1);               //Num1  :       10
        printf("Num2    :       %d\n", num2);               //Num2  :       20
        return 0;
}
```

we should use using statement carefully otherwise we may get unexpected results. consider following code snippet.

```
namespace NA
{
        int num1 = 10;
}
int main(void)
{
        int num1 = 20;
        using namespace NA;
        printf("Num1    :       %d\n", num1);               //Num1  :       20
        printf("Num1    :       %d\n", num1);               //Num1  :       20

        printf("Num1    :       %d\n", NA::num1);           //Num1  :       20
```

```
        printf("Num1    :       %d\n", num1);                   //Num1  :       20
        return 0;
}
```

If name of members of namespace is same and if we try to access it with the help of using directive then compiler produces ambiguity error. In this case we should use fully qualified name. consider following code.

```
namespace NA
{
        int num1 = 10;
        namespace NB
        {
                int num1 = 20;
                int num2 = 30;
        }
}
int main(void)
{
        using namespace NA;
        printf("Num1    :       %d\n",num1);            //Num1  :       10
        using namespace NB;
        //printf("Num1  :       %d\n", num1);           //ambiguity error
        printf("Num1    :       %d\n", NB::num1);       //Num1  :       20
        printf("Num1    :       %d\n", num2);           //Num2  :       30
        return 0;
}
```

We can declare using directive at local scope as well as file scope.

In following code snippet we have declared "using namespace namespace_name" statement at local scope.

```
namespace NA
{
        int num1 = 10;
}
void show_record(void)
{
        using namespace NA;
        printf("Num1    :       %d\n", num1);
}
void display_record(void)
{
        using namespace NA;
        printf("Num1    :       %d\n", num1);
}
int main(void)
{
        ::show_record();
```

```
        ::display_record();
        return 0;
}
```

If we want to access namespace elements at multiple location then we should declare "using namespace namespace_name" statement at file scope. Consider the following example.

```
namespace NA
{
        int num1 = 10;
}
using namespace NA;
void show_record(void)
{
        printf("Num1     :        %d\n", num1);
}
void display_record(void)
{
        printf("Num1     :        %d\n", num1);
}
int main(void)
{
        ::show_record();
        ::display_record();
        return 0;
}
```

unnamed or anonymous namespace -

In C/C++,if we declare any global element as static then we can access such element inside same file only. In C++ unnamed namespace or anonymous namespace is superior alternative given for global static member declarations.

```
namespace
{
        int num1;
}
int main(void)
{
        num1 = 10;
        printf("Num1     :        %d\n", num1);
        return 0;
}
```

Each unnamed namespace has an identifier, assigned and maintained by the program and represented here by unique

above code is implicitly interpreted as follows:

```
namespace unique
{
        int num1;
}
int main(void)
{
        unique::num1 = 10;
        printf("Num1     :        %d\n", unique::num1);
        return 0;
}
```

namespace Alias -

if name of the namespace is too lengthy then we can create alias for it. Consider the following syntax.

```
namespace namespace_name
{
        int num1;
}
namespace NA = namespace_name;
int main(void)
{
        printf("num1     :        %d\n", NA::num1);
        return 0;
}
```

Points To Remember:

1. Namespace is logical term which is used to group or organize functionally equivalant or related code together.
2. In C++, We can achieve logical modularity with the help of namespace.
3. To access members of namespace we should use scope resolution(::)operator.
4. We can write any global variable, function, enum, structure, class as well as namespace inside namespace.
5. We can not write namespace inside class i.e. A namespace definition must appear either at file scope or immediately within another namespace definition.
6. We can not implement entry point function(main) inside namespace.
7. Inside same as well as different file we can give same name to the multiple namespaces.
8. We can not create instance of namespace.
9. Instead of declaring global element as static we should put them inside unnamed namespace.

Process of storing user defined value inside variable/object at the time of creation is called initialization.

e.g. `int number = 123;` //initialization

Constructor is special member function of class which is having same name as a class and it gets call implicitly when we create instance of class.

```
namespace NPoint
{
```

```
        class CPoint
        {
        private:
                int _xPosition;
                int _yPosition;
        public:
                CPoint( void )     //constructor
                {
                        this->_xPosition = 0;
                        this->_yPosition = 0;
                }
        };
}
int main(void)
{
        NPoint::CPoint pointInstance;        //Here for pointInstance object constructor will call
        return 0;
}
```

Due to following reasons constructor is special member function

-its name is same class name.

-it do not have any return type (but we can write return statement inside constructor).

-it is not designed to call explicitly on object rather it gets call implicitly.

-in complete life cycle of an object, constructor gets call only once.

Types of Constructor:

C++ supports three kinds of constructor.

1. Parameterless constructor.
2. Parameterized constructor.
3. Default constructor.

Most of us consider copy constructor is fourth type of constructor.But it is kind of parameterized constructor. We will discuss copy constructor seperatly later.

Parameterless constructor: A constructor which do not take any parameter is called parameterless constructor.

```
        class CPoint
        {
        private:
                // declaration of data members
        public:
                CPoint( void )//parameterless constructor
                {
                        //TODO
                }
```

```
};
```

parameterless constructor is also called zero parameter constructor or user defined default constructor.
if we create an object without passing argument then parameterless constructor gets call.

```
CPoint pt1;        //Here for pt1 object parameterless constructor will call
```

Note:No doubt it is default constructor but to avoid future confusions please refer it parameterless constructor.

Parameterized constructor: If constructor takes parameters then it is called parameterized constructor.

```
class CPoint
{
private:
        int _xPosition;
        int _yPosition;
public:
        CPoint(int value)//parameterized constructor
        {
                this->_xPosition = value;
                this->_yPosition = value;
        }
        CPoint( int xPosition, int yPosition )//parameterized constructor
        {
                this->_xPosition = xPosition;
                this->_yPosition = yPosition;
        }
};
```

If we create an object by passing argument then parameterized constructor gets call.

```
CPoint pt1( 10, 20 );      //Here for pt1 object parameterized constructor which is taking 2 parameters will call
CPoint pt2( 500 );         //Here for pt2 object parameterized constructor which is taking 1 parameter will call
```

How many and which kind of connstructor should be exist inside a class depends on instantiation. Constructor calling sequence is depends on order of object declaration.

Default constructor:If class do not contain any kind of constructor then compiler provides one constructor for the class it is called default constructor.
Default constructor do not perform any operation on data member declared in our class.It look like as follows

```
CPoint( void )
{
        /*empty*/
};
```

Compiler generated default constructor do not take any parameter i.e. compiler never generate default parameterized construcrtor.
If we want to create an object by passing argument then we can not rely on default constructor.In this case we should define parameterized constructor inside a class.
We will see exact use of default constructor in virtual function topic.

Constructor is designed for initialization it doesnt mean we can not call methods from it.Consider the following code:

```
class CPoint
{
private:
        int _xPosition;
        int _yPosition;
public:
        CPoint( int xPosition, int yPosition )
        {
                this->SetXPosition(xPosition);      //Here we have called methods from constructor.
                this->SetYPosition(yPosition);
        }
        void SetXPosition(int xPosition)
        {
                this->_xPosition = xPosition;
        }
        void SetYPosition(int yPosition)
        {
                this->_yPosition = yPosition;
        }
};
```

We can not declare constructor as const, volatile,virtual or static. We can declare constructor as inline only.Inline functtion we will discuss later.

Aggregate type:

If type allows us to store multiple elements then it is called aggregate type and object is called aggregate object.

In C language, array, structure and union is called aggregate type.Aggregate type allows us to initialize object using initializer list.Consider the following code:

```
int arr[ ] = { 10, 20, 30 };
struct point pt1 = { 10, 20 };
```

class may be classified as aggregate type if it obeys following rule.

1. class should not contain user defined constructor.
2. Class should contain public members.(at lest data members).
3. class should not be derived from any other class.
4. class should not contain virtual function.

consider the following example:

```
class CPoint        //Aggrgate class.
{
public:
        int _xPosition;
        int _yPosition;
```

```
public:
        int GetXPosition(void)
        {
                return this->_xPosition;
        }
        int GetYPosition(void)
        {
                return this->_yPosition;
        }
};
int main(void)
{
        CPoint pt1 = { 10, 20 };    //allowed
        return 0;
}
```

Aggregate class is also called Plain Old Data struct(POD- struct).

Test your understanding:

```
class CPoint
{
private:
        int _xPosition;
        int _yPosition;
public:
        CPoint( void );
        CPoint( int value );
        CPoint(int xPosition, int yPosition);
};
```

In following cases which constructor will call?

1. CPoint pt1;

2. CPoint pt2(10);

3. CPoint pt3(20, 30);

4. CPoint pt4( );

5. CPoint pt5 = 40 ;

6. CPoint pt6 = 50, 60;

7. CPoint pt7 = (50, 60);

8. CPoint pt8 = { 70, 80 };

9. CPoint pt9(90, 100);
CPoint pt10 = pt9;

10. CPoint* ptr;

As per our requirement,we can use any access specifier for constructor.If we declare constructor as private or protected then we can create instance of class only inside member function.If we declare constructor as public then we can create instance of class inside member function as well as non member function.

Note: Generally we should public access specifier for constructor.

Constructors member initializer list:

c++ allows us to intialize data members of the class inside constructors body as well as constructors member initializer list.If we initialize data members inside constructors body then it gets executed in the same order in which it is written inside constructor body.For example:

```
class CTest
{
private:
        int _x;
        int _y;
        int _z;
public:
        CTest(void)
        {
                this->_z = this->_y;
                this->_y = this->_x;
                this->_x = 10;
        }
};
int main(void)
{
        CTest test;         //10,garbage value,garbage value
        return 0;
}
```

If we initialize data members inside constructors memeber initializer list then it gets executed as per order of data member declarations inside class.For example:

```
class CTest
{
private:
        int _x;
        int _y;
```

```
        int _z;
public:
        CTest(void) : _z(_y), _x(10), _y(_x)        //ctor's member initializer list
        {       }
};
int main(void)
{
        CTest test;        //10,10,10
        return 0;
}
```

we can not initialize array inside constructors member intializer list. It is limitation of constructors member intializer list.
we can use constrctors member initializer list and constructor body together.In this case run time evironment executes constructors member intializer list first and then constructors body.

Note:Generally we should use constructor's member initializer to initialize data members.In case of modularity constructors member initializer list must appear in definition part only.

Default argument:

```
int sum(int num1, int num2)
{
        return num1 + num2;
}
int sum(int num1, int num2, int num3)
{
        return num1 + num2 + num3;
}
int sum(int num1, int num2, int num3, int num4 )
{
        return num1 + num2 + num3 + num4;
}
int main(void)
{
        int result = 0;
        result = sum(10, 20);
        result = sum(10, 20, 30);
        result = sum(10, 20, 30, 40);
        return 0;
}
```

if we consider above functions then its implimentation is logically equivalant.Since all the function having same name and same type of parameters then we can reuse its implementation by assigning some default values.For example

```
int sum(int num1, int num2, int num3 = 0, int num4 = 0 )
```

```
{
        return num1 + num2 + num3 + num4;
}
int main(void)
{
        int result = 0;
        result = sum(10, 20);
        result = sum(10, 20, 30);
        result = sum(10, 20, 30, 40);
        return 0;
}
```

Default values assign to the parameters of function is called default argument and function parameter is called optional parameter.In above code 0 is default argument and num3 and num4 are the optional parameters.We can not assign default arguments randomly.If we want to assign default argument to the parameter then it is necessary to assign default arument to all its right side parameters.

We can assign default arguments to parameters of any member function as well as non member function.

Note:In case of modularity, we must specify default arguments in declarations part only.

Constructor Chaining:

Process of calling constructor from another constructor is called constructor chaining.If we want to reuse implementation of existing constructor then we can use this feature.Since c++ support default argument, it do not support constructor chaining.

In C++ we can not call constructor with pointer or object explicitly.But it is possible to call to constructor explicitly.

l-value(Locator Value):

Expressions that refer to memory locations are called "l-value" expressions. An l-value represents a storage region's "locator" value, or a "left" value, implying that it can appear on the left of the equal sign (=). L-values are often identifiers.

In following example, x is an l-value because it persists beyond the expression that defines it.

```
int main()
{
  int x = 3 + 4;
  cout << x << endl;
}
```

r-value(Reference Value)

An rvalue is a temporary value that does not persist beyond the expression that uses it. All l-values are r-values but not all r-values are l-values.

In above example, the expression 3 + 4 is an r-value because it evaluates to a temporary value that does not persist beyond the expression that defines it.

Readonly Members:

Once intialized, if we dont want modify state of an object then we should declare variable/object as a constant.In C++ it is compulsory to initialize constant variable.

Note:Generally we should declare function parameter as a constant.

Constant Data Member:

if we dont want to modify state of the data member inside any member function of the class(including contructor) then we should declare data member as a constant.

Note:It is compulsory to initialize constant data member inside constructors member initializer list.

```
class Math
{
private:
        const double PI;  //constant data member
public:
        Math(void) : PI(3.14)
        {
                //this->PI = 3.142;          //Not allowed
        }
};
```

Constant Member function:

If we dont want to modify state on current object inside member function then we should declare member function as constant.We can declare only such function as a constant which gets this pointer. Since global function do not get this pointer, we can not declare global function as a constant.

Non constant member function always get this pointer of following type:

```
ClassName* const this;
```

Constant member function always get this pointer of following type:

```
const ClassName* const this;
```

As shown in above declaration,if we declare member function as constant then current object is considered as const object inside that member function.

```
class CTest
{
private:
        int _number;
public:
        CTest(void) : _number(10)
        {       }
        //CTest* const this = &testInstance;
        void Increment()
        {
                this->_number = this->_number + 1;          //allowed
        }
        //const CTest* const this = &testInstance;
        int GetNumber(void)const
```

```
        {
                //this->_number = this->_number + 1;        //Not allowed
                return this->_number;
        }
};
int main(void)
{
        CTest testInstance;
        return 0;
}
```

With the help of non constant object we can invoke constant as well as non constant member function.

mutable is a keyword in c++, which allows us to modify state of non constant data member inside constant member function.

```
class CTest
{
private:
        const int _num1;
        const int _num2;
        mutable int _count;
public:
        CTest(const int num1 = 0, const int num2 = 0) : _num1(num1), _num2(num2)
        {
                this->_count = 0;
        }
        void Print(void)const
        {
                this->_count = this->_count + 1;    //allowed
                cout << "Count   :       " << this->_count << endl;
                cout << "Num1    :       " << this->_num1 << endl;
                cout << "Num2    :       " << this->_num2 << endl;
        }
};
int main(void)
{
        CTest testInstance;
        testInstance.Print();
        testInstance.Print();
        return 0;
}
```

We can declare object of class as a constant. Using constant object we can call only constant member function.

const, volatile and mutable is called Access modifier.

Reference:

typdef is feature of C language which allows us to create alias for the existing data type.

e.g.
```
typedef  unsigned int      size_t;
size_t size = sizeof( int );
```

Reference is feature of C++ which allows to create alias for existing object.

e.g.
```
int num1 = 10;
int& num2 = num1;
```

In above code, num2 is reference variable where num1 is referant variable.Once initialized we can not change referent of reference.

```
int num1 = 10;
int num2 = 20;
int& num3 = num1;        //num3 is reference variable for num1;
num3 = num2;             //Here we are not changing reference rather copy value from num2 into num3;
```

Since references are implicitly considered as a constant pointer, it is compulsory to initiaize references.Following code illustrate how references are implicitly const and pointer.

```
class CTest
{
private:
        double& num3;
public:
        //It is compulsory to initialize references in ctors member initilaizer list
        CTest(double& num2) : num3( num2 )
        {       }
};
int main(void)
{
        double num1 = 10;
        CTest testInstance( num1 );
        size_t size = sizeof(testInstance);    // 4 bytes
        return 0;
}
```

Even though data member is a type of double, size of an object is 4 bytes, it means that references are implicitly treated as a pointer. References need to initialize inside ctors member initilaizer list it means that references are implicitly treated as const.

```
int num1 = 10;
int& num2 = num1;
```

Now we can treat above statement as follows:

```
int num1 = 10;
int& num2 = num1;        //int* const num2 = &num1;
```

i.e. References are automatically dereferenced constant pointer variables.

In following code i have used constant in combination with reference.

```
int num1 = 10;
int& num2 = num1;
const int& num3 = num1;
```

Here we can read as well as modify state of num1 via num2 but we can not modify state via num3.

References are not designed to use independantly rather it is designed to pass and return object from function by reference.

Note:Generally we should not return local variable by reference.In this case we should declare local variable as static.

Points To Remember:

1.Reference is derived data type.

2.Reference is used to create alias for the object.

3.We can not change referent of the reference.

4.We can not create reference to reference.

5.We can not initialize reference to NULL.

6.We can not create reference to constant.

7.We can not create array of references.

8.We can create reference to pointer, array, function as well as object.

9.We can pass object to the function by value, by address and by reference.

10.With the help of reference we can not save the memory rather we can minimize complexities of pointer.

Test your understanding:

How will you create reference to array?

What is the difference between pointer and reference?

Macro: A macro is short form of macro instruction.

It is a rule or pattern that specifies how a certain sequence of characters should be mapped to a replacement output sequence of characters according to a defined procedure.In short, symbolic constant is also called macro.The mapping process that transforms a macro use into a specific sequence is known as macro expansion.Preprocessor is program which exapands the macro.

Preprocessor performs two main task:

1.Expands the macro and

2.Removes the comments which included in .c file

There are two types of macros, object-like and function-like. Object-like macros do not take parameters; function-like macros do.

Consider the example of object like macro

```
#define  PI        3.142
```

Consider the exaple of function like macro

```
#define SQUARE( number )        number * number
int main(void)
{
```

```
        int number = 5;
        cout << SQUARE(number) << endl;
        return 0;
}
```

Several identifiers are predefined, and expand to produce special information.

__LINE__ A decimal constant containing the current source line number.

__FILE__ A string literal containing the name of the file being compiled.

__DATE__ A string literal containing the date of compilation, in the form "Mmmm dd yyyy"

__TIME__ A string literal containing the time of compilation, in the form "hh:mm:ss"

Inline Function:

Giving call to the function is always overhead to the compiler because each time a function is called, a significant amount of overhead is generated by the calling and return mechanism. Typically, arguments are pushed onto the stack and various registers are saved when a function is called, and then restored when the function returns. The trouble is that these instructions take time.

An inline function is one for which the compiler copies the code from the function definition directly into the code of the calling function rather than creating a separate set of instructions in memory. Instead of transferring control to and from the function code segment, a modified copy of the function body may be substituted directly for the function call. In this way, the performance overhead of a function call is avoided.

To declare function as inline we should use inline keyword.

```
inline int max( int num1, int num2 )
{
        return num1 > num2 ? num1 : num2;
}
```

If we implement member function inside structure/class then it is by default treated as inline.If we define functions globaly then we must specify inline keyword explicitly.Consider the following code.

```
class CPoint
{
private:
        int _xPosition;
        int _yPosition;
public:
        inline CPoint();
        inline void PrintRecord();
};
inline CPoint::CPoint()
{
        this->_xPosition = 0;
        this->_yPosition = 0;
}
inline void CPoint::PrintRecord()
{
```

```
        cout << "X Position      :       " << this->_xPosition << endl;
        cout << "Y Position      :       " << this->_yPosition << endl;
}
int main(void)
{
        CPoint pt1;
        pt1.PrintRecord();
        return 0;
}
```

We can declare constructor as inline only.

The inline specifier is only a suggestion/request to the compiler that an inline expansion can be performed; the compiler is free to ignore the suggestion.

In following condition, function can not be classified as inline
1.If function contains a code which leads to larger executables.
2.If we implement function using recursion.
3.If we implement function using loop.

Note:If we declare function as inline then we can not divide class declaration and definition in multiple files.
Test your understanding:
Why inline functions are hazardous?

string class:
C as well as C++ programming language supports strings but string is not a fundamental data type in it.Standard Template Library contains class 'basic_string' which allows us to handle the string. string is typedef of basic_string.

```
typedef basic_string string.
```

'basic_string' class is declared in string(#include<string>)header file.Lets see how to use it.

```
string name;
cout<<"Enter your name  :       ";
cin>>name;
cout<<"Name     :       "<<name<<endl;
```

Now no need to declare character array. In MS Visual Studio size of object os string class is 32 bytes.

Exception Handling:

'An exception is an event, which occurs during the execution of a program, that disrupts the normal flow of the program's instructions'.

To create an application, most of the times we take help of operating system resources.Following are some the resources we use frequently:

1. Memory
2. File
3. Socket etc.

Since these resources are very limited we should use it carefully.To manage these resources efficiently, we sholud use exception handling.In C++ we can handle exceptions with the help of following keywords:

1.try

2.throw

3.catch

if we want to inspect group of statements for exception then we should put such statement inside try block or try handler.If we want to use try block then it is necessary to provide at least one catch block.

Exceptions may be generated implicitly by the C++ runtime or programmer can generate it explicitly with the help of throw keyword. throw is jump statement.We can use throw statement inside try as well as catch block.

To handle exception thrown from try block we should use catch block or catch handler.Single try block may have multiple catch block but it must have at least one catch block.

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

void accept_record(int& number)
{
	cout << "Enter number	:		";
	cin >> number;
}
void print_result(const int& result)
{
	cout << "Result	:		" << result << endl;
}
int main(void)
{
	try
	{
		int num1;
		::accept_record(num1);

		int num2;
		::accept_record(num2);

		if (num2 == 0)
			throw string("Divide By Zero Exception");
```

```
            int result = num1 / num2;
            cout << "Result   :          " << result << endl;
        }
        catch ( string ex )
        {
            cout << ex << endl;
        }
        return 0;
}
```

We can write try catch block inside another try as well as catch block.Outer catch block can handle exception which are unhandled by the inner catch block but catch block of inner try can not handle exception which are unhandled by outer catch block.

If we do not handle thrown exception then C++ runtime implicitly gives call to 'std::terminate' function which implicitly gives call to the 'std::abort' function which stops execution of program abnormally.In this case we should write such a catch block which can handle all kind of exception thrown by user.A catch block which can handle all types of exception is called generic catch block.

```
        catch( ... )
        {
            /* Inside generic catch block*/
        }
```

If we want to use generic as well as spesific catch handler together then generic catch handler must be the last handler.

In C++, we can throw exception from outside try block via function.Consider the following example:

```
void accept_record(int& number)
{
        cout << "Enter number     :          ";
        cin >> number;
}
int calculate(int num1, int num2)
{
        if (num2 == 0)
            throw string("Divide By Zero Exception");
        return num1 / num2;
}
void print_result(const int& result)
{
        cout << "Result   :          " << result << endl;
}
int main(void)
{
        try
        {
            try
            {
```

```
                int num1;
                ::accept_record(num1);
                int num2;
                ::accept_record(num2);
                int result = ::calculate(num1, num2);
                ::print_result(result);
        }
        catch (string ex)
        {
                cout << ex << endl;
        }
    }
    catch (...)
    {
        cout << "Exception" << endl;
    }
    return 0;
}
```

In case of modularity, We can intimate to the user to handle spesific exceptions with the help of exception specification list.Consider the following code

```
int calculate(int num1, int num2) throw ( string )
{
    if (num2 == 0)
        throw string("Divide By Zero Exception");
    return num1 / num2;
}
```

throw followed by type of exception is called exception spesification list.

Note:We can write exception spcesification list inside declartion as well definition but ideally we should write it in declaration part only.

If we forget to specify type of exception in exception specification list then cath block do not handle such exception rather C++ runtime implicitly gives call to the std::unexpected function which gives call to the std:: terminate function. This feature do noty works in Microsoft Visual Studio.To test this feature we should write code in unix/linux.

If information required to handle the exception is incomplete at given place then we can rethrow the exception to its outer catch using throw keyword.

```
    try
    {
        try
        {
                int num1;
                ::accept_record(num1);
                int num2;
                ::accept_record(num2);
```

```
                int result = ::calculate(num1, num2);
                ::print_result(result);
        }
        catch (string ex)
        {
                throw;  //rethrow
        }
    }
    catch (string ex)
    {
        cout << ex << endl;
    }
    catch (...)
    {
        cout << "Exception" << endl;
    }
```

Sometimes we need to handle the exception by throwing different type of exception. It is called exception chaining.Consider the following example:

```
class LinkedList
{
    Node* head;
public:
    bool Empty(void)
    {
        //TODO
    }
    void DeleteFirst(void)throw( string )
    {
        if (this->Empty())
            throw string("List is empty");
        else
            //TODO
    }
};
class Stack
{
private:
    LinkedList collection;
public:
    void Pop(void)throw( string )
    {
        try
        {
```

```
                collection.DeleteFirst();
            }
            catch (string e)
            {
                throw string("Stack is full");          //Exception chaining
            }
        }
};
```

Stack Unwinding:

In the C++ exception mechanism, control moves from the throw statement to the first catch statement that can handle the thrown type. When the catch statement is reached, all of the automatic variables that are in scope between the throw and catch statements are destroyed in a process that is known as stack unwinding.

In stack unwinding, execution proceeds as follows:

1.Control reaches the try statement by normal sequential execution. The guarded section in the try block is executed.

2.If no exception is thrown during execution of the guarded section, the catch clauses that follow the try block are not executed. Execution continues at the statement after the last catch clause that follows the associated try block.

3.If a matching handler is still not found, or if an exception occurs during the unwinding process but before the handler gets control, the predefined run-time function terminate is called. If an exception occurs after the exception is thrown but before the unwind begins, terminate is called.

4.If a matching catch handler is found the process of unwinding stack begins. This involves the destruction of all automatic objects that were fully constructed—but not yet destructed—between the beginning of the try block that is associated with the catch handler and the throw site of the exception. Destruction occurs in reverse order of construction. The catch handler is executed and the program resumes execution after the last handler—that is, at the first statement or construct that is not a catch handler.

Types of Function:

An operation denotes a service that a class offers to its clients. Client typically performs five kinds of operations on an object.These functions allows us to write loosely coupled class.

1. Selector: an operation that accesses the state of an object but does not alter the state.Selector function is also called inspector function.

```
int GetReal( void )const
{
    return this->_real;
}
```

Generally we should declare inspector function as a constant.

2.Modifier: an operation that alters the state of an object.Modifier function is also called mutator function.

```
void SetReal( const int real )
{
    this->_real = real;
```

```
}
```

3. **Iterator:** an operation that permits all parts of an object to be accessed in some well-defined order.

4. **Constructor:** an operation that initializes an object.
5. **Destructor:** an operation that deinitializes an object.

Header Guard:

In case of repeated inclusion of headers, if we want to replace contents of header only once then we should use header guard.Header guard is also called #include guard or macro guard.
Header guards are implemented by using three preprocessor directives in a header file. Two are placed at the beginning of the file, before any pertinent code. The last is placed at the end of the file.

The first header guard line is of the form
#ifndef unique_symbol
and is followed immediately by the line
#define unique_symbol
The line
#endif

is placed at the end of the file.
The symbol used is not crucial, but it must be unique. It is traditional to use all capital letters for the symbol. Only letters, numbers and the underscore character can be used in the symbols. No other punctuation is allowed. A very common symbol is to use the name of the header file, converting the .h suffix to a _H.
Consider the Complex.h header file with header guard as follows:

```
#ifndef  COMPLEX_H
#define  COMPLEX_H

        class CComplex
        {
        private:
                int _real;
                int _imag;
        public:
                int GetReal()const;
                void SetReal( const int real );
                ...
                ...
        };

#endif
```

Multiple File Demo:

If we want to divide code into multiple files then class declaration should appear in header file and member function definition should appear in source file.We should define all global funcrtions in Main.cpp file.

To define functions globally, we should use following format:

ReturnType ClassName::MemberFunctionName(ParameterType parameterName)

Points To Remember:

1.We must specify default arguments only in declaration part only(.h file).
2.Constructors member initializer list must be appear in definition part only(.cpp).
3.const keyword must be appear in declaration as well as definition part(.h & .cpp).
4.static keyword must be appear in declaration part only(.h file).
5.Global definition for static data member must be appear in source file(.cpp).
6.Friend keyword must be appear in declaration part only.(.h)
7.virtual keyword must be appear in declaration part only(.h file).
8.We should not define pure virtual function inside source file.
9.Exception specification list can be appeared in declaration as well as definition.But it must appear in declaration.
10.User defined header files should be included using double quotes(#include"Complex.h").
11.In case of multiple files,we should use following sequence of header files.
   i.all C language standard header files.
   ii.all C++ language standard header files.
   iii.all user defined header files.

Modularity:

-Process of dividing complex system into modules which can be compiled separately, but which have connections with othe modules is called modularity.
-It is major pillar of oops.
-Modularity helps to minimize module dependancy.
-In C++, we can achieve logical modularity with the help of namespace and physical modularity with the help of multiple files.

Abstraction:

-Process of getting essential characteristics of an object is called abstraction.
-An abstraction focuses on the outside view of an object.
-Abstraction focuses on the essential characteristics of some object,relative to the perspective of the viewer.
kinds of abstractions include the following:
Entity abstraction =>An object that represents a useful model of a problem domain or solution domain entity
Action abstraction =>An object that provides a generalized set of operations, all of which perform the same kind of function.
Virtual machine abstraction => An object that groups operations that are all used by some superior level of control, or operations that all use some junior-level set of operations
Coincidental abstraction => An object that packages a set of operations that have no relation to each other

Encapsulation:

-Abstraction and encapsulation are complementary concepts: Abstraction focuses on the observable behavior of an object, whereas encapsulation focuses on the implementation that gives rise to this behavior.

-Encapsulation is the process of compartmentalizing the elements of an abstraction that constitute its structure and behavior or binding of data and code together is called encapsulation.

-Encapsulation hides the details of the implementation of an object.

Information Hiding :

It is the process of hiding all the secrets of an object that do not contribute to its essential characteristics. Generally we can hide data/information with the help of private or protected access specifier.

Data Security.

Process of giving control access to the mebers of class is called data security.To achieve data security first encapsulate variable and function inside a class and then the help of access specifiers give access to the data mebers using getter and setter method.

```
class CAccount
{
private:
        float _balance;
public:
        void Withdraw( float amount )
        {
                if( amount >= 100 )
                        this->_balance = this->_baance - amount;
                else
                        throw CException("Invalid amount");
        }
}
```

Function Overloading:

If implementation of function is logically equivalant then we should give same name to the function.If we want to give same name to the function then we must obey following rules:

1.number of parameters pass to the function must be different.

```
void sum( int num1, int num2 ){     }
void sum( int num1, int num2, int num3 ){   }          //allowed
```

2.If number of parameters are same then type of at least one parameter must be different.

```
void sum( int num1, int num2 ){     }
void sum( int num1, double num2 ){          }          //allowed
```

3.If number of parameters are same then order of type of parameter must be different.

```
void sum( int num1, float num2 ){  }
void sum( float num1, int num2 ){  }          //allowed
```

4.On the basis of return type we can not give same name to the function. i.e. return type is not considered.

```
void sum( int num1, int num2 ){     }
int sum( int num1, int num2 ){      }          //Not allowed
```

Process of writting such functions which is having same name but different signature is called function overloading.Functions which are taking part into function overloading are called overloded function.In case of overloading functions must be exist inside same scope.Except main function and destructor, we can overload any global function as well as member function.

Since ANSI has not defined any strict specification on return type, catching values from the function are optional. so return type is not considered at the time of overloading.

Test Your Understanding

1.What will be the output of following program?

```
void print(int number)
{
        cout << "int      :        " << number << endl;
}
void print(double number)
{
        cout << "double   :        " << number << endl;
}
1.      print( 10 );

2.      print( 10.5 );

3.      print( 10.5f );
```

2.What will be the output of following program?

```
void print(int number)
{
        cout << "int      :        " << number << endl;
}
void print(float number)
{
        cout << "float    :        " << number << endl;
}
1.      print( 10 );

2.      print( 10.5 );

3.      print( 10.5f );
```

3.What will be the output of following program?

```
void print(int number)
{
        cout << "int      :        " << number << endl;
}
```

```
void print(int& number)
{
        cout << "int&     :          " << number << endl;
}
1.      print( 10 );

2.      int number = 10;
        print( number );
```

4.What will be the output of following program?

```
void print(bool object)
{
        cout << "bool     :          " << object << endl;
}
void print(char object)
{
        cout << "char     :          " << object << endl;
}
1.      print( true );

2.      print( 'A' );

3.      print( 65 );
```

5.What will be the output of following program?

```
void print(int num1 = 0)
{
        cout << "int      :          " << num1 << endl;
}
void print(int num1, int num2 = 0)
{
        cout << "int      :          " << num1 << endl;
        cout << "int      :          " << num2 << endl;
}
1.      print(0, 10);

2.      print(30);

3.      print(10, 20 );

4.      print();
```

6.What will be the output of following program?

```
void print( char text[])
{
        cout << "char[]   :          " << text << endl;
}
void print(char* text)
{
        cout << "char*    :          " << text << endl;
}
1.      print("SunBeam");

2.      char name[] = "SunBeam";
        print(name);
```

7.What will be the output of following program?

```
void print( const char* text)
{
        cout << "const char*       :          " << text << endl;
}
void print(char* const text)
{
        cout << "char* const       :          " << text << endl;
}
1.      print("SunBeam");

2.      char name1[] = "SunBeam";
        print(name1);

3.      const char name2[] = "SunBeam";
        print(name2);
```

8.What will be the output of following program?

```
class Test
{
public:
        void Print( void )
        {
                cout << "void print( void )" << endl;
        }
        void Print(void)const
        {
                cout << "void print( void )const" << endl;
```

```
        }
};
1.      Test t1;
        t1.Print();

2.      const Test t2;
        t2.Print();
```

9.What will be the output of following program?

```
enum E
{
        eNumber = 10
};
void print(const int number)
{
        cout << "Number:         " << number << endl;
}
void print(E number)
{
        cout << "Number:         " << number << endl;
}
1.      print(10);

2.      const int number = 10;
        print(number);

3.      print(eNumber);
```

Name Mangling and Mangled Name:

C++ compiler generates encoded name by looking name of the function and type of argument pass to the function is called mangled name.

```
        void sum( int num1, int num2 );             //sum@@int,int

        double sum( int num1, float num2, double num3);          //sum@@int,float,double
```

Since ANSI has not defined any specification on mangled name, compiler vendors are free to generate mangled name.It means that mangled name may vary from compiler vendor to compiler vendor.It you try to access any element without definition, compiler will return you mangled name.

Process or algorithm which decides how to generate mangled name is called name mangling.Generating mangled name is job of compiler.

If we want to access functions implemented in .c file into .cpp then it is necessary to declare functions as extern "C".

For functions declared as extern "C", compiler do not generate mangled name.

C as well as C++ language allows us to reserve space for an object statically as well as dynamically.First we will see how to reserve space using C and then C++.
To allocate and deallocate space for an object dynamically we should use following functions.To use these functions we should use stdlib.h (cstdlib in C++) header file.

1. void* malloc(size_t size);
   - The malloc() function allocates size bytes and returns a pointer to the allocated memory.
   - The memory is not initialized.
   - If size is 0, then malloc() returns either NULL, or a unique pointer value.

2.void* calloc(size_t nmemb, size_t size);
   - The calloc() function allocates memory for an array of nmemb elements of size bytes each and returns a pointer to the allocated memory.
   - The memory is set to zero.
   - If nmemb or size is 0, then calloc() returns either NULL, or a unique pointer value.

3.void* realloc(void * ptr, size_t size);
   - The realloc() function changes the size of the memory block pointed to by ptr to size bytes.
   - The contents will be unchanged in the range from the start of the region up to the minimum of the old and new sizes.
   - If the new size is larger than the old size, the added memory will not be initialized.
   - If ptr is NULL, then the call is equivalent to malloc(size).
   - if size is equal to zero, and ptr is not NULL, then the call is equivalent to free(ptr).
   - If there is not enough available memory to expand the block to the given size, the original block is left unchanged, and NULL is returned.
   - If the area pointed to was moved, a free(ptr) is done.

4.void free(void * ptr);
   - The free() function deallocates the memory space pointed to by ptr, which must have been returned by a previous call to malloc(), calloc() or realloc().
   - If ptr is NULL, no operation is performed.

Standard directory for visual studio(C:\Program Files (x86)\Microsoft Visual Studio 12.0\VC\include) contains new.h (and new also) header file. It contains declarations of following functions:

```
struct nothrow_t{ // Empty structure          };

extern const nothrow_t nothrow;     // constant for placement new tag

void* operator new(size_t size) throw(bad_alloc);

void operator delete(void *) throw();
```

```
void* operator new[](size_t _Size)throw(bad_alloc);  // allocate array or throw exception

void  operator delete[](void *) throw();        // delete allocated array

void* operator new(size_t size, void *where) throw();

void* operator new(size_t _Size, const nothrow_t&) throw();

void*  operator new[](size_t _Size, const  nothrow_t&)throw();          // allocate array or return null pointer
```

In C++, to allocate memory dynamically we should use new operator and to deallocate that memory we should use delete operator. new implicitly calls operator new and delete implicitly calls operator delete function which are declared in new.h header file. Lets see how to use new and delete operator.

1. Allocating and deallocating space dynamically for single integer varaible:

```
try
{
        int* ptr  = new int;          //int* ptr = ( int* )::operator new( sizeof( int ) * 1 );
        ...
        ...
        delete ptr;                   //::operator delete( ptr );
        ptr = NULL;
}
catch( bad_alloc ex )
{
        cout<<ex.what()<<endl;
}
```

2.Allocating and deallocating space dynamically for single dimensional array of integer:

```
try
{
        int* ptr = new int[ 3 ];      //int* ptr = ( int* )::operator new[]( sizeof( int ) * 3 );
        ...
        ...
        delete[] ptr;                         //::operator delete[]( ptr );
        ptr = NULL;
}
catch( bad_alloc ex )
{
        cout<<ex.what()<<endl;
}
```

3.Allocating and deallocating space dynamically for multidimensional array of integer:

```
try
{
        int** ptr = new int*[ 2 ];          //int** ptr = ( int** )operator new[]( sizeof( int* ) * 2 );
        for( int index = 0; index < 2; ++ index )
                ptr[ index ] = new int[ 3 ]; //ptr[ index ] = ( int* )::operator new[]( sizeof( int ) * 3 );
        .....
        .....
        for( int index = 0; index < 2; ++ index )
                delete[] ptr[ index ];              //::operator delete( ptr[ index ] );
        delete[] ptr;                               //::operator delete( ptr );
        ptr = NULL;
}
catch( bad_alloc ex )
{
        cout<<ex.what()<<endl;
}
```

If new operator fails to allocate memory then C++ runtime implicitly throws bad_alloc exception.

We can allocate and deallocate memory for an object of user defined data type.In this case new implicitly calls constructor and delete implicitly calls destructor.

```
1. CPoint* ptr = new CPoint;        //parameterless constructor will call
2. CPoint* ptr = new CPoint();      //parameterless constructor will call//Recommended
3. CPoint* ptr = new CPoint( 10, 20 );      //parameterized constructor will call
4. delete ptr;              //destructor will call
```

Main advantage of new operator over malloc function is that it allows us to reserve space for an object on allocated as well as un-allocated memory area.To reserve space on allocated space we should use following form of new operator.It is called placement new operator.Consider the example;

```
char buffer[ 30 ];
int* ptr = new ( buffer ) int;      //int* ptr = ( int* )::operator new( sizeof( int ) * 1 , buffer );
....
....
//delete ptr;       //Not allowed
ptr = NULL;
```

if we want to place an object at a particular location in memory then we can use placement new operator.

If malloc fails to allocate memory then it returns NULL.new operator may behave like this if we use nothrow object.

consider the following example:

1. Following code demonstrate bad_alloc exception.

```
try
{
        int size = 1000000000;
        int* ptr = new int[ size ];
}
```

```
catch (bad_alloc ex)
{
        cout << ex.what();
}
//Output :        bad allocation
```

2. Following code demonstrate use of nothrow.

```
try
{
        int size = 1000000000;
        int* ptr = new (nothrow)int[size]; //int* ptr = (int*)::operator new[](sizeof(int)* size, nothrow);
        if (ptr == NULL)
                cout << "NULL" << endl;
}
catch (bad_alloc ex)
{
        cout << ex.what();
}
//Output :        NULL
```

nothrow_t is an empty structure which is declared in new.h header file and nothrow is an instance of nothrow_t struct.

Wild pointer: Uninitialized pointer is called wild pointer.

```
int* ptr; //wild pointer
```

Dangling pointer : A pointer which stores address of released memory or pointer which points to the invalid memory location is called dangling pointer;

```
int* ptr1 = new int();      //default value is 0
int* ptr2 = ptr1;   //two pointers for same location
*ptr2 = 10;
cout << "Value   :       " << *ptr2 << endl;
delete ptr2;
//Now ptr1 becomes dangling pointer
cout << "Value   :       " << *ptr2 << endl;
delete ptr1;        //Not allowed
```

To avoid dangling pointer we should deinitialize all pointers which are pointing to same location to NULL.

Memory leakage: If we reserve space in memory but no pointer is pointing to it then wastage of such memory is called memory leakage.

```
int* ptr = new int[3];
...
ptr = new int[5];  //this statement definitely loose 12 bytes.
```

To avoid memory leakage either we should use delete operator properly or we should use smart pointer.

Smart Pointer : If we treat an object as pointer then such object is called smart pointer.auto_ptr and unique_ptr are the predifined smart pointers. We will discuss smart pointer later.Consider the following example:

```
#include<iostream>
#include<memory>
using namespace std;
int main(void)
{
        auto_ptr<int> autoPointer(new int());//Here autoPointer is a object.
        *autoPointer = 10;
        cout << "Value    :          " << *autoPointer << endl;
        //Now no need to use delete operator here; auto_ptr will take care of it.
        return 0;
}
```

Test your understanding:

1. int* ptr = new int( );

2. int* ptr = new int(3);

3. int* ptr = new int[3];

What is the difference between malloc and new?

Destructor:

A member function of class that has the same name as the class and is prepended with ~ operator is called destructor.It is used to do clean up after an object of a class is no longer in the scope of program segment.Consider the syntax for destructor:

```
~CPoint( void )   //destructor for the CPoint class
{

}
```

There are two situations in which a destructor is called. The first is when an object is destroyed under "normal" conditions, e.g., when it goes out of scope or is explicitly deleted. The second is when an object is destroyed by the exception-handling mechanism during the stack-unwinding part of exception propagation.

If we do not define destructor for the class then compiler provides one destructor for the class by default.It is called default destrucotr.Default destructor always does the job of deinitialiation only.

On exit from function or block, Objects created in that function are destroyed by the compiler automatically.But an object might have an accumulated resources (dynamic memory, disk blocks, network connections etc.)during its life time.These resources are stored in the objects data members and are manipulated by the member functions.When the object is destroyed, it is necessary that the resources held by the object are released because object is no longer accessible.Because compiler isn't aware of the resources held by the object, it is responsibility of object to release such resources.To help an object in doing so, on exit from function,

destructor is called on all objects created statically.In other words, to release operating system resources held by the object we should write destructor inside class. Consider the following code snippet:

```
namespace NString
{
	class CString
	{
	private:
		unsigned int _length;
		char* _buffer;
	public:
		CString(unsigned int length = 0 )//ctor
		{
			//TODO
		}
		~CString(void)//dtor
		{
			if (this->_buffer != NULL)
			{
				delete[] this->_buffer;
				this->_buffer = NULL;
			}
		}
	};
}
```

Points To Remember:

1.To Prevent resource leak we should use destructor.
2.Destructor calling sequence is exactly opposite of constructor calling sequence(in array too).
3.Since destructor do not take any parameter we can not overload it.
4.We can declare destructor as inline and virtual only.
5.Even though destructor is designed to call implicitly, we can call it explicitly too.
6.if we try to create object using new then constructor gets call and if we try to delete that object using delete operator then destructor gets call.Consider following example:

```
CString* ptr = new CString( );	//Here ctor will call implicitly
...
delete ptr;	//Here dtor will call implicitly
```

If we create an instance with the help of placement new operator then we can not deallocate memory of an object using delete operator.If we do not use delete operator then destructor will not be call.But to avoid resource leak we need to call destructor explicitly.Consider the following code.

```
char arr[30];
```

```
CString* ptr = new ( arr )CString();
....
....
ptr->~CString();  //explicit call to dtor
ptr = NULL;
```

Array of Objects:

```
CPoint** ptr = new CPoint*[3];      //A
for (int index = 0; index < 3; ++index)
        ptr[index] = new CPoint();

.....
.....
for (int index = 0; index < 3; ++index)
        delete ptr[index];
delete[] ptr;
ptr = NULL;
```

Object Copy Semantics:

There are three different strategies for object coping.
1.Shallow Copy. (Also known as bitwise copy)
2.Deep Copy. (Also known as memberwise copy)
3.Lazy Copy. (Also known as Copy On Write(COW).)
The term shallow,deep & lazy copy originated in smalltalk and these terms are generally used to describe copy semantics

Shallow Copy:
At the time of copy operation, if we copy every member of source object into destination object as it is then such type of copy is shallow copy.Consider the following example:

```
1.      int num1 = 10;
        int num2 = num1;          //shallow copy.

2.      double num1 = 20.5;
        double num2;
        num2 = num1;              //shallow copy.

3.      CPoint pt1( 10, 20 );
        CPoint pt2;
        pt2 = pt1;                //shallow copy.

4.      CString str1("Sandeep");
        CString str2 = str1;      //shallow copy
```

Deep Copy:

If class data member holds operating system resource & if we want to create copy of the object then we need to rely on deep copy operation.Here i am going to consider memory for discussing deep copy.

If class contains at least one data member of pointer type, class contains user defined destructor which does job of deallocation and if we try to copy object into another object then instead of coping memory address from data member of source object into data member of destination object, we should allocate new memory for the data member of destination object and then we should copy contents from memory of source object into memory of destination object.Such type of copy is called deep copy.

Conditions to create deep copy.

1.Class should contain at least one data member of pointer type.

2.Class contains user defined destructor which does job of deallocation

3.We should create copy of object

Copy of the object gets created due to following reasons:

1.Initialization

2.Assignment

3.Passing object to the function by value.

4.Returning object function by value.

Steps to create deep copy:

1.Copy the size/length from data member of source object into data member of destination object.

2.Allocate new memory for the pointer data member of destination object.

3.Copy the contents from memory of source object into memory of destination object.

Where to create deep copy.:

1.In case of assignemnt we should create deep copy inside assignment operator function.[Operator Overloading]

2.In rest of the condition we should create deep copy inside copy constructor.

Copy Constructor:

A member function of class having same name as a class and which takes only one parameter of same type but as a reference is called copy constructor of the class.Copy constructor is a kind of parameterized constructor(Single Parameter Constructor).

General Syntax of copy constructor is as follows:

```
ClassName( const ClassName& other )
{
        //TODO:Copy operation
}
```

Initilaization is the process of storing known value in a variable during its creation.Job of copy constructor is to initialize newly created object from exisiting object.

The copy constructor gets invoked in following conditions:

1.If we pass an object to the function by value.

```
class CPoint
```

Prasanna
Wadekar


```
{
private:
        int _xPosition;
        int _yPosition;
public:
        //CPoint* const this = & pt1;
        //CPoint pointInstance = pt2;
        void Sum(CPoint pointInstance)     //Here for pointInstance, copy constructor will call
        {
                //TODO
        }
};
int main(void)
{
        CPoint pt1(10, 20);        //for pt1 parameterized ctor will call
        CPoint pt2(30, 40);        //for pt2 parameterized ctor will call
        pt1.Sum(pt2);
        return 0;
}
```

2.If we return an object from function by value.

```
class CPoint
{
private:
        int _xPosition;
        int _yPosition;
public:
        //CPoint* const this = & pt1;
        //CPoint pointInstance = pt2;
        CPoint Sum(CPoint pointInstance)//Here for pointInstance, copy constructor will call
        {
                CPoint tempInstance;//for tempInstance parameterless ctor will call
                        //TODO
                return tempInstance;
        }
};
int main(void)
{
        CPoint pt1(10, 20);        //for pt1 parameterized ctor will call
        CPoint pt2(30, 40);        //for pt2 parameterized ctor will call
        CPoint pt3;                //for pt3 parameterless ctor will call
        pt3 = pt1.Sum(pt2);//Here First Sum function & then for annonymous object copy constructor will call
        return 0;
```

```
}
```

3.If we initialize newly created object from existing object.

```
CString str1("Sandeep");
CString str2 = str1;          //Here for str2 copy constructor will call
```

or

```
CString str1("Sandeep");
CString str2( str1 );         //Here for str2 copy constructor will call
```

4. If we throw and catch object of user defined type.

```
class CException
{
private:
        int _length;
        char* _text;
};

int main(void)
{
        int num1 = 10;
        int num2 = 0;
        try
        {
                if (num2 == 0)
                        throw CException("Divide By Zero Exception");
                int res = num1 / num2;
        }
        catch (CException ex)      //Here for ex copy constructor will call
        {
                cout << "Exception" << endl;
        }
        return 0;
}
```

Note:It is however, not guaranteed that a copy constructor will be called in these cases, because the C++ Standard allows the compiler to optimize the copy away in certain cases, one example being the return value optimization

If class do not contain copy constructor then compiler provides one copy constructor for the class.Such copy constructor is called default copy constructor.Default behaviour of default copy constructor always creates shallow copy of an object.In case of initialization of an object, if there is need to create deep copy then we should define user defined copy constructor inside a class.

```
class CString
```

```
{
private:
        int _length;
        char* _buffer;
public:
        CString( const CString& other ) throw( bad_alloc )    //copy constructor with deep copy
        {
                this->_length = other._length;
                this->_buffer = new char[ this->_length + 1 ];
                strcpy( this->_buffer, other._buffer );
        }
};
```

If we pass an object to the function by address or by reference then copy constructor do not call.

Lazy Copy:

CString class which is declared above is fairly easy to understand and implement.What happens when objects of this class are used heavily as a function arguments and as a return values using the pass by value scheme?Since class uses deep copy semantics, if number of characters in the CString object is not too small, significanttime is spent in copying characters and in deleting the dynamically allocated memory.This also implies that object creation and destruction is expensive.The intention is that wherever character strings are needed,CString object should be used.But if cost of creation, copying, assignment and destruction of these CString objects is prohibitive, clients will refrain from using the class.

To optimize implementation,We could change implementation such that when a copy of a CString object is made, both the copies share the characters in the original string without actually copying them.But when one of the copies try to modify the contents then it should get seperate copy of characters without affecting other objects.This scheme of making a true copy of object when modification is attempted is called Copy-On-Write(COW) or Lazy Copy.

Concept of lazy copy or copy on write is based on reference counting mechanism.Reference counting is a technique that allows multiple objects with the same value to share a single representation of that value.

```
#include<cstring>
#include<iostream>
using namespace std;
class CString
{
private:
        struct SStringValue
        {
                unsigned _referenceCount;
                unsigned _length;
                char* _buffer;
        public:
                SStringValue(void)
                {
```

```
                this->_referenceCount = 1;
                this->_length = 0;
                this->_buffer = NULL;
        }
        SStringValue(const char* string)throw(bad_alloc)
        {
                this->_referenceCount = 1;
                this->_length = strlen(string);
                this->_buffer = new char[ this->_length + 1 ];
                strcpy(this->_buffer, string);
        }
        ~SStringValue(void)
        {
                delete[] this->_buffer;
                this->_buffer = NULL;
        }
    };
private:
    SStringValue* string;
public:
    CString(void)
    {
        this->string = new SStringValue();
    }
    CString( const char* str )
    {
        this->string = new SStringValue(str);
    }
    CString(const CString& other)//copy ctor
    {
        //First increment the reference count
        other.string->_referenceCount = other.string->_referenceCount + 1;
        //then share the resource
        this->string = other.string;
    }
    CString& operator=(const CString& other)
    {
        if (this == &other)
            return *this;
        this->~CString();
        other.string->_referenceCount = other.string->_referenceCount + 1;
        this->string = other.string;
```

```
		return *this;
	}
	CString& ToLower(void)  //Copy-On-Write Strategy
	{
		if (this->string->_referenceCount > 1)
		{
			char* str = this->string->_buffer;
			this->string->_referenceCount = this->string->_referenceCount - 1;
			this->string = new SStringValue(str);
		}
		char* str = this->string->_buffer;
		while (*str != '\0')
		{
			*str = tolower(*str);
			++str;
		}
		*str = '\0';
		return *this;
	}
	~CString(void)
	{
		this->string->_referenceCount = this->string->_referenceCount - 1;
		if (this->string->_referenceCount == 0)
			delete this->string;
		this->string = NULL;
	}
};

int main(void)
{
	CString str1("Sandeep Kulange");
	str1 = str1.ToLower();
	return 0;
}
```

Test Your Understanding:

1. CPoint pt1;
   CPoint* ptr = &pt1;

2. CPoint pt1;
   CPoint& pt2 = pt1;

3. CPoint pt1;
   CPoint pt2(pt1);

4. CPoint pt1;
   CPoint pt2 = pt1;

5. CPoint pt1,pt2;
   pt2 = pt1;

"Scope" decides the area or region in which we can access the portion of the program.Following are the types of scope:
1.Statement Scope
2.Block Scope
3.Function Scope
4.Class Scope
5.Namespace Scope
6.File Scope

"Lifetime" is the period during execution of a program in which a variable or function exists.
Storage classes govern the scope and lifetime of an object in C as well as C++.Following are the storage classes in C/C++.
1.auto
2.static
3.register
4.extern

Local variables are also called automatic variables becoz if we give call to the function then its memory gets managed automatically. Consider the following example:

```
void print(void)
{
        auto count = 0;
        ++count;
        cout << "Count   :          " << count << endl;
}
int main(void)
{
        ::print(); //1
        ::print(); //1
        ::print(); //1
        return 0;
}
```

If we want to persist value of the variable during function call then we should use static storage class.static variables are same as global variable but it is having limited scope.Consider the following example:

```
void print(void)
```

```
{
        static int count = 0;
        ++count;
        cout << "Count   :          " << count << endl;
}
int main(void)
{
        ::print(); //1
        ::print(); //2
        ::print(); //3
        return 0;
}
```

We can declare global function as a static.In this case we can not acess it outside the file.We can declare main function as static but compiler ignores it.

Non static data member is called instance variable and non static member function is called instance method.To access instance members(instance variable & method), inside member function we should use this pointer and inside non member function we should use object/instance of a class.

static data member is called class level variable and static member function is also called class level method.To access class level members(class level variable & method), inside member function as well as non member function we should use class name and scope resolution operator.

Static Data Member:

If we declare non static data member inside the class then every data member gets space inside object.

If we want to share value of any data memeber among all the objects of same type then we should declare data member as static.In other words all the objects of same type shares single copy of static data member.To declare data member as a static it is compulsory providre global definition for it.Because all the static variable/object gets space before starting execution of main.

Consider the following example:

```
namespace NTest
{
        class CTest
        {
        private:
                static int _count;
        };
        int CTest::_count = 0;//global definition
}
```

If we forget to provide global definition then linker produces "unresovled external symbol" error.

Note:Generally we should declare constant data member as static.

```
namespace NMath
{
        class CMath
```

```
        {
        private:
                static const float _PI;
        };
        const float CMath::_PI = 3.142f;//global definition
}
```

In c++, only static constant objects can be intialized at the time of declaration.

```
namespace NMath
{
        class CMath
        {
        private:
                static const int _count = 10;          //Here type must be static const
        };
}
```

Static Member Function:

To access non static data members we should define non static methods inside the class and to access static data members outside the class we should declare static method inside the class.Since static methods are designed to call using class name only, it do not get this pointer.Consider the following code:

```
namespace NInstanceCount
{
        class CInstanceCount
        {
        private:
                static int _instanceCount;
        public:
                CInstanceCount(void)
                {
                        CInstanceCount::_instanceCount = CInstanceCount::_instanceCount + 1;
                }
                static int GetInstanceCount(void)
                {
                        return CInstanceCount::_instanceCount;
                }
        };
        int CInstanceCount::_instanceCount = 0;      //global definition
}
int main(void)
{
        using namespace NInstanceCount;
        cout << "Instance Count    :          " << CInstanceCount::GetInstanceCount() << endl;    //0
        CInstanceCount obj1, obj2, obj3;
```

```
        cout << "Instance Count    :          " << CInstanceCount::GetInstanceCount() << endl;   //3
        return 0;
    }
```

Inside non static member function we can access both static as well as non static members.

```
    namespace NTest
    {
        class CTest
        {
        private:
            int _num1;
            static int _num2;
        public:
            CTest(void)
            {
                this->_num1 = 10;
                CTest::_num2 = 20;          //Now old value of num2 will be over written with 20.
            }
            void Print(void)
            {
                cout << "Num1   :          " << this->_num1 << endl;
                cout << "Num2   :          " << CTest::_num2 << endl;          //Allowed
            }
        };
        int CTest::_num2;          //First it will be initialized to 0
    }
```

Since static member function do not get this pointer we can not access non static members inside static member function directly.To access non static members inside static member function we should create object of the class.Consider the following code.

```
    namespace NTest
    {
        class CTest
        {
        private:
            int _num1;
            int _num2;
        private:
            CTest(void)
            {
                this->_num1 = 10;
                this->_num2 = 20;
            }
        public:
            void Print(void)
```

```
                {
                        cout << "Num1   :        " << this->_num1 << endl;
                        cout << "Num2   :        " << this->_num2 << endl;
                }
                static void Invoke()
                {
                        CTest test;         //Instantiation
                        test.Print();       //Now it is allowed
                }
        };
}
int main(void)
{
        using namespace NTest;
        CTest::Invoke();
        return 0;
}
```

We can not declare static member function as constant, volatile or virtual.

If we dont want to modify state of the current object inside member function then generally we declare member function as constant.Since static member functions are not designed to call using object we can not declare static member function as constant.

Even though static members are designed to call using class name, it is possible to access static members using object name.

Note:Generally we should use classname and scope resolution operator to access static members.

Design Pattern And Its Classification:

Design patterns are not algoritms or components.The design patterns are language independant strategies for solving common object oriented design problems. Here is the classification of design patterns which is listed in book of GOF.

1. Creational Design Patterns
   - i.Abstract Factory
   - ii.Builder
   - iii.Factory Method
   - iv.Prototype
   - v.Singleton
2. Structural Design Patterns
   - i.Adapter
   - ii.Bridge
   - iii.Composite
   - iv.Decorator
   - v.Facade
   - vi.Flyweight
   - vii.Proxy
3. Behavioural Design Patterns
   - I.Chain of responsiblity

ii.Command
iii.Interpreter
iv.Iterator
v.Mediator
vi.Memento
vii.Observer
viii.State
ix.Strategy
X.Template method
xi.Visitor

Singleton class: A class which allows us to create only one instance of it and which provides global point of access to it is called singleton class.

```
namespace NSingleton
{
        class CSingleton
        {
        private:
                static CSingleton* ptrSingletonInstance;
        private:
                CSingleton()
                {       }
        public:
                static CSingleton* GetInstance()
                {
                        if (CSingleton::ptrSingletonInstance == NULL)
                                CSingleton::ptrSingletonInstance = new CSingleton();
                        return CSingleton::ptrSingletonInstance;
                }
        };
        CSingleton* CSingleton::ptrSingletonInstance = NULL;
}
```

Test Your Understanding:

1.Why static member function do not get this pointer?

2.Why we can not declare static member function as a constant.

3.Observe the following code and decide wheather it is overloading or not?

```
class CTest
{
public:
        void print();
```

```
        static void print();
};
```

If we want to access private members of the class inside non member function then we can use any one of the following:

1. Inspector/mutator function
2. Friend function/class
3. Pointer.

Already we have discussed how to use inspector and mutator function.We will discuss pointer part later.

Friend function:

A non member function of a class which is designed to access private and protected members of the class is called friend function.friend is a keyword in C++.

1. Including main function we can declare any global function as friend.But generally we should not declare main function as a friend.Consider the following code:

```
namespace NTest
{
        class CTest
        {
        private:
                int _num1;
                int _num2;
        public:
                CTest(void) : _num1(10), _num2(20)
                {       }
                friend void sum(void);
        };
        void sum(void)
        {
                CTest test;
                int result = test._num1 + test._num2;
                cout << "Result   :          " << result << endl;
        }
}
int main(void)
{
        NTest::sum();
        return 0;
}
```

Friend declaration statement(friend void sum(void);)may appear in any region of the class(private, protected, public).

If we want to declare any function as a friend then class and function must be exist inside same namespace.

2. Member function of a class can be declared as friend function inside another class.

Note:It is trick.People generally preffered friend class.

```
class A
{
public:
        void Sum(void);
        void Sub(void);
};
class B
{
private:
        int num1;
        int num2;
public:
        B();
        friend void A::Sum(void);
        friend void A::Sub(void);
};
```

3. A Class can be declared as friend class of another class.

```
class A
{
public:
        void Sum(void);
        void Sub(void);
};
class B
{
private:
        int num1;
        int num2;
public:
        B();
        friend class A;
};
```

4. We can not declare mutual friend functions but we can declare mutual friend classes.

```
class B; //forward declaration
class A
{
private:
        int number;
public:
```

```
        void Print();
        friend class B;
};
class B
{
private:
        int number;
public:
        void Print();
        friend class A;
};
```

To access private or protected elements of the class inside another non member function or class, we declare function or class as a friend. With the help of friend function we can not access members of function so we can not declare mutual friend funtions.

we can declare single function as a friend inside multiple classes.

```
class CComplex
{
private:
        int _real;
        int _imag;
public:
        CComplex(void) : _real(10), _imag(20)
        {       }
        friend void sum();
};
class CPoint
{
private:
        int _xPosition;
        int _yPosition;
public:
        CPoint(void) : _xPosition(30), _yPosition(40)
        {       }
        friend void sum();
};
void sum()
{
        //TODO
}
```

Linked list is a collection of items/elements where each item/element is called node.Node is an object which may consist of either two parts or three parts depending on type of linked list.

A pointer which stores address of first node is called head pointer.If value of head is NULL then linked list is considered as empty.

A pointer which stores address of next node is called tail pointer.

There are two types of linked list:
1. Singly linked list.
2. Doubly linked list.

Singly Linked List : In a linked list if node consist of two parts i.e.
1. Data
2. A pointer which stores address of next node(next)
Then such type of linked list is called singly linked list.

Doubly Linked List : In a linked list if node consist of three parts i.e.
1.A pointer which stores address of previous node(prev)
2.Data
3.A pointer which stores address of next node(next) then such type of linked list is called doubly linked list.
In a singly linked list, if next pointer of last node stores NULL value then it is called linear singly linked list.
In a singly linked list, if next pointer of last node stores address of first node then it is called circular singly linked list.
In a doubly linked list, if previous pointer of first node and next pointer of last node stores NULL value then it is called linear doubly linked list.
In a doubly linked list, if previous pointer of first node stores address of last node and next pointer of last node stores address of first node then it is called circular doubly linked list.
Process of visiting nodes in linked list is called traversing.Generally to traverse linked list or any collection we should use iterator. We will discuss iterator in operator overloading.

```
namespace NList
{
        class CList;        //Forward declaration
        class CNode
        {
        private:
                int _data;
                CNode* _next;
        public:
                CNode(const int data = 0) : _data(data), _next(NULL)
                {       }
                friend class CList;
        };
        class CList
        {
        private:
```

```
        CNode* _head;
        CNode* _tail;
    public:
        CList(void) :_head(NULL), _tail(NULL)
        {       }
        bool Empty(void)const
        {
            return this->_head == NULL;
        }
        void AddFirst(const int data)throw( bad_alloc)
        {
            CNode* newnode = new CNode(data);
            if (this->Empty())
                this->_tail = newnode;
            else
                newnode->_next = this->_head;
            this->_head = newnode;
        }
    };
}
int main(void)
{
    using namespace NList;
    CList list;
    list.AddFirst(10);
    list.AddFirst(20);
    return 0;
}
```

Test your understanding:
Why friend function do not get this pointer?

Template:

C++ requires us to declare variables, functions, and most other kinds of entities using specific types. However, a lot of code looks the same for different types. Especially if we implement algorithms, such as quicksort, or if we implement a linked list or a binary tree for different types, the code looks the same despite the type used.

If language doesn't support a special feature for this, we only have bad alternatives:

1. we can implement the same behavior again and again for each type.
2. we can write general code using void*.
3. we can use special preprocessors.

Consider the following example:

```
void swap_object(int& object1, int& object2)
{
        int temp = object1;
        object1 = object2;
        object2 = temp;
}
void swap_object(string& object1, string& object2)
{
        string temp = object1;
        object1 = object2;
        object2 = temp;
}
```

In this case we can save programmers effort with the help of template.When we use a template, we can pass types as arguments.In other words "Parameterized type is also called template". we can write above code with the help of template.

```
template< typename Type >
void swap_object(Type& object1, Type& object2)
{
        Type temp = object1;
        object1 = object2;
        object2 = temp;
}
```

Now it is not a specific function it is generic function.More specific it is a Function template.
As shown in above code , template parameters must be announced with syntax of the following form:

template < comma-separated-list-of-parameters >

In our example, the list of parameters is typename Type.The keyword typename introduces a so-called type parameter.We can use any identifier as a parameter name, but using T is the convention.

```
int main( void )
{
        int num1 = 10;
        int num2 = 20;
        swap_object<int>( num1, num2 );   //Template argument list
        return 0;
}
```

At compile time Type will be replaced with int.Here swap_object<int>(...); indicated template argument list.
For historical reasons, you can also use class instead of typename to define a type parameter. The keyword typename came relatively late in the evolution of the C++ language.
With the help of template we can not save space or time rather we can save programmers effort only.
Not only function but also class can can be declared as template class.

```
template< class Type >
class CStack
{
private:
        int _top;
        int _size;
        Type* _arr;
public:
        CStack( void );
        bool Empty( void ) const;
        bool Full( void ) const;
        void Push( Type data );
        Type Peek( void );
        void Pop( void );
};
```

For class templates we can also define default values for template parameters. These values are called default template arguments.

```
template< class Type = int>
class CStack
{

};
int main(void)
{
        CStack<> s1;
        return 0;
}
```

Note:We can not divide implementation of template class into multiple files. declaration and definition must be in same file.

Operator Overloading:

In C as well as C++ we can use operators with the objects of fundamental types only.In C we can not use operators with the objects of user defined type directly(except sizeof).

In C++ if we want to use operators with objects then it is necessary to overload an operator.And to overload an operator it is necessary to define operator function.Now, we can define operator function as a member or non member function.

if we write pt3 = pt1 + pt2 then compiler can assume it like

```
pt3 = pt1.operator+( pt2 );
```

```
pt3 = operator+( pt1, pt2 );
```

operator is keyword in C++. If we want use + operator with objects then we should define logic of addition inside operator+() function. Now where to write definition of operator function is upto the programmer.

Consider the following code.

```
pt3 = pt1 + pt2;            //pt3 = pt1.operator+( pt2 );
```

in this case operator function will look like as follows

```
class CPoint
{
private:
        int _xPosition;
        int _yPosition;
public:
        //CPoint* const this = &pt1;
        //const CPoint& pointInstance = pt2;
        CPoint operator+( const CPoint& pointInstance )const
        {
                CPoint temp;
                temp._xPosition = this->_xPosition + pointInstance._xPosition;
                temp._yPosition = this->_yPosition + pointInstance._yPosition;
                return temp;
        }
};
```

And if we write `pt3 = pt1 + pt2;  //pt3 = operator+( pt1, pt2 );`
in this case operator function will look like as follows

```
class CPoint
{
private:
        int _xPosition;
        int _yPosition;
public:
        //const CPoint& pt1 = pt1;
        //const CPoint& pt2 = pt2;
        friend CPoint operator+(const CPoint& pt1, const CPoint& pt2)
        {
                CPoint temp;
                temp._xPosition = pt1._xPosition + pt2._xPosition;
                temp._yPosition = pt1._yPosition + pt2._yPosition;
                return temp;
        }
};
```

With the help of operator overloading we can not create new operator rather we can increase capability of the existing operators.
"Process of giving extension to the meaning of the operator is called operator overloading".

Limitations Of Operator Overloading:

1.We can not overload following operators as a member or non member function.

a. Member selection operator. (.)
b. Pointer to member selection operator. (.*)
c. Conditional/Ternary operator. (? :)
d. Scope resolution operator. (::)
e. sizeof operator.
f. typeid operator.
g. static_cast operator.
h. dynamic_cast operator.
i. const_cast operator.
j. reinterpret_cast operator.

2.We can not overload following operators as a non member function.

a. Assignment operator. (=)
b. Index/Subscript operator. ([])
c. Function call operator. (())
d. Dereferencing operator. (->)

3.We can change meaning of the operator with the help of operator overloading.

```
CPoint operator+( const CPoint& pointInstance )const
{
        CPoint temp;
        temp._xPosition = this->_xPosition - pointInstance._xPosition;
        temp._yPosition = this->_yPosition - pointInstance._yPosition;
        return temp;
}
```

First we will discuss overloading by implementing operator function as a member of class.

If we want to overload binary operator using member function then operator function should take only one parameter.
If we want to overload unary operator using member function then operator function should not take any paramater.

Overloading Arithmetic Operator:

```
1.      pt3 = pt1 + pt2;            //pt3 = pt1.operator+( pt2 );
2.      pt4 = pt1 + pt2 + pt3;      //pt4 = pt1.operator+( pt2 ).operator+( pt3 );
3.      pt3 = pt1 - pt2;            //pt3 = pt1.operator-( pt2 );
4.      pt3 = pt1 * pt2;            //pt3 = pt1.operator*( pt2 );
5.      pt3 = pt1 / pt2;            //pt3 = pt1.operator/( pt2 );
6.      pt3 = pt1 % pt2;            //pt3 = pt1.operator%( pt2 );
7.      pt2 = pt1 + 5;              //pt2 = pt1.operator+( 5 );
8.      pt2 = 5 + pt1;              //Not allowed;
class CPoint
{
public:
        CPoint operator+( const CPoint& pointInstance )const
```

```
        {
                CPoint temp;
                //TODO
                return temp;
        }
};
```

Overloading Relational Operator:

```
1.      bool result = pt1 == pt2;       //bool result = pt1.operator==( pt2 );
2.      bool result = pt1 != pt2;       //bool result = pt1.operator!=( pt2 );
3.      bool result = pt1 > pt2;        //bool result = pt1.operator>( pt2 );
4.      bool result = pt1 < pt2;        //bool result = pt1.operator<( pt2 );
5.      bool result = pt1 >= pt2;       //bool result = pt1.operator>=( pt2 );
6.      bool result = pt1 <= pt2;       //bool result = pt1.operator<=( pt2 );
class CPoint
{
public:
        bool operator==( const CPoint& pointInstance )const
        {
                return this->_xPosition == pointInstance._xPosition;
        }
};
```

Overloading Short Hand Operators:

```
1.      pt1 += pt2;             //pt1.operator+=( pt2 );
2.      pt1 -= pt2;             //pt1.operator-=( pt2 );
3.      pt1 *= pt2;             //pt1.operator*=( pt2 );
class CPoint
{
public:
        CPoint operator+=( const CPoint& pointInstance )
        {
                return *this;
        }
};
```

Overloading Unary Operators:

```
1.      pt2 = ++ pt1;           //pt2 = pt1.operator++( );
2.      pt2 = -- pt1;           //pt2 = pt1.operator--( );
3.      pt2 = pt1 ++;           //pt2 = pt1.operator++( 0 );
4.      pt2 = pt1 --;           //pt2 = pt1.operator--( 0 );
class CPoint
{
public:
        CPoint operator++( void ) //pre-increment
```

```
        {
                //TODO
        }
        CPoint operator++( int) //post-increment
        {
                //TODO
        }
};
```

Now, we will discuss overloading by implementing operator function as a non member of class.
If we want to overload binary operator using non member function then operator function should take two parameters.
If we want to overload unary operator using non member function then operator function should take one paramater.
Overloading arithmetic operator:

```
1.      pt3 = pt1 + pt2;        //pt3 = operator+( pt1, pt2 );
2.      pt4 = pt1 + pt2 + pt3;          //pt4 = operator+( operator+( pt1, pt2 ),pt3 );
3.      pt3 = pt1 - pt2;        //pt3 = operator-( pt1, pt2 );
4.      pt2 = pt1 + 5;                  //pt2 = operator+( pt1, 5 );
5.      pt2 = 5 + pt1;                  //pt2 = operator+( 5, pt1 );
class CPoint
{
public:
        friend CPoint operator+( const CPoint& pt1, const CPoint& pt2 );
};
CPoint operator+( const CPoint& pt1, const CPoint& pt2 )
{
        CPoint temp;
        //TODO
        return temp;
}
```

or

```
class CPoint
{
public:
        friend CPoint operator+( const CPoint& pt1, const CPoint& pt2 )
        {
                CPoint temp;
                //TODO
                return temp;
        }
};
```

Overloading Relational Operator:

```
1.      bool result = pt1 == pt2;       //bool result = operator==( pt1, pt2 );
```

```
class CPoint
{
public:
        friend bool operator==( const CPoint& pt1, const CPoint& pt2 )
        {
                //TODO
        }
};
```

Overloading Short Hand Operators:

```
1.      pt1 += pt2;                     //operator+=( pt1, pt2 );

class CPoint
{
public:
        friend CPoint operator+=( CPoint pt1, const CPoint& pt2 )
        {
                //TODO
                return pt1;
        }
};
```

Overloading Unary Operators:

```
1.      pt2 = ++ pt1;                   //pt2 = operator++( pt1 );
2.      pt2 = -- pt1;                   //pt2 = operator--( pt1, 0 );
class CPoint
{
public:
        friend CPoint operator++( CPoint& pt1 ) //pre-increment
        {
                //TODO
        }
        friend CPoint operator++( CPoint& pt1, int) //post-increment
        {
                //TODO
        }
};
```

There are two rules of thumb when implementing overloaded operators

1. Any operator that does not require an l-value is better implemented as non member function.
   a.Arithmetic operators
   b.Relational operators
2. Any operator that requires an l-value is better implemented as member function.
   a.Shorthand operators

b.Increment and decrement operators

c.Assignment,Index, Function call and dereferencing operator.

Overloading Insertion Operator (<<):

if we want to print state of an object on console then we should overload insertion operator.

```
cout<<pt1;              //operator<<( cout, pt1 );
cout<<pt1<<endl;        //operator<<( cout, pt1 )<<endl;
cout<<pt1<<pt2;         //operator<<( operator<<( cout, pt1 ), pt2 )
```

following is the general format to overload inserion operator.

```
class ClassName
{
        friend ostream& operator<<( ostream& out, const ClassName& referenceName )
        {
                //TODO
                return out;
        }
};
```

If we pass object of ostream class by value then its copy constructor will call.But copy constructor of ostream class is private so it is necessary to pass and return object of ostream class by reference.

Overloading Extraction Operator(>>):

If we want to read state for an object from console then we should overload extraction operator.

```
cin>>pt1;               //operator>>( cin, pt1 );
cin>>pt1, pt2;          //operator>>( operator>>( cin, pt1 ), pt2 );
```

Following is the general format to overload extraction operator.

```
class ClassName
{
        friend istream& operator>>( istream& in, ClassName& referenceName )
        {
                //TODO
                return in;
        }
};
```

If we pass object of istream class by value then its copy constructor will call.But copy constructor of istream class is private so it is necessary to pass and return object of istream class by reference.

Overloading Assignment Operator:

If we intialize newly created object from existing object then copy constructor gets call.

```
CPoint pt2 = pt1; //Here for pt2 object copy constructor will call.
```

If we assign existing object to another existing object then assignment operator gets call.

```
pt2 = pt1;              //Here assignment operator function gets call.
```

If we do not provide assignment operator function for the class then compiler provides one assignment operator function for that class by default.Such assignment operator function is called default assignment operator function.

Class contains following function by default:

1.default constructor
2.default destructor
3.default copy constructor
4.default assignment operator function.

Consider the assignment statement statement and its implicit call.

```
pt2 = pt1;                  //pt2.operator=( pt1 );

pt3 = pt2 = pt1;   //pt3.operator=( pt2.operator=( pt1 ) );
```

We can not overload assignment operator as non member function.Consider the following code:

```
class CPoint
{
private:
        ...
public:
        CPoint& operator=(const CPoint& other)
        {
                if (this == &other)
                        return *this;
                else
                {
                        this->_xPosition = other._xPosition;
                        this->_yPosition = other._yPosition;
                }
                return *this;
        }
};
```

In above program we have overloaded assignment operator and we have done shallow copy. If we want to do shallow copy then no need to overload assignement operator because default assignment operator do shallow copy.In case of assignment if we want to do deep copy or lazy copy then we should overload assignment operator.

Consider the following code:

```
class CString
{
private:
        int _length;
        char* _buffer;
public:
        CString& operator=(const CString& other)
        {
```

```
                if (this == &other)
                        return *this;
                else
                {
                        this->~CString();
                        this->_length = other._length;
                        this->_buffer = new char[this->_length + 1];
                        strcpy(this->_buffer, other._buffer);
                }
                return *this;
        }
};
```

Overloading Index/Subscript Operator:

We know that there are some disadvantages of language defined array:

1.We can not decide size for an array at runtime.

2.The built in array type does not have any kind of subscript range checking.

3.We can not copy arrays directly inside another array.

To avoid these limitations lets put array inside object and perform operations on object as array. Consider the following code.

```
class CArray
{
private:
        int _size;
        int* _arr;
public:
        CArray(const int size)
        {
                this->_size = size;
                this->_arr = new int[this->_size];
        }
        int& operator[](const int index)
        {
                if (index >= 0 && index < this->_size)
                        return this->_arr[index];
                throw CException("Array index out of bounds");
        }
};
```

Now we can specify size for an array at runtime.

```
        int size = 5;
        CArray a1( size );//valid;
```

Note: When operator [] is on the left hand side of =, It must return an address(l-value) and when it is on the right side of =, it must return a value (r-value).

```
int number = a1[ 2 ];          //int number = a1.operator[]( 2 );
```

In this case reference is act as value;

```
a1[ 2 ] = 300;                 //a1.operator[]( 2 ) = 300;
```

In this case reference is act as address.

In short, if we want to treat any object as a array then we should overload subscript or index operator.To overload index/subscript operator class should contain object of collection as a data member.

Overloading function call operator:

If we treat any function as a object then it is called function object. To create function object it is necesary to overload function call operator.

```
class CString
{
private:
        int _length;
        char* _buffer;
public:
        void operator()( const char* str )
        {
                this->~CString();
                this->_length = strlen(str);
                this->_buffer = new char[this->_length + 1];
                strcpy(this->_buffer, str);
        }
};
int main(void)
{
        CString str;
        str("Sandeep");   //here str is function object.
        return 0;
}
```

Overloading Dereferencing Operator:

if we want to use dot /member selection operator to access members of the class then left side oprand must be object.

```
e.g.    pt1.Print();
```

if we want to use arrow operator to access members of the class then left side oprand must be pointer of class.

```
e.g.    ptr->Print();
```

To avoid memory leakage we can use auto_ptr smart pointer. Lets write own version of auto_ptr class.

```
template< class Type >
class AutomaticPointer
{
private:
        Type* ptr;
public:
```

```
        AutomaticPointer(Type* ptr)
        {
                this->ptr = ptr;
        }
        Type* operator->(void)
        {
                return this->ptr;
        }
        ~AutomaticPointer(void)
        {
                delete ptr;
        }
};
int main(void)
{
        AutomaticPointer<CComplex> autoPtr(new CComplex());
        ptr->Print();
        return 0;
}
```

In above code autoPtr is object which act as a pointer, so it is called smart pointer.

Netsed class:If we define class inside scope of another class then it is called nested class.

```
class List
{
private:
        class Node          //Nested class.
        {
                ....
        };
private:
        Node* head;
};
```

Local class: If we define class inside scope of another function then it is called local class.

```
void f1( void )
{
        class SomeClass  //Local class.
        {
                ....
        };
}
```

Iterator is a smart pointer which is used to traverse the collection.It is a behavioural design pattern.

```
class CList
```

```
{
private:
        class CIterator;    //Forward declaration
        class CNode
        {
        private:
                int _data;
                CNode* _next;
        public:
                friend class CList;
                friend class CIterator;
        };
public:
        class CIterator
        {
        private:
                CNode* _ptrNode;
        public:
                CIterator(CNode* ptrNode)
                {
                        this->_ptrNode = ptrNode;
                }
                bool operator!=(CIterator& other)
                {
                        return this->_ptrNode != other._ptrNode;
                }
                int operator*(void)
                {
                        return this->_ptrNode->_data;
                }
                void operator++(void)
                {
                        this->_ptrNode = this->_ptrNode->_next;
                }
        };
private:
        CNode* _head;
        CNode* _tail;
public:
        CIterator Begin(void)
        {
                return CIterator(this->_head);
```

```
        }
        CIterator End(void)
        {
                return CIterator(NULL);
        }
};
int main(void)
{
        CList list;
        CList::CIterator itrStart = list.Begin();
        CList::CIterator itrEnd = list.End();
        while (itrStart != itrEnd)
        {
                cout << *itrStart << endl;  //itrStart : smart pointer
                ++itrStart;
        }
        return 0;
}
```

Conversion Function:

A member function of the class which is used to convert state of an object one type into another.In C++, there are three conversion functions:

1.Single parameter constructor.

```
int num1 = 10;
CPoint pt1 = num1;
```

In above statement, single parameter constructor is reponsible for converting value of num1 into pt1 so single parameter constructor is also called conversion constructor.

2. Assignment Operator:

```
int num1 = 10;
CPoint pt1;
pt1 = num1;        //pt1.operator=( CPoint( num1 ) );
```

In this case assignment operator is responsible for converting state of num1 into pt1 so it is called conversion function. To convert state of num1 into pt1, assignment takes help of annonymous object.If we want to keep control on anonymous object then we should declare single parameter constructor as explicit.

3. Type Conversion operator.

```
CPoint pt1( 10,20 );
int xPosition = pt1;        //        int xPosition = pt1.operator int();
```

In this case type conversion operator (operator int( )) is responsible for converting state of pt1 into xPosition so it is called conversion function.

Hierarchy and its Types:

-Hierarchy is a ranking or ordering of abstractions.

-With the help of hierarchy we can achieve code reusability.

-Four types of hierarchies are as follows:

1.has-a / part-of
2.is-a / kind-of
3.use-a
4.creates-a

Composition:

-If object is a component of another object then it is called composition.

-In C++, if we declare object of a class as data member inside another class then it represents composition programatically.

-Composition denotes has-a hierarchy

```
class CPerson
{
private:
        string _name;
        CDate _birthDate;
        CAddress _currentAddress;
};
```

-In case of composition, to call parameterized constructor of composed object, we should use constructor member initializer list.

Inheritance:

-It is journey from generalization to specialization.

-Without changing implementation of existing class, if we want extends meaning of the class then we should use inheritance.

-Parent class is called base class and child class is called derived class.

-Process of acquiring properties and behaviour of base class by the derived class is called inheritance.

-Except following members, all members inherit into derived class.

1. Constructor
2. Destructor
3. Copy Constructor
4. Assignment operator function
5. Friend function.

sizeof derived class object = sizeof all base class data members + sizeof all derived class data members

If name of base class and derived class method is same then derived class method hides implementation of base class method. In this case to access base class method we should use classname and scope resolution operator.

In following cases we can use scope resolution operator:

1.To access members of namespace.

2.To define member function globally

3.To accss static members of the class

4.To access base class members inside derived class.

If we create an instance of derived class then first base class and then derived class constructor gets call.Destructor calling sequance is exactly oposite.In case of inheritance, by default derived class constructor gives call to the parameterless constructor of base class.With the help of constructor's base initializer list we can call parameterized constructor of base class.It is a explicit call to the constructor.

If we use private, protected or public keyword at the time of inheritance to inherit base class inside derived class then it is called as mode of inheritance.

A.Public Mode of Inheritance:

If is-a relation exist between base type and derived type then we should use public mode of inheritance.

	Base class	Derived class	Indirect derived class	Non Member function
private	A	NA	NA	NA
protected	A	A	A	NA
public	A	A	A	A

B.private mode of inheritance

If has-a relation exist between base type and derived type then we should use private mode of inheritance.

	Base class	Derived class	Indirect derived class	Non Member function
private	A	NA	NA	NA
protected	A	A	NA	NA
public	A	A	NA	A using Base class object but NA using Derived class object

Note:We should use composition instead of private mode of inheritance.

C.protected mode of inheritance

	Base class	Derived class	Indirect derived class	Non Member function
private	A	NA	NA	NA
protected	A	A	A	NA
public	A	A	A	A using Base class object but NA using Derived class object

Object slicing.

Since data members of the base class inherit into the derived class, we can treat every derived class object as a base class object.If we assign derived class object to the base class object then value of base class portion exist inside derived class object gets copied into base class object.This process is called object slicing.

```
Derived derivedInstance;
Base baseInstance = dereivedInstance;        //Object slicing.
```

If we want to avoid object slicing then we should declare base class as a abstract.

Types of Inheritance:

1.Single inheritance:

If single base class is having single derived class then it is called single inheritance.

2. Multiple Inheritance:

If multiple base classes are having single derived class then it is called multiple inheritance.

3. Hierarchical Inheritance:

If single base class is having multiple derived classes then it is called hierarchical inheritance.

4.Multilevel Inheritance:

If single inheritance is having number of levels then it is called multilevel inheritance.

Combination of any two or more than two types of inheritance is called hybrid inheritance.

Diamond Problem:

```
class A
{
private:
        int num1;
public:
        A();
        A(int num1);
        void Print();
};
class B :public A
{
private:
        int num2;
public:
        B();
        B(int num1, int num2);
        void Print();
};
class C :public A
```

```
{
private:
        int num3;
public:
        C();
        C(int num1, int num3);
        void Print();
};
class D :public B, public C
{
private:
        int num4;
public:
        D();
        D(int num1, int num2, int num3, int num4);
        void Print()
        {
                A::Print();          //Ambiguity error
        }
};
```

Consider the above code. Since indirect derived class gets multiple copies of members of indirect base class, we can not access members of indirect base class inside indirect derived class directly such problem created by hybrid inheritance is called diamond problem.

To avoid diamond problem we should declare base class as a virtual.i.e. we should derive class B and class C from class A virtually. It is called virtual inheritance. Consider the following code.

```
class A
{

};
class B : virtual public A   //virtual inheritance
{

};
class C : virtual public A   //virtual inheritance
{

};
class D :public B, public C
{

};
```

Due to virtual inheritance, indirect derived class gets single copy of members of indirect base class.

Typing/Polymorphism:

-Polymorphism is a greek word.

-It is a minor pillar of oops.

-Ability of an object to take multiple forms is called polymorphism or one interface having many behaviour this phenomenon is also called polymorphism.

-Without changing interface, if we want to add new features into existing system then we should use polymorphism.

-With the help of polymorphism we can write maintainable code.

There are two types of polymorphism

1.Compile time polymorphism

2.Run time polymorphism

Compile Time Polymorphism:

-If call to the function gets resolved at compile time then it is called compile time polymorphism.

-It is also called as static polymorphism/early binding/false polymorphism/weak typing.

-We can achieve it with the help of

1.Function overloading.

2.Operator overloading.

3.Template

Runtime Polymorphism:

-If call to the function gets resolved at runtime then it is called runtime polymorphism.

-It is also called dynamic polymorphism/late binding/true polymorphism/strong typing.

-we can achieve it with the help of

1.Function overriding.

Upcasting:

-Process of converting pointer/reference of derived class into pointer/reference of base class is called upcasting.

-Process of storing address of derived class object into pointer of base class is also called upcasting.

-In case of upcasting mode of inheritance must be public.

-With the help of upcasting we can write maintainable code.

Downcasting:

-Process of converting pointer/reference of base class into pointer/reference of derived class is called downcasting.

Virtual Function and Function Overriding:

-A member function of a class which gets called depending on type of object rather than type of pointer is called virtual function.

-In other words member function of derived class which is designed to access using pointer/reference of base class is called virtual function.

-virtual functions are not designed to call using object,rather it is designed to call using pointer/reference of base class.

-A class which contains at least one virtual function is called polymorphic class.

-If we declare any function in base class as virtual then it remains virtual in derived class also.So it is optional to use virtual keyword in derived class.

-Process of reimplementing virtual function of base class inside derived class with same signature is called function overriding.

-For function overriding signature of base class and derived class function must be same.

-We can not declare static member function and constructor as a virtual.

Virtual Function Table and Pointer:

If class contains at least one virtual function then compiler creates one table to store address off virtual function declared inside that class.Such table is called virtual function table.Virtual function table gets created per class.

To store address of virtual function table, compilers adds one hidden pointer as data member inside the class.It is called virtual function pointer.Virtual function pointer gets generated per object.

We can declare destructor as a virtual but we can not declare constructor as virtual.

In case of upcasting, to call derived class destructor it is necessary to declare destructor in base class as a virtual.

Since contructor's are not designed to call expilictly with the help of pointer, we can not declare constructor as virtual.

RTTI:

-Process of getting information of any object at runtime is called run time type information/identification.

-To use RTTI it is necessary to use typeid operator.

-RTTI is mainly designed for polymorphic types only.

-If try to use RTTI on NULL pointer then it throws bad_typeid exception.

Advanced Casting Operators:

C++ supports four casting operators:

1. static_cast:

- If we want to do type conversion between compatible types then we should use static_cast operator
- In case of non polymorphic type, if we want to do downcasting then we should use static_cast operator.
- It do not check wheather type conversion is valid or not, It only checks inheritance relationship between source and destination type at compile time.

2. dynamic_cast

-In case of polymorphic type, if we want to do downcasting then we should use dynamic_cast operator.

-if we want to check inheritance relationship at runtime as well as if we want to check wheather type conversion is valid or not then we can use dyanamic_cast operator.

-In case of pointer, if dynamic_cast operator fails to do conversion then it returns NULL

-In case of reference, it dynamic_cast operator fails to do conversion then it throws bad_cast exception.

3.const_cast

-To convert const pointer/reference into non const Pointer/reference then we should use const_cast operator.

4.reinterpret_cast

-To do type conversion between incompatible types we should use reinterpret_cast operator.

Pure Virtual Function:

-If implementation of base class function is partially complete and if there is need to provide new implemetation in derived class then we should declare function in base class as virtual.If implementation of base class function is totally incomplete then we should declare function in base class as pure virtual.

-If we equate virtual function to zero then such virtual function is called pure virtual function.

-Pure virtual function may or may not have a body.But generally do not contain body.

-If class contains at least one virtual function then it is called abstract class.

-We can not crate object of abstact class but we can create pointer and reference of abstract class.

-Without pure virtual function we can not create abstract class in C++.

-Even though it is not possible to create instance of abstract class, we can write constructor as well as destructor inside abstract class.

Abstract Factory Method:

A method which hides object creattion process from end user is called factory method.It is a creational design pattern.

Appendix - I

Bjarne strostrup has not recommended any standard coding conventions for a program. We will use following conventions.

01. Global variable names usually begin with g.
    e.g. int gCount = 0;

02. Namespace names begin with N.
    e.g. namespace NComposition{ }

03. Class names begin with C.
    e.g. class CEmployee{ };

04. Structure names begin with S.
    e.g. struct SPoint{ };

05. Enum names begin with E.
    e.g. enum EColor{ };

06. Enum member names begin with e.
    e.g. enum EColor{ eRed, eGreen, eBlue };

07. Data member names begin with (_)underscore.
    e.g. int _numberOfPayingCars;

08. Member function names begin with an upper case alphabet.
    e.g. void SetRealNumber(...)

09. Local variable names begin with lower case alphabet.
    e.g. int numberOfPayingCars;

10. Name of the constant should be in upper case alphabet.
    e.g. const unsigned DAYS_IN_WEEK = 7;

11. Name of the class (Except C/S/E) and name of file should be same.
    e.g. if class name is CDate then file name should be Date.h or Date.cpp

12. Comments in header files precede the declaration.Short comments may appear on same line as code.

Appendix – II

Use following steps to create and execute Console Application in MSVS 2013.

Step 1: Start visual studio.[In Win 8/ 8.1 click on search-> type 2013-> select Visual Studio 2013]
[Or in any OS press Windows button & R then type devenv.exe and then click on OK button.]

Step 2: Click on File->New->Project->Installed->Templates->Other Project Types->Visual Studio Solutions->Blank Solution.

Step 3: Type name for Solution (.sln) and select location then click on ok:
Name: CPlusPlusSolution
Location: D:\Sandeep\Programs\...

Step 4: Click on View menu and select Solution Explorer.(ctrl + alt + L)

Step 5: In Solution Explorer select solution name (CPlusPlusSolution) and right click on it.

Step 6: Click on Add->New Project->Installed->Visual C++->Win2->Win32 Console Application.

Step 7: Type name for project(.vcxproj) and optionally select location then click on ok.
Name: SimpleHelloProject
Location: D:\Sandeep\Programs...\CPlusPlusSolution

Step 8: Click on Next->Application Type:Console Application->Additional Options->Empty project. Uncheck remaining options and click on Finish.

Step 9: In Solution Explorer select project name(SimpleHelloProject) and then Source Files folder.

Step 10: Right click on Source Files->Add->New Item->Installed->Visual C++->Code->C++ File.

Step 11: Type name for file(.h or .cpp) and optionally select location then click on Add.
Name: Main.cpp
Location: D:\Sandeep\Programs...\CPlusPlusSolution\SimpleHelloProject

Step 12: Write a program in file and save it.

Step 13: To compile and execute a code, first make it as start up project.
In Solution Explorer, select project and right click on it then select "Set as Startup Project" option.

Step 14: To build the project press ctrl + F7 and

Step 15: To execute application press ctrl + F5.

Appendix – III

To create a dynamic link library (DLL) project

1.Create Blank Solution and Save it with name "MathSolution".
2.In Solution Explorer select solution name("MathSolution) and right click on it.
3.Click on Add->New Project->Installed->Visual C++->Win2->Win32 Console Application.
4.Type name for project("MathDLL") and optionally select location then click on ok.
5.Click on Next->Application Type:DLL->Additional Options->Empty project. Uncheck remaining options and click on Finish.
6.Add MathFunction.h file in a projet and write a code as follows:

```
namespace NMath
{
	class MathFunction
	{
	public:
		static __declspec(dllexport) double Addition(double num1, double num2);

		static __declspec(dllexport) double Subtraction(double num1, double num2);
	};
}
```

__declspec(dllexport) modifier in the member function declarations enables the function to be exported by the DLL so that it can be used by other applications.

7.Add MathFunction.cpp file in a project and write a code as follows:

```
#include"MathFunction.h"
using namespace NMath;

__declspec(dllexport) double MathFunction::Addition(double num1, double num2)
{
	return num1 + num2;
}
__declspec(dllexport) double MathFunction::Subtraction(double num1, double num2)
{
	return num1 - num2;
}
```

Step 8: Build the project and check debug folder of solution.[dont check debug folder of Project].Not it contains two important files:

i. MathDLL.dll

Prasanna wadekar

ii. MathDLL.lib

Step 9: In same solution add new empty Win32 Console application["MathDLLTest"] to test function implmented in dll.
It must be Set as Start Up Project.

Step 10: Add reference of MathDLL.dll in console application[MathDLLTest].
Right click on MathDLLTest project-->Properties-->Common Properties-->References-->Add New Reference-->Project.
Select the dll and click on Ok.

Step 11: To reference the header files of the DLL, we must modify the included directories path.
To do this, in the Property Pages dialog box, expand the Configuration Properties node, expand the C/C++ node, and then select General. Next to Additional Include Directories, specify the path of the location of the MathFunction.h header file.

Step 12: Add Main.cpp file in MathDLLTest project and test the functionalities:

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

#include"MathFunction.h"
using namespace NMath;

int main(void)
{
	double result =  MathFunction::Addition(10.5, 20);
	cout << "Result   :		" << result << endl;
	return 0;
}
```

Step 13: Build and execute application.

Appendix – IV

Following are the steps to compile program from the command line.

Step 1: Write a simple "Hello World" program and save it "D:\SunBeam\SimpleHello" directory by the name Main.cpp.
Step 2: Open Command Prompt.[Start->Run->cmd.exe].
Step 3: Drag n drop 'vcvars32.bat' file on command prompt.
[C:\Program Files (x86)\Microsoft Visual Studio 12.0\VC\bin\vcvars32.bat]
Step 4: Compile the code using C++ compiler.
D:\SunBeam\SimpleHello> cl.exe
Main.cppoutput of this command=>Main.obj and Main.exe
Step 5: Execute the code
D:\SunBeam\SimpleHello>Main.exe
After pressing enter output will be "Hello World"

You should also try following commands:
1. Command "cl.exe -P filename.cpp" is used to generate intermediate file(.i).
D:\SunBeam\SimpleHello> cl.exe -P Main.cpp
Output of this command=>Main.i

2. Command "cl.exe -Fa filename.cpp" is used to generate assembly file(.asm).
D:\SunBeam\SimpleHello> cl.exe -Fa Main.cpp output of this command=>Main.asm

Test Series - Paper

Section-A

Q.1.

Which sentence does NOT contain a demonstrative adjective?

a. This museum has a display of tiaras.

b. They are delicate crowns covered with diamonds and pearls.

c. These brilliant examples were worn by royalty.

d. Look at those sparkling jewels!

Q.2.

Find correct sequence of sentence

P: appear in examinations

Q: many students from all over India

R: by different organizations

S: which are held on all India basis

a. QPSR

b. QRSP

c. QPRS

d. PRQS

Q.3.

Read the dialogues and choose the most appropriate options to complete the dialogues.Father : What? You crashed the car again?

Son : ----------

Father: I'm sure it wasn't. This is the third accident you have had this year.

Son : You're very angry now, dad. We had better talk about this later on.

a. I do apologize. I promise it won't happen again.

b. Was the car in good condition?

c. But it wasn't my fault. You've got to believe me.

d. Was it worth the money and time you had wasted?

e. Why do you ask? Don't you know it's too late now?

Q. 4.

Read the paragraph and answer question :

In the 1920s, new technology allowed filmmakers to attach to each film a soundtrack of speech, music and sound effects synchronized with the action on the screen. These sound films were initially distinguished by calling them talking pictures, or talkies. The next major step in the development of cinema was the introduction of color. While the addition of sound to film revolutionized the medium, quickly driving out silent movies, color was adopted more gradually. The public was relatively indifferent to color photography as opposed to black-and-white. But as color processes improved and became as affordable as black-and-white film, more and more movies were filmed in color after the end of World War II, as the industry in America came to view color an essential to attracting audiences in its competition with the television, which remained a black-and-white medium until the mid-60s. By the end of the 1960s, color had become the norm for filmmakers. It is stated in the reading that ----.

a. 1920s marked the end to the era of sound films

b. the transition to color films was not as rapid as the transition to sound films.

c. color movies did not appear until the mid- 60s

d. television didn't become popular until the mid-60s

e. color TVs were expensive initially

Q. 5. Read the paragraph answer question :
In the 1920s, new technology allowed filmmakers to attach to each film a soundtrack of speech, music and sound effects synchronized with the action on the screen. These sound films were initially distinguished by calling them talking pictures, or talkies. The next major step in the development of cinema was the introduction of color. While the addition of sound to film revolutionized the medium, quickly driving out silent movies, color was adopted more gradually. The public was relatively indifferent to color photography as opposed to black-and-white. But as color processes improved and became as affordable as black-and-white film, more and more movies were filmed in color after the end of World War II, as the industry in America came to view color an essential to attracting audiences in its ompetition with TV, which remained a black-and-white medium until the mid-60s. By the end of the 1960s, color had become the norm for filmmakers.
One can understand from the passage that ----.
a. black-and-white films are still a wonderful source of entertainment
b. the film industry fell into a serious crisis after the end of World War II
c. the' arrival of sound films ended the popularity of silent movies
d. the introduction of color to movies didn't help movie industry in its competition with television
e. the public was indifferent to sound films

Q. 6. Find the sentence that is closest in meaning to the one given for each Q.. French is the only language other than English spoken on five continents
a. French and English are the only languages that are spoken on five continents.
b. Unlike French, English is spoken on five continents.
c. French and English are spoken widely in official and commercial circles.
d. Before English, French was the only language spoken on five continents.
e. Worldwide, French is the most widely taught second language after English.

Q. 7. Choose the pair, which is most like the relationship expressed in the original pair in capital letters.
CAPTAIN : SHOAL:: ?
a. A lawyer : litigation
b. pilot : radar
c. doctor : hospital
d. solider : ambus

Q.8. Identify the numbered pair which does not exhibit the same relationship as the capitalized pair and mark its number as your answer.
FRIGHTEN : PETRIFY
a. Enamor : Protect
b. Sneer : Appreciate
c. Abbreviate : Interest
d. Humiliate : Mortify

Q.9.
Hematology
a. study of blood
b. study of human
c. study of nervous system
d. study of birds

Q. 10. Find out the error in any part of the sentence. For the Thanks giving reunion, relatives were sitting in the dining room, on the porch, and in the carport.
a. Thanksgiving, reunion
b. Were, sitting
c. Porch
d. No error

Q. 11. Find out the error.The boy in the kayak, whispered Sue "is the new football captain".
a. Boy, in the
b. New, football
c. Whispered Sue
d. No error

Q. 12. Choose the correct form of the verb to be - am/is/are It ---------- cold today.
a. am
b. is
c. are

Q. 13.
Find out the error. Oh I forgot to bring the cookies.
a. Oh
b. I, forgot
c. To, bring
d. No error

Q. 14.
A bag contains 10 balls numbered from 0 to 9. the balls are such that the person picking a ball out of the bag is equally likely to pick anyone of them. A person
picked a ball and replaced it in the bag after noting its number. He repeated this process 2 more times. What is the probability that the ball picked first is
numbered higher than the ball picked second and the ball picked second is numbered higher than the ball picked third?
a. 72/100
b. 3/25
c. 4/5
d. 1/6

Q. 15.
Six bells commence tolling together and toll at intervals of 2, 4, 6,8,10 and 12 seconds respectively. In 30 minutes, how many times do they toll together.
a. 4
b. 15
c. 10
d. 16

Q. 16. Which one of the following cannot be the square of natural number ?
a. 15186125824
b. 49873162329
c. 14936506225

d. 60625273287

Q. 17.
John gets on the elevator at the 14th floor of a building and rides up at the rate of 84 floors per minute. At the same time, Vinod gets on an elevator at the 58th floor of the same building and rides down at the rate of 92 floors per minute. If they continue travelling at these rates, then at which floor will their paths cross?

a. 38
b. 36
c. 32
d. 35

Q. 18.
Two fifth of one-fourth of three-seventh of the same by 10.The number is :
a. 94
b. 96
c. 188
d. None of these

Q. 19.
Of two numbers, 4 times the smaller one is less than 3 times the larger one by 5, If the sum of the numbers is larger than 6 times their difference by 6, find the two numbers.
a. 55 and 58
b. 23 and 28
c. 59 and 43
d. 65 and 67

Q. 20.
A large tanker can be filled by two pipes A and B in 60 minutes and 40 minutes respectively ,How many minutes will it take to fill the tanker from empty state if B is used for half the time and A and B fill it together for the other half?
a. 15 min
b. 20 min
c. 27.5min
d. 30 min

Q. 21.
Vikas bought paper sheets for Rs 7200 and spent. Rs 200 on transport. Paying Rs 600, he had 330 boxes made, which he sold at Rs 28 each. His profit percentage is
a. 15.5
b. 60
c. 40
d. None of these

Q. 22.
Put the verbs in brackets into the gaps. Form a Conditional sentence - type I. Only use the will-future in the main clauses. Mind the negations in the sentences. If she ---------- (not/to read) the novel, she ---------- pass (not/to pass) the literature test.
a. do not, will not

b. does not, is not going to
c. does not, will not
d. cannot read, won't

Q. 23.
If the manufacturer gains 10 %,the wholesale dealer 15 % and the retailer 25 %, then find the cost of production of a table if the retail price was Rs 1265?
a. Rs. 750
b. Rs. 800
c. Rs. 850
d. Rs. 900

Q. 24.
If man walks at the rate of 5 kmph, he misses a train by 7 minutes. however, if he walks at the rate of 6 kmph , he reaches the station 5 minutes before the arrival of the train, find the distance covered by him to reach the station.
a. 2 km
b. 4 km
c. 5 km
d. 6 km

Q. 25.
Two cyclist A and B cover a distance of 110 km, A reaches to the destination one hour before B. Even if B had given a start of 8 km, he would have reached 12 minutes late. Find the speed of A.
a. 12 km/h
b. 11km/h
c. 10 km/h
d. 8 km/h

Q. 26.
A train overtakes two persons who are walking in the same direction in which the train is going, at the rate of 2 kmph and 4 kmph and passes them completely in 9 and 10 seconds respectively. The length of the train is :
a. 72 meters
b. 54 meters
c. 50 meters
d. 45 meters

Q. 27.
A train 100 meters in length passes a milestone in 10 seconds and another train of the same length travelling in opposite direction in 8 seconds. The speed of the second train is :
a. 36 kmph
b. 48 kmph
c. 54 kmph
d. 60 kmph

Q. 28.
Today is 1st April. The day of the week is Wednesday. This is a leap year. The day of the week on this day after 3 years will be:
a. Saturday

b. Friday
c. Sunday
d. Tuesday

Q. 29.
This time tomorrow some of the people in this room _____ to their home countries.
a. will travel
b. will travelling
c. will be travelling
d. will have travelling

Q. 30.
Complete the following sentence I wished I
a. have a car
b. had a car
c. have had a car
d. had had a car

Q. 31.
Complete the following sentence The doctor advised the patient
a. that no to neglect his health
b. to not to neglect his health
c. not to neglect his health
d. do not neglect your health

Q. 32.
Complete the following sentence No sooner did the thief see the policeman
a. he ran away
b. he had run away
c. when he ran away
d. then he ran away

Q. 33.
Put in the correct preposition. Do you agree ------- staying in a foreign country?
a. to
b. for
c. with
d. at

Q. 34.
Complete the following sentence He asked me
a. that when i did go there
b. when did I go there
c. when I went there
d. When went I

Q. 35.

In the following items each passage consists of six sentences. The first and the sixth sentences are given in the beginning and labeled 1 and 6 respectively. The middle four sentences in each have been removed and jumbled up. These are labeled as P, Q R, and S. You are required to find the proper order for the four sentences and mark accordingly 1. Much of our adult behavior and our attitudes are determined by our upbringing.6 : Physiologists have studied these forces in depth.

P : But the process does not stop here.

Q : In particular by the effects of that small part of society which is our family.

R : As we grow, we are constantly increasingly affected by new forces such as the social pressure of our friends and the larger world of society.

S : The family and our early life have profound effect on our later life.

a. SRPQ
b. PRSQ
c. QPSR
d. QSPR

Q. 36.

In the following items each passage consists of six sentences. The first and the sixth sentences are given in the beginning and labeled 1 and 6 respectively. The middle four sentences in each have been removed and jumbled up. These are labeled as P, Q R, and S. You are required to find the proper order for the four sentences and mark accordingly 1 : Chankya, by his foresight and strength of character, built up the Mauryan Empire. 6 : It is not true as some people said that Sardar Patel was not particular about the mean as long as he achieved his end. P : They were men of iron. Q : Therefore, they were impervious to criticism and had implicit confidence in themselves. R: Both of them were guided by stern realism and would tolerate no deviations from the objectives which they pursued relentlessly. S: Similarly, Patel refashioned a compact India out of the fragments left by the British.

a. SQRP
b. SRQP
c. SRPQ
d. PQSR

Q. 37.

Tom said he would collect it when he ----- back .

a. came
b. should come
c. would come
d. comes

Q. 38.

As we now know how it works, we have the conviction ---- it works rather well.

a. as
b. how
c. why
d. that

Q. 39.

During the later years of the American Revolution, the Articles of Confederation government was formed. This government _____ severely from a lack of power. Each state distrusted the other and gave little authority to the central or federal government.

a. damaged
b. suffered
c. fail
d. diminished
e. none of these

Q. 40.
Oil is mostly _____ by trucks in Pakistan.
a. transfer
b. transmit
c. transport
d. travel
e. deliver

Q. 41.
Find the wrong part of the sentence The Ministry has directed banks / to do away with their / separate promotion policies, a move / strongly opposed by the officers' unions.
a. The Ministry has directed banks
b. to do away with their
c. separate promotion policies, a move
d. strongly opposed by the officers' unions

Q. 42.
Select the word out of the choices that is most appropriate the given meaning Weeping with short breaths _____
a. squeezing
b. staring
c. twinkling
d. sobbing

Q. 43.
A baby deer is called_____
a. cub
b. fawn
c. calf
d. colt

Q. 44.
Choose the correct possessive determiner -------- parents live in Maharashtra.
a. our
b. Us
c. We

Q. 45.
I am tired as I am working since 7 O'clock in the morning.
a. I was working
b. I have been working
c. I had been working
d. I will be working

Q. 46.
Choose the person who is be ing explained. _____ = the person who leads an orchestra
a. conductant
b. conductee
c. conducter
d. conductist
e. conductor

Q. 47.

_____ = someone who has retired from active working

a. retirant
b. retiree
c. retireer
d. retirer
e. retirist

Q. 48.

How many students in your class ------- from Srinagar?

a. comes
b. come
c. came
d. are coming

Q. 49.

Juhi: "What ------- in the evening?" Minu : "Usually I watch TV or read a book."

a. you doing
b. you do
c. do you do
d. are you doing

Q. 50.

When the RET instruction at the end of subroutine is executed,

a. the information where the stack is initialized is transferred to the stack pointer
b. the memory address of the RET instruction is transferred to the program counter
c. two data bytes stored in the top two locations of the stack are transferred to the program counter
d. two data bytes stored in the top two locations of the stack are transferred to the stack pointer

Section-B

Q.1
What is the output?
void main() { char ch[12]; strcpy(ch, "Sun Beam"); printf("%d", sizeof(ch)); }
a. 7
b. 8
c. 9
d. 12

Q.2
Which one of the following C operator is right associative?
a. ,
b. =
c. ^
d. ->

Q.3
What is the output of a code ??
void main() { long int a = scanf("%ld %ld %ld ", &a, &a, &a); printf("%ld", a); }
a. 0
b. 3
c. error
d. garbage value

Q.4
What is the output of the below given code :
void main() { int x = 61, y = 13; printf("%d", x>>x%y-6); }
a. 5
b. 7
c. 9
d. none of these

61/13 = 9

64 32 16 8 4 2 1
0 0 1 1 1 1 0

Q.5
On 32 bit compiler int *ptr = (int *)realloc(NULL,100); this is a same as...
a. int *ptr = (int *)malloc(100);
b. int *ptr = (int *)malloc(sizeof(int)* 25);
c. int *ptr = (int *)calloc(25, 4);
d. all of the above
e. none of these

Q.6
Continue statement used for ?
a. To handle run time error
b. To continue to the next line of code
c. To stop the current iteration and begin the next iteration from the beginning
d. None of above

Q.7
Wild pointer in C ?
a. if pointer pointing to more than one variable
b. if pointer is pointing to a memory location from where variable has been deleted
c. if pointer has not defined properly
d. if pointer has not been initialized

Q.8
What is the output of the below given code :
void main() { int *p, num; p=# *p=100; printf("%d", num); (*p)++; printf("%d", num); (*p)--; printf("%d", num); }
a. 100, 100, 100
b. 100, 101, 102
c. 100, 101, 100
d. 100, 101, 101

Q.9 _____ is a efficient data structure for flexible storage.
a. Static array
b. Structure
c. Queue
d. Link list

Q.10
If ptr is pointing to a first node of linked list, Then
void f(Node **ptr) { Node *t=*ptr; *ptr=*ptr->next; free(t); }
a. removes first element in the list.
b. removes next element in the list.
c. removes last element in the list
d. None of these

Q.11
A binary tree whose every node has either zero or two children is called
a. Complete binary tree
b. Binary search tree
c. Extended binary tree
d. None of above

Q.12
What is the output of the below given code :
#define mult(a, b) a*b void main() { int a=6, b=4; printf("%d", mult(b+2,a-3); }
a. -4
b. 8
c. 13
d. 18

Q.13
A _____ is a graph in which each connection has two directions.
a. undirected graph
b. bigraph
c. bidirectional graph
d. None of the above

Q.14
An algorithm that calls itself directly or indirectly is known as
a. Sub algorithm
b. Recursion
c. Polish notation
d. Traversal algorithm

Q.15
The complexity of merge sort algorithm is
a. O(n)
b. O(log n)
c. O(n2)
d. O(n log n)

Q.16
A data structure where elements can be added or removed at either end but not in the middle
a. Linked lists
b. Stacks
c. Queues
d. Dequeue

Q.17
Merge sort uses which technique?
a. Divide and conquer strategy
b. Backtracking approach
c. Heuristic search
d. Greedy approach

Q.18
The term "push" and "pop" is related to the
a. array
b. lists
c. stacks
d. all of above

Q.19
Which of the following ways below is a Inorder traversal?
a. Root->left sub tree->right sub tree
b. Root->right sub tree ->left sub tree
c. right sub tree->left sub tree->Root
d. left sub tree->Root->right sub tree

Q.20
A network that needs human beings to manually route signals is called....
a. Fiber Optic Network
b. Bus Network
c. T-switched network
d. Ring network

Q.21
In a ..... topology, if there are n devices in a network, each device has n-1 ports for cables.
a. Mesh
b. Star
c. Bus
d. Ring

Q.22
Devices on one network can communicate with devices on another network via a
a. File Server
b. Utility Server
c. Printer Server
d. Gateway

Q.23
What is the output of the below given code :
void main() { int x[]={1,5,4,6,2 }; printf("%d",(&x[4]-&x[1])); }
a. -3
b. 3
c. -5
d. 5

Q.24
Which of the following is/are Protocols of Application?
a. FTP
b. DNS
c. Telnet
d. All of above

Q.25
Why IP Protocol is considered as unreliable?
a. A packet may be lost
b. Packets may arrive out of order
c. Duplicate packets may be generated
d. All of the above

Q.26
How many pins does an RJ-45 modular plug have?
a. 8
b. 10
c. 11
d. 6

Q.27
What is the meaning of Bandwidth in Network?
a. Transmission capacity of a communication channels
b. Connected Computers in the Network
c. Class of IP used in Network
d. None of Above

Q.28

Which is a reliable connection-oriented service has two subparts
a. Message sequences
b. Byte streams
c. Both 1 & 2
d. None of these

Q.29
Which one of the following is a data link protocol?
a. Ethernet
b. point to point protocol
c. HDLC
d. all of the mentioned

Q.30
Which one of the following is the multiple access protocol for channel access control?
a. CSMA/CD
b. CSMA/CA
c. both (a) and (b)
d. none of the mentioned

Q.31
Find out the CLASS of IP address 4.5.6.7
a. Class A
b. Class B
c. Class C
d. Class D

Q.32
What is true about this pointer?
i. It is initialized automatically to the memory address of the object in which it is contained
ii. It is a pointer to an objects member function.
iii It can't be used explicitly in a program.
iv It is the same type as the object in which it is contained.
a. i,ii,iii
b. i,ii,iii,iv
c. i,iv
d. i,iii

Q.33
Deleting an object in a member function through 'this' result in
a. ambiguity
b. it is legal sometimes
c. compilation error
d. runtime error

Q.34
What is the output of the below given code :
int main() { void fun(); int i =1; while(i <= 5) { printf("%d",++i); if(i > 2) goto here ; } here: return 0;}void fun() { here : printf(" here");}

a. 23
b. 12
c. 12 here
d. Error

Q.35
Give output of the below given code :

```
int a = 2; int main(){ a += 2; ++::a = ++a;  cout << ::a;}
```

a. 5,5
b. 5,0
c. 6,5
d. Compile Error

Q.36
What is the output? class base { public : base() { cout<<" base "; } }; class derived : public base { public : derived () { cout<<" derived "; } }; void main() { derived d1; base b1; }
a. derived base
b. derived base derived
c. base derived
d. base derived base

Q.37
A big advantage of OOP over traditional programming is _____.
a. The objects are all declared Public
b. The convenience of giving all objects in a project the same name
c. The ability to reuse classes
d. None of the above

Q.38
Get and Set accessor methods must be _____ in order to allow other modules (forms or classes) to assign and retrieve their values.
a. Public
b. Private
c. Functions
d. Strings

Q.39
What is the output of the below given code :
#include<iostream> using namespace std; namespace Box1 { int a = 4; } namespace Box2 { int a = 13; } int main () { int a = 16; Box1::a; Box2::a; cout a; return 0; }
a. 4
b. 13
c. 16
d. compile time error

Q.40
class test { int day, mon; public : test() { cout<<"const"; } ~test() { cout<< "dest"; } }; void main() { test *t = new test; cout<<" process"; }
a. process const dest
b. const process
c. const process dest
d. Compile Error

Q.41
Functions can be declared with default values in parameters. We use default keyword to specify the value of such parameters.
a. True
b. False

Q.42
What is the difference between const int *ptr1 and int *const ptr2 ?
a. ptr1 and ptr2 both are constant pointer to the non constant integer .
b. ptr1 and ptr2 both are non constant pointer to the constant integer .
c. ptr1 is non constant pointer to constant integer and ptr2 is constant pointer to non constant integer
d. ptr1 is constant pointer to non constant integer and ptr2 is constant pointer to non constant integer

Q.43
Which of the following is not a type of constructor?
a. Copy Constructor
b. Default constructor
c. Friend constructor
d. Parameterised constructor.

Q.44
iterators are classified into _____
a. input iterator
b. forward iterator
c. Bidirectional iterator
d. Random access iterator
e. all of the above
f. none of these

Q.45
Give output of the below given code :
int main() { int main=7; printf("%d", main); }
a. Compile time error
b. Run time error
c. 7
d. Garbage value

Q.46
How many loop will be executed?
void main() { int i; for(i=0;i=3;i++) printf("%d", i); }
a. 3
b. 2
c. infinite
d. 0

Q.47
Give output of the below given code :
double p = 3.2, q; int j=2, r; q = (r=p/j)*2; printf("\n q = %g \t r = %g \n", (double)q, (double)r);
a. q = 2, r = 1
b. q = 1, r = 2
c. q = 3.2 , r = 1
d. q = 3.2, r = 1.6

Q.48

What is the output of the below given code :

int a[3][3]= { {1,2,3}, {4,5,6}, {7,8,9 } }; int *ptr[3] = { a[0], a[1], a[2] }; int *p = a[0]; main() { int i; for(i=0; i3;i++) printf("\n %d %d %d \n", a[i][2-i], *a[i], *(*(a +i)+i)); }

a. 3,1,1 5,4,5 7,7,9

b. 1,1,3 5,4,5 9,7,7

c. 1,3,1 4,5,5 7,9,7

d. 3,1,1 4,5,5 7,7,9

Q.49

Give output of the below given code :

main() { int x=3,y,z; y=z=4; printf("\n answer: : = %d\n", (z >;= y>= x)? 1:0); }

a. 1

b. 0

c. 3

d. 4

Q.50

2. Which of the following is correct regarding Class B Address of IP address.

a. Network bit 14, Host bit 16

b. Network bit 16, Host bit 14

c. Network bit 18, Host bit 16

d. Network bit 12, Host bit 14

Section-C

Q.1.What is the table that lists the required inputs for a given change of state?
a. Characteristic table
b. Truth table
c. Null table
d. Excitation table
e. Binary table

Q.2. Which of the following statements is true regarding HTTP?
a. It refers to resources using their Universal Resource Identifier (URI)
b. It does not carry browser Information to the server.
c. It is designed to route information based on content
d. Web browsers use only HTTP as a communication protocol with servers
e. It does not maintain any connection information on previous transactions

Q.3. WWW provides standardized access to.
a. GOPHER
b. TELNET
c. FTP
d. All of the above

Q.4 The phrase _____ describes viruses, worms, Trojan horse attack applets, and attack scripts.
a. malware
b. spam
c. phish
d. virus

Q.5 _____ is a small program embedded inside of a GIF image.
a. web bug
b. cookie
c. spyware application
d. spam

Q.6. Why is an exclusive-NOR gate also called an equality gate?
a. The output is false if the input are equal.
b. The output is true if the inputs are opposite
c. The output is true if the inputs are equal.

Q.7.Calculate the resolution of an 16-bit Analogue to Digital (A/D) converter with an input span of 10 V.
a. 1.5 mv
b. 6.5mv
c. 0.65 mv
d. 0.15 mv

Q.8.The instructions which copy information from one location to another either in the processors internal register set or in the external main memory are called
a. Data transfer instruction

b. Program control instruction
c. Input
d. Logical instruction

Q.9.The minimized form of the logical expression (A'B'C' + A' B C' + A' B C + A B C') is :
a. A'C' + BC' + A'B
b. A'C' + B'C + A'B
c. A'C + B'C + A'B
d. AC' + B'C + AB'

Q.10.
Relative Addressing Mode is used to write position independent code because
a. The code in this mode is easy to atomize
b. The code in this mode is easy to relocate in the memory
c. The code in this mode is easy to make resident
d. Code executes faster in this mode

Q.11. Excess 3 code is known as
a. weighted code.
b. redundancy code.
c. self complementing code.
d. algebraic code.

Q.12. If a counter having 10 flip flops is initially at 0, What count will if hold after 2060 pulses?
a. 000 000 1000
b. 000 000 1110
c. 000 001 1100
d. 000 000 1100

Q.13. Which of the following condition is true for determining overflow condition in 2's complement?
a. When adding two positive numbers gives a negative result or when two negatives give a positive result.
b. If sign bit (MSB) of result and sign bit of two operands are of different signs.
c. The 1 in the MSB position indicates a negative number after adding two positive numbers.
d. All of the above

Q.14. What is the 8-bit EBCDIC representation of alphabet M?
a. 1100100
b. 11001001
c. 11010100
d. 010001

Q.15.A _____ is a table, which consists of every possible combination of inputs and its corresponding outputs.
a. Last table
b. Truth Table
c. K- Map
d. Excitation Table

Q.16. The minimum time for which the input signal has to be maintained at the input of flip-flop is called _____ of the flip-flop.
a. Set-up time
b. Hold time
c. Pulse Interval time
d. Pulse Stability time (PST)

Q.17.
Stack is an acronym for _____
a. FIFO memory
b. FILO memory
c. Flash Memory
d. Bust Flash Memory

Q.18.
The minimum hardware required to construct a 3 to 8 decoder is using _____.
a. Two 2 to 4 decoders
b. Two 2 to 4 decoders and a two 1 to 2 decoders
c. Depends upon the technology (TTL, CMOS, etc.)
d. None of the above

Q.19. Dual of a+b.c is _____.
a. (a+b).(a+c)
b. c.(b+a)
c. a.(b+c)
d. (a+c).(a+b)

Q.20. To achieve mod-78, which counters should be connected in cascade?
a. MOD-70, MOD-8
b. MOD-50, MOD-28
c. MOD-13, MOD-6
d. MOD-10, MOD-8

Q.21. What is an optical storage?
a. Has faster access time than disk storage
b. Smaller capacity than CD-ROM
c. Greater capacity than DAT storage
d. Smaller capacity than DAT storage
e. Greater capacity than CD-ROM.

Q.22. Match the pairs and select the correct answer from the codes given below the list
List I
1 A shift register can be
2 A multiplexer
3 A decoder can
List II
1 for parallel to serial conversion
2 to generate memory can be used chip select
3 for parallel to serial conversion

CODES : ABC
a. 3 1 2
b. 2 3 1
c. 1 3 2
d. 1 2 3

Q.23. The resolution of D/A converter is approximately 0.4% of its full scale range. It is a _____
a. 8 bit converter
b. 10 bit converter
c. 12 bit converter
d. 16 bit converter

Q.24. State True(T) / False(F).
1. Flip-flops consist of two cross connected NAND or NOR gates
2. Computers cannot be considered as programmable calculators.
3. The Roman number system is a place value number system.
4. Number systems using digits 0 to 5 have a base 5.
a. TTTT
b. FTTF
c. TFFF
d. TTTF

Q.25. State True(T) / False(F).
1 Synchronous data transfer is less expensive than asynchronous data transfer.
2 The CPU program counter keeps track of the time of the next program instruction to be executed.
3 During the decoding stage the decoder in the program memory interprets the opcode that currently resides in the instruction register.
4 A program composed of symbolic statements in sometimes called a source program.
a. TTTT
b. FFTF
c. TFTF
d. TTTF

Q.26. The number of Boolean functions that can be generated by n variables is equal to
1. 2^n
2. 2^2^n
3. 2^n-1
4. 2^n

Q.27. Positive logic in a logic circuit is one in which
a. logic 0 and 1 are represented by 0 and positive voltage respectively
b. logic 0 and, -1 are represented by negative and positive voltages respectively
c. logic 0 voltage level is higher than logic 1 voltage level
d. logic 0 voltage level is lower than logic 1 voltage level.

Q.28. A number of 256 * 1 Kbits memory chips are available, how many such chips are required to provide the memory of 1024 Kbytes
a. 32
b. 4

c. 8
d. 16

Q.29. The instruction RET transfers the content of the top two locations of the stack to the _____.
a. Accumulator
b. H and L Register
c. B and C registers
d. Program counter

Q.30. The TRAP is one of the interrupts available its INTEL 8085. Which one of the following statements is true of TRAP?
a. It is level triggered
b. It is negative edge triggered
c. It is positive edge triggered
d. It is both positive edge triggered and level triggered

Q.31. How many General registers are in 8085 microprocessor?
a. 5
b. 4
c. 6
d. 8

Q.32.ADI
a. Add to Accumulator using carry Flag
b. Add to Accumulator
c. Add Immediate data to Accumulator
d. Add Immediate data to Accumulator Using carry

Q.33. The address bus flow in _____.
a. bidirectional.
b. unidirectional.
c. mulidirectional.
d. circular

Q.34. The 8085 is based in a _____ pin DIP
a. 40
b. 45
c. 20
d. 35

Q.35. Which is a 8 bit Microprocessor ?
a. Intel 4040
b. Pentium-I
c. 8088
d. Motorola MC-6801

Q.36. The address / data bus in 8085 is _____.

a. multiplexed
b. demultiplexed
c. decoded
d. encoded

Q.37. The microprocessor 8085 has _____ basic instructions and _____ opcodes.
a. 80, 246
b. 70, 346
c. 80, 346
d. 70, 246

Q.38. In 8086, the RD, WR, M/IO is the heart of control for a _____ mode
a. minimum
b. maximum
c. compatibility mode
d. control mode

Q.39. In 8086 the following has the highest priority among all the interrupts
a. NMI
b. DIV O
c. TYPE 255
d. OVER FLOW

Q.40. Which Segment is used to store interrupt and subroutine return address registers?
a. Stack Segment
b. Code Segment
c. Data Segment
d. Extra Segment

Q.41. CS connect the output of _____
a. encoder
b. decoder
c. slave program
d. buffer

Q.42.Instruction queue in 8086 is -------- byte long.
a. 6
b. 4
c. 8
d. 16

Q.43. The code used to boot up a computer is stored in
a. RAM
b. ROM
c. PROM
d. EPROM
e. EEPROM

Q.44. Which is not flag in 8086?
a. Carry flag
b. Overflow flag
c. Direction flag
d. Data Flag

Q.45. A micro program is sequencer perform the operation
a. read
b. write
c. execute
d. read and write
e. read and execute

Q.46. Memory access in RISC architecture is limited to instructions
a. CALL and RET
b. PUSH and POP
c. STA and LDA
d. MOV and JMP

Q.47. Who invented the microprocessor?
a. Marcian E Huff
b. Herman H Goldstein
c. Joseph Jacquard
d. All of above

Q.48. How is the 8085 microprocessor put in the wait state?
a. Lower the READY input
b. Lower the HOLD input
c. Raise the READY input
d. Raise the HOLD input

Q.49. What is meant by maskable interrupts?
a. an interrupt which can never be turned off.
b. an interrupt that can be turned off by the programmer.
c. an interrupt which can never be turned on.
d. an interrupt which can never be turned on or off.

Q.50. Which of the following describes the Mnemonic SPA?
a. Skip If (Address Resister) Positive
b. Skip If (Accumulator) Positive
c. Skip If (Adder-Subtractor) Positive
d. Skip If (Associative Mapping) Positive
e. Skip If (Assembler) Positive.

Answer Key - Section A

1.	B	2.	B	3.	C	4.	B	5.	C
6.	A	7.	C	8.	D	9.	A	10.	D
11.	C	12.	B	13.	A	14.	B	15.	D
16.	D	17.	D	18.	D	19.	C	20.	D
21.	A	22.	B	23.	B	24.	D	25.	B
26.	C	27.	C	28.	A	29.	C	30.	D
31.	C	32.	C	33.	C	34.	C	35.	D
36.	C	37.	D	38.	D	39.	B	40.	B
41.	D	42.	D	43.	B	44.	A	45.	B
46.	E	47.	B	48.	B	49.	C	50.	C

Answer Key - Section B

1.	D	2.	B	3.	B	4.	B	5.	D
6.	C	7.	D	8.	C	9.	D	10.	A
11.	C	12.	C	13.	D	14.	B	15.	D
16.	D	17.	A	18.	C	19.	D	20.	C
21.	A	22.	D	23.	B	24.	D	25.	D
26.	A	27.	A	28.	C	29.	D	30.	C
31.	A	32.	C	33.	B	34.	A	35.	A
36.	D	37.	C	38.	A	39.	C	40.	B
41.	B	42.	C	43.	C	44.	E	45.	C
46.	C	47.	A	48.	A	49.	B	50.	A

Answer Key - Section C

1.	D	2.	E	3.	D	4.	A	5.	A
6.	C	7.	D	8.	A	9.	A	10.	B
11.	C	12.	D	13.	D	14.	C	15.	B
16.	B	17.	B	18.	A	19.	C	20.	C
21.	A	22.	C	23.	A	24.	C	25.	B
26.	B	27.	D	28.	A	29.	D	30.	D
31.	C	32.	C	33.	B	34.	A	35.	D
36.	A	37.	A	38.	A	39.	A	40.	A
41.	B	42.	A	43.	B	44.	D	45.	E
46.	C	47.	A	48.	A	49.	A	50.	B