Tribhuvan University
Institute of Science and Technology

Course Title: Object Oriented Programming — Full Marks: 60
Course No: CSC161 — Pass Marks: 24
Level: B. Sc. CSIT First Year/ Second Semester — Time: 3 Hrs.

TU QUESTIONS-ANSWERS 2075

Section A

Long Answer Questions

Attempt any two questions. **[2*10=20]**

1. ***Explain the concept of user-defined to user-defined data conversion with the conversion routine located in the destination class.***

Ans: This conversion is exactly like conversion of basic to user defined data type i.e. one argument constructor is used. Conversion routine is defined as a one argument constructor in destination class and takes the argument of the source class type.

E.g.

```
classA objA;
classB objB;
objA=objB;
```

Here, objB is the source of the class classB and objA is the destination object of class classA. For this expression to work, the conversion routine should exist in class A (destination class type) as a one argument constructor as,

```
//source object class
Class classB
{
//body of classB
}
//destination object class
Class classA
{
private:
//....
public:
classA (class BobjB) //object of source class
{
//code for conversion from classB to classA
}
};
```

Complete Example in C++

```
class distance
{
int meter;
float centimeter;
public:
```

```
distance(int m, int c)
{
meter = m;
centimeter = c;
}
Int getmeter()
{
return meter;
}
Float getcentimeter()
{
return centimeters;
}
};
Class dist
{
int feet;
int inch;
public:
dist(int f, inti)
{
feet = f;
inch = i;
}
dist(distance d)
{
Int m,c;
m=d.getmeter();
c=d.getcentimeter();
feet= m*3.3;
inch= c*0.4;
feet=feet+inch/12;
inch= inch%12;
}
void display()
{
cout<<"Feet = "<<feet<<endl<<"Inches = "<<inches;
}
};
void main( )
{
clrscr();
distance d1(6,40);
dist d2;
d2=d1;
d2.display();
getche();
}
```


2. ***Depict the difference between private and public derivation. Explain derived class constructor with suitable example.***

Ans:

- **Private Modifier:** The scope of private members are restricted to its own class. Private members can't be accessed by the derived class or in main() function.
- **Public Modifier:** Public members can be accessed by its own class, derived class and in main() function.

	Own class	Derived class	main()
Private	√	×	×
Protected	√	√	×
Public	√	√	√

A constructor plays a vital role in initializing an object. An important note, while using constructors during inheritance, is that, as long as a base class constructor does not take any arguments, the derived class need not have a constructor function. However, if a base class contains a constructor with one or more arguments, then it is mandatory for the derived class to have a constructor and pass the arguments to the base class constructor. Remember, while applying inheritance, we usually create objects using derived class. Thus, it makes sense for the derived class to pass arguments to the base class constructor. When both the derived and base class contains constructors, the base constructor is executed first and then the constructor in the derived class is executed.

In case of multiple inheritance, the base class is constructed in the same order in which they appear in the declaration of the derived class. Similarly, in a multilevel inheritance, the constructor will be executed in the order of inheritance.

The derived class takes the responsibility of supplying the initial values to its base class. The constructor of the derived class receives the entire list of required values as its argument and passes them on to the base constructor in the order in which they are declared in the derived class. A base class constructor is called and executed before executing the statements in the body of the derived class.

```
// program to show how constructors are invoked in derived class
#include <iostream.h>
class alpha
{
private:
int x;
public:
alpha(inti)
{
x = i;
cout<< "\n alpha initialized \n";
}
Void show_x()
{
```

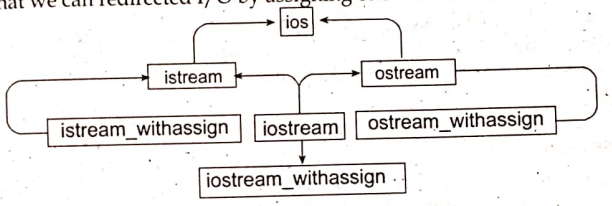
```
cout<< "\n x = "<<x;
}
};
class beta
{
private:
float y;
public:
beta(float j)
{
y = j;
cout<< "\n beta initialized \n";
}
Void show_y()
{
cout<< "\n y = "<<y;
}
};
class gamma : public beta, public alpha
{
private:
int n, m;
public:
gamma(int a, float b, int c, int d):: alpha(a), beta(b)
{
m = c;
n = d;
cout<< "\n gamma initialized \n";
}
Void show_mn()
{
cout<< "\n m = "<<m;
cout<< "\n n = "<<n;
}
};
void main()
{
gamma g(5, 7.65, 30, 100);
cout<< "\n";
g.show_x();
g.show_y();
g.show_mn();
}
```

Output:

```
beta initialized
alpha initialized
gamma initialized
x = 5
y = 7.65
m = 30
n = 100
```


3. ***Briefly explain the hierarchy of stream classes. Write a program that overloads extraction and insertion operators.***

Ans: All the stream classes are derived from the base class ios, which stores the state of the stream and handles error. The ios class has an associated streambuf object that acts as the buffer for the stream. The istream and ostream classes, devided from ios, are meant for input and output, respectively. The iostream class uses multiple inheritance to acquire the capabilities of both istream and ostream class and there for support both input and out. The classes iostream_withassign, ostream_withassign and istream_withassign are derived from istream, ostream and iostream, respectively, by adding the definition of the assignment operator (=) so that we can redirected I/O by assigning one stream to another.



C++ is able to input and output the built-in data types using the stream extraction operator >> and the stream insertion operator <<. The stream insertion and stream extraction operators also can be overloaded to perform input and output for user-defined types like an object. Here, it is important to make operator overloading function a friend of the class because it would be called without creating an object. Following example explains how extraction operator >> and insertion operator <<.

```
#include <iostream>
using namespace std;
class Distance
{
private:
int feet;          // 0 to infinite
int inches;        // 0 to 12
public:
// required constructors
Distance()
{
feet = 0;
inches = 0;
}
Distance(int f, int i)
{
feet = f;
inches = i;
}
friend ostream &operator<<( ostream &output, const Distance &D )
{
output << "F : " << D.feet << " I : " << D.inches;
return output;
```

```
}
friend istream &operator>>( istream  &input, Distance &D )
{
input >> D.feet >> D.inches;
return input;
}
};
int main()
{
Distance D1(11, 10), D2(5, 11), D3;
cout << "Enter the value of object : " << endl;
cin >> D3;
cout << "First Distance : " << D1 << endl;
cout << "Second Distance :" << D2 << endl;
cout << "Third Distance :" << D3 << endl;
return 0;
}
```

Short answer questions

Group B

Attempt any eight questions

4. ***Write a member function called reverseit() that reverses a string (an array of characters). Use a for loop that swaps the first and last character, then the second and next-to last characters and so on. The string should be passed to reverseit() as argument***

Ans:

```
#include<iostream.h>
#include<iomanip.h>
#include<conio.h>
#include<string.h>
void reverseit(char s[100])
{
int i, l;
char temp;
l=strlen(s);
int j=l-1;
for(i=0;i<l/2;i++)
{
temp=s[j];
s[j]=s[i];
s[i]=temp;
j--;
}
s[i]='\0';
cout<<s;
}
void main()
{
char st[100];
cout<<"Enter any string";
cin>>st;
```



```
reverseit(st);
getch();
}
```

5. ***What is the principle reason for using default arguments in the function? Explain how missing arguments and default arguments are handled by the function simultaneously?***

Ans: In C++, a function can be called without specifying all its arguments. But it does not work on any general function. The function declaration must provide default values for those arguments that are not specified. When the arguments are missing from function call, default value will be used for calculation.

```
#include<iostream.h>
float interest(int p, int t = 5, float r = 5.0);
void main()
{
float rate, i1,i2,i3;
int pr , yr;
cout<<"Enter principal, rate and year";
cin>>pr>>rate>>yr;
i1= interest(pr ,yr ,rate);
i2=interest(pr , yr);
i3=interest(pr);
cout<<i1<<i2<<i3;
return(0);
}
float interest(int p, int t, float r)
{
return((p*t*r)/100);
}
```

In the above program, t and r has default arguments. If we give, as input, values for pr, rate and yr as 5000, 10 and 2, the output will be

1000 500 1250

6. ***"An overloaded function appears to perform different activities depending the kind of data send to it". Justify the statement with appropriate example.***

Ans: Function that share the same name are said to be overloaded functions and the process is referred to as function overloading. I.e. function overloading is the process of using the same name for two or more functions. Each redefinition of a function must use different type of parameters, or different sequence of parameters or different number of parameters. The number, type or sequence of parameters for a function is called the function signature. When we call the function, appropriate function is called based on the parameter passed. Two functions differing only in their return type cannot be overloaded. For e.g.-

int add(int , int) and float add(int, int)

A function call first matches the declaration having the same number and type of arguments and then calls the appropriate function for execution.

```
#include<iostream.h>
float perimeter(float);
int perimeter(int,int);
int perimeter(int,int,int);
void main()
{
cout<<"Perimeter of a circle: "<<perimeter(2.0)<<endl;
cout<<"Perimeter of a rectangle: "<<perimeter(10,10)<<endl;
cout<<"Perimeter of a triangle: "<<perimeter(5,10,15);
return (0);
}
//function definitions
float perimeter(float r)
{
return(2*3.14*r);
}
int perimeter(int l,int b)
{
return(2*(l+b));
}
int perimeter(int a,int b,int c)
{
return(a+b+c);
}
```

In the above program, a function "perimeter" has been overloaded. The output will be as follows:

Perimeter of a circle 12.56
Perimeter of a rectangle 40
Perimeter of a triangle 30

7. ***Explain the default action of the copy constructor. Write a suitable program that demonstrates the technique of overloading the copy constructor.***

Ans: The copy constructor is a constructor which creates an object by initializing it with an object of the same class, which has been created previously. The copy constructor is used to –

- Initialize one object from another of the same type.
- Copy an object to pass it as an argument to a function.
- Copy an object to return it from a function.

If the class has pointer variables and has some dynamic memory allocations, then it is a must to have a copy constructor. The most common form of copy constructor is shown here;

```
Class name (const classname &obj)
{
// body of constructor
}
```

Example:

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



```
class A
{
int x, y;
public:
A()
{
cout << "Default constructor called!";
}
A(int a, int b)
{
cout << "Parameterized Constructor called!\n";
x = a;
y = b;
}
A(const A &old)
{
// old is the old object being passed
x = old.x; //This object's x to old object's x
y = old.y;
cout << "Copy Constructor called!\n";
}
void print()
{
cout << x << " " << y << "\n";
}
};
int main()
{
// Sample Code to show default constructor
A obj(10, 20); // making a object of class A  -- >Implicit
A obj2(obj);  // Copy Constructor called old object 'obj' is passed
obj2.print();
return 0;
}
```

8. ***Briefly explain types of inheritance need in object oriented programming.***

Ans: Inheritance (or derivation) is the process of creating new classes, called derived classes, from existing classes, called base classes. The derived class inherits all the properties from the base class and can add its own properties as well. The inherited properties may be hidden (if private in the base class) or visible (if public or protected in the base class) in the derived class. Inheritance uses the concept of code reusability. Once a base class is written and debugged, we can reuse the properties of the base class in other classes by using the concept of inheritance. Reusing existing code saves time and money and increases program's reliability. An important result of reusability is the ease of distributing classes. A programmer can use a class created by another person or company, and, without modifying it, derive other classes from it that are suited to particular situations.

Types of Inheritance in C++

Single Inheritance: In single inheritance, a class is allowed to inherit from only one class. i.e. one sub class is inherited by one base class only.

Syntax:

```
class subclass_name : access_mode base_class
{
//body of subclass
};
```

2. **Multiple Inheritances:**

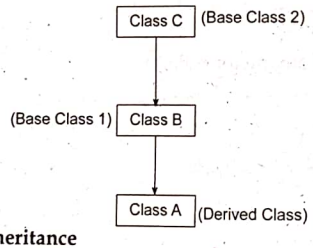
Multiple inheritance is a feature of C++ where a class can inherit from more than one classes. i.e one sub class is inherited from more than one base classes.

Syntax:

```
class subclass_name : access_mode base_class1, access_mode base_class2,
....
{
//body of subclass
};
```

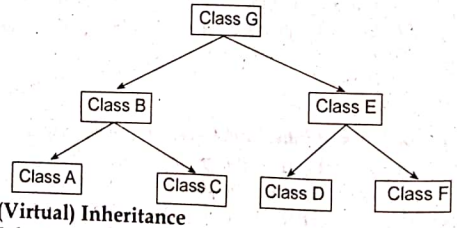
3. **Multilevel Inheritance**

In this type of inheritance, a derived class is created from another derived class.



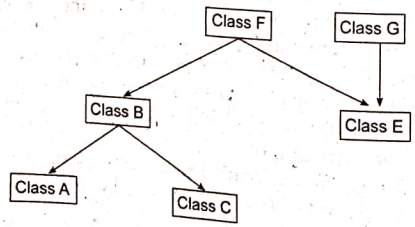
5. **Hierarchical Inheritance**

In this type of inheritance, more than one sub class is inherited from a single base class. I.e. more than one derived class is created from a single base class



Hybrid (Virtual) Inheritance

Hybrid Inheritance is implemented by combining more than one type of inheritance. For example: Combining Hierarchical inheritance and Multiple Inheritance. Below image shows the combination of hierarchical and multiple inheritance:



9. ***Create a real scenario where static data members are useful. Explain with suitable example.***

Ans: If a data member in a class is defined as static, then only one copy of that member is created for the entire class and is shared by all the objects of that class, no matter how many objects are created. Hence, these data members are normally used to maintain values common to the entire class and are also called class variables.

Example:

```
class rectangle
{
private:
int length;
int breadth;
static int count; //static data member
public:
void setdata(int l, int b)
{
length = l;
breadth = b;
count++;
}
void displaycount()
{
cout<<count<<endl;
}
};
int rectangle :: count;
void main()
{
clrscr();
rectangle r1, r2, r3;
r1.displaycount();
r2.displaycount();
r3.displaycount();
r1.setdata(15, 6);
r2.setdata(9, 8);
r3.setdata(12, 9);
r1.displaycount();
r2.displaycount();
r3.displaycount();
getche();
}
```

In this program, the static variable count is initialized to zero when the objects are created. The count is incremented whenever data is supplied to an object. Since the data is supplied three times, the variable count is incremented three times. Because there is only one copy of the count shared by all the three objects, all the three calls to the display count member function cause the value 3 to be displayed.

10. ***Create a function called swaps() that interchanges the values of the two arguments count to it (pass these arguments by reference). Make the function into a template. No it can used with all numerical data types (char, int, float and so on). Write a main () program to execute the function with several types.***

Answer: Templates help in defining generic classes and functions and hence allow generic programming. Generic programming is an approach where generic data types are used as parameters and the same piece of code work for various data types. Function templates are used to create family of functions with different argument types. The format of a function template is shown below:

```
template<class T>
return_type function_name (arguments of type T)
{
 ....
 ....
}
```

I have written a program below which will swap two numbers using function templates.

```
#include<iostream>
using namespace std;
template <class T>
void swap(T&a,T&b)     //Function Template
{
T temp=a;
a=b;
b=temp;
}
int main()
{
 int x1=4,y1=7;
 float x2=4.5,y2=7.5;
cout<<"Before Swap:";
cout<<"nx1="<<x1<<"ty1="<<y1;
cout<<"nx2="<<x2<<"ty2="<<y2;
swap(x1,y1);
swap(x2,y2);
cout<<"nnAfter Swap:";
cout<<"nx1="<<x1<<"ty1="<<y1;
cout<<"nx2="<<x2<<"ty2="<<y2;
return 0;
}
```

11. ***Explain how exceptions are used for handling C++ errors in a symmetric and OOP oriented uses with the design that includes multiple exceptions.***

Answer: Exception handling mechanism in C++ is basically built upon three keywords: try, throw, and catch. The keyword try is used to surround a block of statements, which may generate exceptions. This block of statements is known as try block. When an exception is detected, it is thrown using the throw statement situated either in the try block or in functions that are invoked from within the try block. This is called

throwing an exception and the point at which the throw is executed is called the throw point.

Exceptions provide a way to transfer control from one part of a program to another. C++ exception handling is built upon three keywords: try, catch, and throw.

- **throw** – A program throws an exception when a problem shows up. This is done using a throw keyword.
- **catch** – A program catches an exception with an exception handler at the place in a program where you want to handle the problem. The catch keyword indicates the catching of an exception.
- **try** – A try block identifies a block of code for which particular exceptions will be activated. It's followed by one or more catch blocks.

Code within a try/catch block is referred to as protected code, and the syntax for using try/catch as follows –

```
try
{
// protected code
} catch( ExceptionName e1 )
{
 // catch block
} catch( ExceptionName e2 )
{
// catch block
} catch( ExceptionName eN )
{
// catch block
}
```

Example:

```
#include<iostream.h>
#include<conio.h>
void test(int x)
{
try
{
if(x == 0) throw x;
if(x == 1) throw 1.0;
}
catch(int m)
{
cout<<"Caught an integer\n";
}
catch(double d)
{
cout<<"Caught a double";
}
}
void main()
{
```

```
clrscr();
test(0);
test(1);
test(2);
getche();
}
```

Output:

```
Caught an integer
Caught a double
```

12. *Use the character I/O differs from binary I/O? Explain with examples.*

Answer: The C++ language supports two types of files:

- Text files
- Binary files

The basic difference between text files and binary files is that in text files various character translations are performed such as "\r+\f" is converted into "\n", whereas in binary files no such translations are performed. By default, C++ opens the files in text mode.

Files Text	Binary Files
For writing data	
ofstream out ("myfile.txt"); or ofstream out; out.open("myfile.txt");	ofstream out ("myfile.txt",ios::binary); or ofstream out; out.open("myfile.txt", ios::binary);
For Appending (adding text at the end of the existing file)	

ofstream out("myfile.txt",ios::app);
or
ofstream out;
out.open("myfile.txt", ios::app);
	ofstream out
	("myfile.txt",ios::app | ios::binary);
	or
	ofstream out;
	out.open("myfile.txt", ios::app
	ios::binary);

For reading data	
ifstream in ("myfile.txt"); or ifstream in ; in.open("myfile.txt");	ifstream in ("myfile.txt", ios::binary); or ifstream in ; in.open("myfile.txt", ios::binary);

TU QUESTIONS-ANSWERS 2076

Section A

Long answer question:
Attempt ant two questions: **(2 x 10 = 20)**

1. **Write a program according to the specification given below:**
   - **Create a class Teacher with data members' tid & subject and ember functions for reading and displaying data members.**
   - **Create another class Staff with data members sid & position, and member function for reading and displaying data members.**
   - **Derive a class Coordinator from Teacher and Staff and the class must have its own data member department and member functions for reading and displaying data members.**
   - **Create two object of Coordinator class and read and display their details.**

Ans:

```
#include <iostream>
using namespace std;
class Teacher
{
 int tid;
 string subject;
 public:
  void TeacherRead()
  {
    cout<<"Enter teacher's id:"<<endl;
    cin>>tid;
    cout<<"Enter teacher's subject:"<<endl;
    cin>>subject;
  }

  void TeacherDisplay()
  {

    cout<<"Teacher's Id:"<<tid<<endl;
    cout<<"Teacher's subject:"<<subject<<endl;
  }
};

class Staff
{
 int sid;
 string position;
 public:
  void StaffRead()
  {
    cout<<"Enter staff's id:"<<endl;
```

```
    cin>>sid;
    cout<<"Enter staff's position:"<<endl;
    cin>>position;
  }

  void StaffDisplay()
  {
    cout<<"Staff's id:"<<sid<<endl;
    cout<<"Staff's position:"<<position<<endl;
  }
};

class Coordinator: public Teacher, public Staff
{
  string department;
  public:
   void CoordinatorRead()
   {
     cout<<"Enter department:"<<endl;
     cin>>department;
   }

   void CoordinatorDisplay()
   {
     cout<<"Department:"<<department<<endl;
   }
};

int main()
{
  Coordinator c1, c2;
  c1.TeacherRead();
  c1.CoordinatorRead();
  c1.TeacherDisplay();
  c1.CoordinatorDisplay();
  c2.StaffRead();
  c2.CoordinatorRead();
  c2. StaffDisplay();
  c2.CoordinatorDisplay();
  return 0;
}
```

2. Explain the concept of operator overloading? List the operators that cannot be overloaded. Write programs to add two object of distance class with data members feet and inch by using by using member function and friend function.

Ans: In C++, we can make operators to work for user defined classes. This means C++ has the ability to provide the operators with a special meaning for a user-defined data type, this ability is known as operator overloading. For example, we can overload an operator '+' in a class like String so that we can

concatenate two strings by just using +. Other example classes where arithmetic operators may be overloaded are Distance which has data member feet and inch, Complex Number etc. Operator that cannot be overloaded are as follows:

- Scope operator (::)
- Sizeof
- member selector(.)
- member pointer selector(*)
- ternary operator(?:)

Program

```
#include <iostream>
using namespace std;
class Distance
{
  private:
    int feet, inch;
  public:
    void readDistance(void)
    {
      cout << "Enter feet: ";
      cin >>feet;
      cout << "Enter inch: ";
      cin >>inch;
    }

    void displayDistance(void)
    {
      cout << "Feet:" << feet << "\t" << "Inches:" << inch << endl;
    }
    friend Distance operator+(Distance, Distance);
};

Distance operator+(Distance d1, Distance d2)
{
  Distance temp;
  temp.inch = d1.inch + d2.inch;
  temp.feet  = d1.feet + d2.feet + (temp.inch/12);
  temp.inch = temp.inch%12;
  return temp;
}

int main()
{
  Distance d1,d2,d3;
  cout << "Enter first  distance:" << endl;
  d1.readDistance();
  cout << endl;
```

```
    cout << "Enter second distance:" << endl;
    d2.readDistance();
    cout << endl;
    d3=d1+d2;
    cout << "The sum of two distance is:" << endl;
    d3.displayDistance();
    return 0;
}
```

3. Explain types of polymorphism briefly. Write down roles of polymorphism. How can we achieve dynamic polymorphism briefly? Write down roles of polymorphism. How can we achieve dynamic polymorphism? Explain with example.

Ans: Polymorphism is a feature of OOPs that allows the object to behave differently in different conditions. In C++ we have two types of polymorphism:

- Compile time Polymorphism – This is also known as static (or early) binding.
- Runtime Polymorphism – This is also known as dynamic (or late) binding.

Function overloading and Operator overloading are perfect example of Compile time polymorphism. Function overriding is an example of Runtime polymorphism.

A real-life example of polymorphism, a person at the same time can have different characteristics. Like a man at the same time is a father, a husband, an employee. So the same person possess different behavior in different situations. This is called polymorphism. Polymorphism is considered as one of the important features of Object Oriented Programming.

Function Overriding: When child class declares a method, which is already present in the parent class then this is called function overriding, here child class overrides the parent class.

In case of function overriding we have two definitions of the same function, one is parent class and one in child class. The call to the function is determined at runtime to decide which definition of the function is to be called, thats the reason it is called runtime polymorphism.

Example of Runtime Polymorphism

```
#include <iostream>
using namespace std;
class A
{
 public:
 void disp()
 {
   cout<<"Super Class Function"<<endl;
 }
};
```



```
class B: public A
{
 public:
 void disp()
 {
   cout<<"Sub Class Function";
 }
};
int main()
{
 //Parent class object
 A obj;
 obj.disp();
 //Child class object
 B obj2;
 obj2.disp();
 return 0;
}
```

Output:

```
Super Class Function
Sub Class Function
```

Group B

Short answer questions:

Attempt any eight questions: **(8 x 5 = 40)**

4. How object oriented Programming differs from object based programming language? Discuss benefits of OOP.

Ans: Most of the readers are unaware of the fact that there is a minor difference between Object-oriented Language and Object-based language. All programming languages that are Object-based Languages are not supposed to have the qualities of Object-oriented Language. Here are the significant difference between Object-oriented Programming Language and Object-based Programming Language:

Objected Programming Language	Object-based Programming Language
All the characteristics and features of object-oriented programming are supported.	All characteristics and features of object-oriented programming, such as inheritance and polymorphism are not supported.
These types of programming languages don't have a built-in object. **Example**: C++.	These types of programming languages have built-in objects. **Example**: JavaScript has a window object.
Java is an example of object-oriented programing language which supports creating and inheriting (which is reusing of code) one class from another.	VB is another example of object-based language as you can create and use classes and objects, but inheriting classes is not supported.

Benefits of OOP

- We can build the programs from standard working modules that communicate with one another, rather than having to start writing the code from scratch which leads to saving of development time and higher productivity,
- OOP language allows to break the program into the bit-sized problems that can be solved easily (one object at a time).
- The new technology promises greater programmer productivity, better quality of software and lesser maintenance cost.
- OOP systems can be easily upgraded from small to large systems.
- It is possible that multiple instances of objects co-exist without any interference,
- It is very easy to partition the work in a project based on objects.
- It is possible to map the objects in problem domain to those in the program.
- The principle of data hiding helps the programmer to build secure programs which cannot be invaded by the code in other parts of the program.
- By using inheritance, we can eliminate redundant code and extend the use of existing classes.
- Message passing techniques is used for communication between objects which makes the interface descriptions with external systems much simpler.
- The data-centered design approach enables us to capture more details of model in an implementable form.

5. What is the use of new and delete operators? Illustrate with example. What are advantages of new malloc?

Ans: **new** and **delete** operators are provided by C++ for runtime memory management. They are used for dynamic allocation and freeing of memory while a program is running. The **new** operator allocates memory and returns a pointer to the start of it. The delete operator frees memory previously allocated using new.

Example:

```
#include <iostream>
#include <new>
using namespace std;
int main()
{
  int *p;
  try
  {
    p = new int; //dynamic allocation of memory
  }
  catch (bad_alloc x)
  {
    cout << "Memory allocation failed";
```



```
    return 1;
  }
  *p = 100;
  cout <<"P has value"<<*p;
  delete p;  //free the dynamically allocated memory
  return 0;
}
```

new is an operator whereas malloc () is a function. Advantages of **new** over malloc () are listed below:

- new does not need the sizeof() operator whereas malloc() needs to know the size before memory allocation.
- Operator new can make a call to a constructor where as malloc() cannot.
- new can be overloaded malloc() can never be overloaded.
- new could initialise object while allocating memory to it where as malloc () cannot.

6. What is meant by return by reference? How can we return values by reference by using reference variable? Illustrate with examples.

Ans: A C++ function can return a reference in a similar way as it returns a pointer. When a function returns a reference, it returns an implicit pointer to its return value. This way, a function can be used on the left side of an assignment statement.

```
// C++ program to illustrate return by reference
#include <iostream>
using namespace std;
// Function to return as return by reference
int& returnValue(int& x)
{
    cout << "x = " << x << " The address of x is "<< &x << endl;
    return x;
}

// Driver Code
int main()
{
  int a = 20;
  int & b = returnValue(a);
  cout << "a = " << a<< " The address of a is "<< &a << endl;
  cout << "b = " << b<< " The address of b is "<< &b << endl;
  returnValue(a) = 13;
  cout << "a = " << a<< " The address of a is "<< &a << endl;
  return 0;
}
```

7. What is destructor? Write a program to show the destructor call such that it prints the message "memory is released".

Ans: Destructor is an instance member function which is invoked automatically whenever an object is going to be destroyed. Meaning, a destructor is the last function that is going to be called before an object is destroyed.

Destructor is a special class function which destroys the object as soon as the scope of object ends. The destructor is called automatically by the compiler when the object goes out of scope. Destructors are declared in a program to release memory space for future utilization.

Program:

```
#include <iostream>
using namespace std;
class MyClass
{
  private:
    int m, n;
  public:
    MyClass( )  // constructor
    {
      m = 10;
      n = 5;
    }
    ~MyClass( )  //destructor
    {
      cout<<"Memory is released!!";
    }
    void display()
    {
      cout<<"The sum is:"<<(m+n)<<endl;
    }
};
int main()
{
  MyClass obj;
  obj.display();
  return 0;
}
```

8. **What is this pointer? How can we use it for name conflict resolution? Illustrate with example.**

Ans: In C++, this pointer is used to represent the address of an object inside a member function. We can use keyword "this" to refer to this instance inside a class definition. One of the main usage of keyword this is to resolve ambiguity between the names of data member and function parameter.

For example,

```
#include <iostream>
#include <conio.h>
using namespace std;
class sample
{
  int a, b;
  public:
    void input(int a, int b)
    {
      this→a=a+b;
      this→b=a-b;
    }
    void output()
    {
      cout<<"a = "<<a<<endl<<"b = "<<b;
    }
};

int main()
{
  sample x;
  x.input(5,8);
  x.output();
  getch();
  return 0;
}
```


9. **How can you define catch statement that can catch any type of exception? Illustrate the use of multiple catch statement with example.**

Ans: Errors can be broadly categorized into two types. We will discuss them one by one.

- Compile Time Errors
- Run Time Errors

Compile Time Errors – Errors caught during compiled time is called Compile time errors. Compile time errors include library reference, syntax error or incorrect class import.

Run Time Errors - They are also known as exceptions. An exception caught during run time creates serious issues.

Errors hinder normal execution of program. Exception handling is the process of handling errors and exceptions in such a way that they do not hinder normal execution of the system. For example, User divides a number by zero, this will compile successfully but an exception or run time error will occur due to which our applications will be crashed. In order to avoid this we'll introduce exception handling technics in our code.

Example:

```
#include<iostream>
using namespace std;
int main()
{
  int a=10, b=0, c;
  try
  {
  //if a is divided by b(which has a value 0);
  if(b==0)
      throw(c);
  else
     c=a/b;
  }
  catch(char c)    //catch block to handle/catch exception
  {
```

```
cout<<"Caught exception : char type ";
 }
 catch(int i)    //catch block to handle/catch exception
 {
cout<<"Caught exception : int type ";
 }
 catch(short s)    //catch block to handle/catch exception
 {
cout<<"Caught exception : short type ";
 }
 cout<<"\n Hello";
}
```

10. Which functions can be used for reading and writing object? Describe briefly. Write a program that read values of two objects of student class (assume data members are sid, sname, and level) and display the data in monitor.

Ans: Files store data permanently in a storage device. With file handling, the output from a program can be stored in a file. Various operations can be performed on the data while in the file.

The fread() function in C++ reads the block of data from the stream. This function first, reads the count number of objects, each one with a size of size bytes from the given input stream.

The total amount of bytes reads if successful is (size*count). According to the no. of characters read, the indicator file position is incremented. If the objects read are not trivially copy-able, then the behavior is undefined and if the value of size or count is equal to zero, then this program will simply return 0.

Syntax:

size_t fread(void * buffer, size_t size, size_t count, FILE * stream)

The fwrite() function writes count number of objects, each of size size bytes to the given output stream.

Syntax:

```
int fwrite( const void *buffer, size_t size, size_t count, FILE *stream );
```

Program,

```
//C++ program to write and read object using read and write function.
#include <iostream>
#include <fstream>
using namespace std;
class student
{
  private:
    int sid;
    char name[30];
    int level;
  public:
    void getData(void)
    {
cout<<"Enter id of student:"; cin>>sid;
cout<<"Enter name:"; cin.getline(name,30);
      cout<<"Enter level:"; cin>>level;
    }
    void showData(void)
    {
cout<<"Id of student="<<sid<<"Name=" <<name<<", Level ="<<level<<endl;
    }
};

int main()
{
  student s;
  ofstream file;
  //open file in write mode
  file.open("stu.txt",ios::out);
  if(!file)
  {
   cout<<"Error in creating file.."<<endl;
   return 0;
  }
  cout<<"\nFile created successfully."<<endl;
  //write into file
  s.getData();   //read from user
  file.write((char*)&s, sizeof(s));   //write into file
  file.close();   //close the file
  cout<<"\n File saved and closed successfully."<<endl;
  //re open file in input mode and read data
  //open file1
  ifstream file1;
  //again open file in read mode
  file1.open("stu.txt", ios::in);
  if(!file1){
    cout<<"Error in opening file..";
    return 0;
  }
  file1.read((char*)&s, sizeof(s));
  s.showData();
  file1.close();
  return 0;
}
```

11. Write short notes on

- **Cascading of IO operators**

Ans: The cascading of the input and output operators refers to the consecutive occurrence of input or output operators in a single statement.

To understand the concept of cascading of the input/output operator, consider these examples. A program without cascading of the input/output operator.

```
#include
using namespace std;
int main ()
{
  int a, b;
```

```
    cin>>a;
    cin>>b;
    cout<<"The value of a is";
    cout<<a;
    cout<<"The value of b is";
    cout<<b;
    return 0;
}
```

In this example, all cin and cout statements use separate input and output operators respectively However, these statements can be combined by cascading the input and output operators accordingly as shown in this example.

A program with cascading of the input/output operator

```
#include
using namespace std;
int main ()
{
  int a, b;
  cin>>a>>b;
  cout<<"The value of b is : "<<b;
  cout<<"The value of a is "<<a;
  return 0;
}
```

- **Pure Virtual Function**

Ans: A pure virtual function is a virtual function in C++ for which we need not to write any function definition and only we have to declare it. It is declared by assigning 0 in the declaration. An abstract class is a class in C++ which have at least one pure virtual function.

Example:

```
#include<iostream>
using namespace std;
class B
{
  public:
    virtual void s() = 0; // Pure Virtual Function
};
class D:public B
{
  public:
    void s()
    {
      cout << "Virtual Function in Derived class\n";
    }
};

int main()
{
  B *b;
  D dobj;
  b = &dobj;
  b→s();
}
```


TU QUESTIONS-ANSWERS 2078

Section A

Long answer questions:

Attempt any two questions: (2x10=20)

1. **Write a program according to the specification given below:**
   - **Create a class Account with data members acc_no, balance, and min_balance(static)**
   - **Include methods for reading and displaying values of objects**
   - **Define static member function to display min_balance**
   - **Create array of objects to store data of 5 accounts and read and display values of each object**

Ans:

```
public class Account
{
  Private:
   int acno;
   float balance;
   static float min_balance;
   public:
   void read()
    {
      cout <<"Enter accno, balance and minimum balance"<<endl;
      cin>>accno>>balance>>min_balance;
    }
    void Display()
    {
  cout<<"Account number="<<acno<<endl<<"Balance="<< balance <<end;
    }
    static void Display_min_Balance()
    {
        cout<<"Minimum Balance="<<min_balance<<endl;
    }
  }
  void main()
  {
    Account Acc[5];
    int i;
    for(i=0;i<5;i++)
    {
        Acc[i].read();
        Acc[i].Display();
        Acc[i].Display_min_balance();
    }
    getch();
  }
```

2. What is meant by type conversion? Define two way of converting one user defined data type (object) to another user defined object? Write a program that converts object of another distance class with data members feet and inch. (Assume 1m = 3.3 feet and 1cm = 0.4 inch)

Ans: Type conversion is the process that converts the predefined data type of one variable into an appropriate data type. The main idea behind type conversion is to convert two different data type variables into a single data type to solve mathematical and logical expressions easily without any data loss.

Type conversion can be done in two ways in C++, one is **implicit type** conversion, and the second is **explicit type** conversion.

Implicit Type Conversion

The implicit type conversion is the type of conversion done automatically by the compiler without any human effort. It means an implicit conversion automatically converts one data type into another type based on some predefined rules of the C++ compiler. Hence, it is also known as the automatic type conversion.

The following is the correct order of data types from lower rank to higher rank:

Bool → char → short int → int → unsigned int → long int → unsigned long int → long long int → float → double → long double

Example:

```
int x = 20;
short int y = 5;
int z = x + y;
```

In the above example, there are two different data type variables, x, and y, where x is an int type, and the y is of short int data type. And the resultant variable z is also an integer type that stores x and y variables. But the C++ compiler automatically converts the lower rank data type (short int) value into higher type (int) before resulting the sum of two numbers. Thus, it avoids the data loss, overflow, or sign loss in implicit type conversion of C++.

Explicit type conversion

The conversion, which is done by the user or requires user interferences called the explicit or user define type conversion.

Example:

```
#include <iostream>
using namespace std;
int main ()
{
// declare a float variable
float num2;
// initialize an int variable
```



```
int num1 = 25;
// convert data type from int to float
num2 = (float) num1;
cout << " The value of int num1 is: " << num1 << endl;
cout << " The value of float num2 is: " << num2 << endl;
return 0;
}
```

Program part,

```
class distance
{
int feet;
int inch;
public:
distance(int f, int i)
{
feet = f;
inch = i;
}
void display()
{
    cout<<"Feet = "<<feet<<endl<<"Inches = "<<inches;
  }
};
class dist
{
int meter;
int centimeter;
public:
dist(int m, int c)
{
meter = m;
centimeter = c;
}
operator distance( )
{
distance d;
int f, i;
f= meter*3.3.;
i=centimeter*0.4;
f=f+i/12;
i=i%12;
return distance(f,i);
}
};
void main()
{
clrscr();
distance d1;
```

```
dist d2(4,50);
d1=d2;
d1.display();
getche();
}
```

3. How ambiguity arises in multipath inheritance? How can you remove this type of ambiguity? Explain with suitable example.

Ans: In multiple inheritance, there may be possibility that a class may inherit member functions with same name from two or more base classes and the derived class may not have functions with same name as those of its base classes. If the object of the derived class need to access one of the same named member function of the base classes then it result in ambiguity as it is not clear to the compiler which base's class member function should be invoked. The ambiguity simply means the state when the compiler confused.

The ambiguity can be resolved by using the scope resolution operator to specify the class in which the member function lies as given below:

```
Objectname .BaseClassName :: FunctionNAme(..);
#include <iostream>
#include<conio.h>
class A
{
public:
void show()
{
    cout << "Class A";
}
};
class B
{
    public:
void show() Multiple Inheritance
{
    cout << "Class B;
}
};
class C : public A, public B
{
};
void main()
{
C objC; //object of class C
objC.show(); //ambiguous--will not compile
objC.A::show(); //OK
objC.B::show(); //OK
getch();
}
```

The problem is resolved using the scope-resolution operator to specify the class in which the function lies.

Thus objC.A::show(); refers to the function show() from A class,
While objC.B::show(); refers to the function in the B class. This is disambiguation.

Group B

Short answer questions:

Attempt any eight questions: **(8x5=40)**

4. What is structured programming? Discuss characteristics and problems associated with structured programming.

Ans: Structured programming is a procedural programming subset that reduces the need for goto statements. In many ways, OOP is considered a type of structured programming that deploys structured programming techniques. Certain languages – like Pascal, Algorithmic Language (ALGOL) and Ada – are designed to enforce structured programming.

Structured programming is a program written with only the structured programming constructions:

(1) Sequence,
(2) Repetition and
(3) Selection

Sequence

Lines or blocks of code are written and executed in sequential order.

Example:

```
x = 5
y = 11
z = x + y
print(z)
```

Repetition

Repeat a block of code (Action) while a condition is true. There is no limit to the number of times that the block can be executed.

```
While condition
        action
End While
```

Example:

```
x = 2
While x < 100
  print(x)
  x = x * x
End
```

Selection

Execute a block of code (Action) if a condition is true. The block of code is executed at most once.

```
If condition Then
  action
End If
```

Example:

```
x = 100
If x % 2 == 0
  print("The number is even.")
End If
```

The main characteristics of Structured Programming are:

- The code should be in modular nature.
- There should be single entry and single exit for each module (i.e. no unconditional gates).
- At least one construct each for sequence, condition and iteration.

Disadvantages of Structured Programming Approach:

- Since it is Machine-Independent, So it takes time to convert into machine code.
- The converted machine code is not the same as for assembly language.
- The program depends upon changeable factors like data-types. Therefore it needs to be updated with the need on the go.
- Usually the development in this approach takes longer time as it is language-dependent. Whereas in the case of assembly language, the development takes lesser time as it is fixed for the machine.

5. What is the use of get and getline functions? Explain with suitable example.

Ans: getline() function is used to read the whole line of text that ends with a newline character.

The C++ getline() is an in-built function defined in the <string.h> header file that allows accepting and reading single and multiple line strings from the input stream. In C++, the cin object also allows input from the user, but not multi-word or multi-line input. That's where the getline() function comes in handy.

Here getline() function is invoked for reading character input into the variable line. When we type \n, the reading is terminated or size-1 characters are read. The newline is accepted but converted to null character.

```
# include <iostream.h>
int main()
{
 char city [50];
cout<< "City Name :";
cin.getline(city, 50);
cout<< "City Name : "<< city << end1;
return 0;
}
```

6. What is meant by pass by reference? How can we pass arguments by reference by using reference variable? Illustrate with example.

Ans: Pass by reference (also called pass by address) means to pass the reference of an argument in the calling function to the corresponding formal parameter of the called function so that a copy of the address of the actual parameter is made in memory, i.e. the caller and the callee use the same variable for the parameter. If the callee modifies the parameter variable, the effect is visible to the caller's variable.

```
void main()
{
    int i = 10, j = 20;
    swapThemByVal(i, j);
```



```
 cout << i << " " << j << endl;   // displays 10 20
 swapThemByRef(i, j);
 cout << i << " " << j << endl;   // displays 20 10
 getch();
}

void swapThemByVal(int num1, int num2)
{
  int temp = num1;
  num1 = num2;
  num2 = temp;
}

void swapThemByRef(int& num1, int& num2)
{
  int temp = num1;
  num1 = num2;
  num2 = temp;
}
```

7. What is constructor? Explain the concept of default and default copy with suitable example.

Ans: A constructor is a special type of member function that is called automatically when an object is created.

In C++, a constructor has the same name as that of the class and it does not have a return type.

Default Constructor

A constructor with no parameters is known as a default constructor. It has no parameters.

```
#include <iostream>
using namespace std;
 class construct
{
public:
  int a, b;

  // Default Constructor
  construct()
  {
     a = 10;
     b = 20;
  }
};

int main()
{
 //Default constructor called automatically when the object is created
  construct c;
  cout << "a: " << c.a << endl
     << "b: " << c.b;
  return 1;
}
```

Copy Constructor

A copy constructor is a member function which initializes an object using another object of the same class. A copy constructor has the following general function prototype:

```
ClassName (const ClassName &old_obj);
```

Example:

```
#include <iostream>
using namespace std;
class Copy_Demo
{
 public: int x;
  Copy_Demo (int a)  // this is parameterized constructor
    {
     x=a;
    }
    Copy_Demo (Copy_Demo &i)  // this is copy constructor
    {
     x = i.x;
    }
};

int main ()
{
 Copy_Demo a1(40); // Calling the parameterized constructor.
 Copy_Demo a2(a1); // Calling the copy constructor.
 cout<<a2.x;
 return 0;
}
```

8. **What is the concept of friend function? How it violates the data hiding principle? Justify with example.**

Ans: The central idea of encapsulation and data hiding concept is that any non-member function has no access permission to the private data of the class. The private members of the class are accessed only from member functions of that class.

C++ allows a mechanism, in which a non-member function has access permission to the private members of the class. This can be done by declaring a non-member function **friend** to the class whose private data is to be accessed.

```
// C++ program to demonstrate the working of friend function
#include <iostream>
using namespace std;
class Distance
{
  private:
    int meter;
    friend int addFive(Distance); // friend function
  public:
    Distance() : meter(0)
    {
    }
};

// friend function definition
int addFive(Distance d)
{
  //accessing private members from the friend function
  d.meter += 5;
  return d.meter;
}
int main()
{
  Distance D;
  cout << "Distance: " << addFive(D);
  return 0;
}
```


9. **What is exception? Why exception handling is better to use? Explain exception handling with try..... catch by using suitable example.**

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

An exception is an unwanted or unexpected event, which occurs during the execution of a program i.e. at run time that disrupts the normal flow of the program's instructions. Exceptions can be caught and handled by the program. When an exception occurs within a method, it creates an object. This object is called the exception object. It contains information about the exception such as the name and description of the exception and the state of the program when the exception occurred.

An exception can occur for many reasons. Some of them are:

- Invalid user input
- Device failure
- Loss of network connection
- Physical limitations (out of disk memory)
- Code errors
- Opening an unavailable file

The benefits of exception handling are as follows,

- Exception handling can control run tune errors that occur in the program.
- It can avoid abnormal termination of the program and also shows the behavior of program to users.
- It can provide a facility to handle exceptions, throws message regarding exception and completes the execution of program by catching the exception
- It can separate the error handling code and normal code by using try-catch block.

Example:

```
#include <iostream>
#include<conio.h>
using namespace std;
int main()
{
  int a=10, b=0, c;
  // try block activates exception handling
  try
  {
    if(b == 0)
    {
      // throw custom exception
      throw "Division by zero not possible";
      c = a/b;
    }
  }
  catch(char* ex) // catches exception
  {
    cout<<ex;
  }
  return 0;
}
```

10. When class templates are useful? How can you define a class that can implement stack with integer as well as sack of strings? Illustrate with example.

Ans: class templates are useful when a class defines something that is independent of the data type. Can be useful for classes like LinkedList, BinaryTree, Stack, Queue, Array, etc.

```
// C++ Program to Implement stack using Class Templates
#include.<iostream>
#include <string>
using namespace std;
#define SIZE 5
template <class T>
class Stack
{
  public:
  Stack();
  void push(T k);
  T pop();
  T topElement();
  bool isFull();
  bool isEmpty();
private:
  int top;
  T st[SIZE];
};
```



```
template <class T> Stack<T>::Stack()
{
  top = -1;
}
template <class T> void Stack<T>::push(T k)
{
  if (isFull())
  {
    cout << "Stack is full\n";
  }
  cout << "Inserted element " << k << endl;
  top = top + 1;
  st[top] = k;
}
template <class T> bool Stack<T>::isEmpty()
{
  if (top == -1)
    return 1;
  else
    return 0;
}
template <class T> bool Stack<T>::isFull()
{
  if (top == (SIZE - 1))
    return 1;
  else
    return 0;
}
template <class T> T Stack<T>::pop()
{
  T popped_element = st[top];

  top--;
  return popped_element;
}
template <class T> T Stack<T>::topElement()
{
  T top_element = st[top];
  return top_element;
}
int main()
{
  Stack<int> integer_stack;
  Stack<string> string_stack;
  integer_stack.push(2);
  integer_stack.push(54);
  integer_stack.push(255);
  // Adding elements to string stack
  string_stack.push("Welcome");
```

```
    string_stack.push("to");
    string_stack.push("New Summit College");

    cout << integer_stack.pop() << " is removed from stack"<< endl;
    cout << string_stack.pop() << " is removed from stack "<< endl;
    cout << "Top element is " << integer_stack.topElement()<< endl;
    cout << "Top element is " << string_stack.topElement()<< endl;
    return 0;
}
```

11. What is meant by stream? Write a program that reads content of file data.txt and displays the content in monitor.

Ans: In C++ stream refers to the stream of characters that are transferred between the program thread and i/o. Stream classes in C++ are used to input and output operations on files and io devices. These classes have specific features and to handle input and output of the program. The iostream.h library holds all the stream classes in the C++ programming language.

Example:

```
#include<iostream>
#include<fstream>
#include<stdio.h>
using namespace std;
int main()
{
  char fileName[30], ch;
  fstream fp;
  cout<<"Enter the Name of File: ";
  gets(fileName);
  fp.open(fileName, fstream::in);
  if(!fp)
  {
    cout<<"\nError Occurred!";
    return 0;
  }
  cout<<"\nContent of "<<fileName<<":-\n";
  while(fp>>noskipws>>ch)
    cout<<ch;
  fp.close();
  cout<<endl;
  return 0;
}
```

12. Write short notes on:

a. Manipulators

Ans: Manipulators are operators used in C++ for formatting output. The data is manipulated by the programmer's choice of display.
Major manipulators used in C++ are: **endl** manipulator, **setw** manipulator, **setfill** manipulator and **setprecision** manipulator.

endl Manipulator:
This manipulator has the same functionality as the 'n' newline character.

For example:

```
cout << "Exforsys" << endl;
cout << "Training";
```

setw Manipulator:
This manipulator sets the minimum field width on output.
Syntax:
setw(x)
Here setw causes the number or string that follows it to be printed within a field of x characters wide and x is the argument set in setw manipulator.
Example:

```
#include <iostream>
#include <iomanip>

void main( )
{
int x1=123,x2= 234, x3=789;
cout << setw(8) << "Exforsys" << setw(20) << "Values" << endl
<< setw(8) << "test123" << setw(20)<< x1 << endl
<< setw(8) << "exam234" << setw(20)<< x2 << endl
<< setw(8) << "result789" << setw(20)<< x3 << endl;
}
```

setfill Manipulator:
This is used after setw manipulator. If a value does not entirely fill a field, then the character specified in the setfill argument of the manipulator is used for filling the fields.
Example:

```
#include <iostream>
#include <iomanip>
void main()
{
cout << setw(15) << setfill('*') << 99 << 97 << endl;
}
```

setprecision Manipulator:
The setprecision Manipulator is used with floating point numbers. It is used to set the number of digits printed to the right of the decimal point. This may be used in two forms:

- fixed
- scientific

These two forms are used when the keywords fixed or scientific are appropriately used before the setprecision manipulator. The keyword fixed before the setprecision manipulator prints the floating point number in fixed notation. The keyword scientific, before the setprecision manipulator, prints the floating point number in scientific notation.
Example:

```
#include <iostream>
#include <iomanip>
void main( )
{
```

```
float x = 0.1;
cout << fixed << setprecision(3) << x << endl;
cout << scientific << x << endl;
}
```

b. Protected Access specifier

Ans: Protected access modifier is similar to private access modifier in the sense that it can't be accessed outside of its class unless with the help of friend class, the difference is that the class members declared as Protected can be accessed by any subclass (derived class) of that class as well.

Class members declared as protected can be used only by the following:

- Member functions of the class that originally declared these members.
- Friends of the class that originally declared these members.
- Classes derived with public or protected access from the class that originally declared these members.
- Direct privately derived classes that also have private access to protected members.

Example

```
#include <iostream>
using namespace std;
// declare parent class
class Sample
{
   protected:
   int age;
};
class SampleChild : public Sample
{
  public:
   void displayAge(int a)
   {
      age = a;
      cout << "Age = " << age << endl;
   }
};
int main()
{
   int ageInput;
   SampleChild child;

   cout << "Enter your age: ";
   cin >> ageInput;
   child.displayAge(ageInput);
   return 0;
}
```