Important Instructions to examiners:

1) The answers should be examined by key words and not as word-to-word as given in the model answer scheme.
2) The model answer and the answer written by candidate may vary but the examiner may try to assess the understanding level of the candidate.
3) The language errors such as grammatical, spelling errors should not be given more Importance (Not applicable for subject English and Communication Skills).
4) While assessing figures, examiner may give credit for principal components indicated in the figure. The figures drawn by candidate and model answer may vary. The examiner may give credit for any equivalent figure drawn.
5) Credits may be given step wise for numerical problems. In some cases, the assumed constant values may vary and there may be some difference in the candidate's answers and model answer.
6) In case of some questions credit may be given by judgement on part of examiner of relevant answer based on candidate's understanding.
7) For programming language papers, credit may be given to any other program based on equivalent concept.

Q1 Marks 20

a) Object oriented programming languages are (½ Mark each)

Smalltalk, Simula, C++, Ada, Java, Effiel, object pascal.

b) Structure is collection of different types of data items. (Def. 1 Mark , Syntax – 1Mark)

Syntax:

```
struct  structurename
       {  dataype  variable1;
          datatype variable2;
                .
                .
           datatype variable;
     }structvariable list;
```

c) Class is user define data type to bind the data and its associated functions together. Using private declaration class accomplishes data hiding. (Def. 1 Mark , Exp. – 1Mark)

d) A constructor is special member function whose task is to initialize the object of its class.

```
class integer
{
  int m,n;
 public:
          integer(void);                //constructor declaration1.
```

```
        integer(int x,int y);           //constructor declaration2.
    };
```
(Def. 1 Mark , Syntax – 1Mark)

e) Pointer is variable which holds the memory address of other variable.

Datatype *pointer_name; (Def. 1 Mark , Syntax – 1Mark)

f) Differences between Static and Dynamic Binding (1 Mark for Each Diff)

Static binding	**Dynamic binding**
a) In static binding linking of procedure call to the code is at compilation.	a) In dynamic binding linking of procedure call to the code is at execution.
b) Static binding is achieved using function overloading, operator overloading.	b) dynamic binding is achieved through virtual function.

g) Only one destructor because we can't overload the destructor. (2 Marks)

h) &:-The &(The address of operator) is a unary operator that returns the memory address of its operand. For ex. If var is an integer variable, then &var is its address.
:-The "" is the indirection operator and it is the complement of &. It is the unary operator that returns the value of the variable located at the address specified by its operand. (1 Mark each)

i) A private member of a base class can't be inherited and therefore it is not available for the derived class directly .what do we do if the private data needs to be inherited by derived?

C++ provides a third visibility modifier, protected ,which serve limited purpose in inheritance .A member declared as protected is accessible by the member functions within its class & any class immediately derived from it.
(2 Marks)

j) The mechanism of giving special meaning to the operator is known as operator overloading. C++ tries to make user defined data types behave in much the same way as built-in types.

The general form of operator function is:

Returntype classname::operator op_sign(arg-list)

```
{

  //Function  body

}
```

(Def. 1 Mark , Syntax – 1Mark)

k) An abstract base class is one that is not used to create objects. An abstract base class is designed only to act as base class. (2 Marks)

l) IOS format functions are

width(int no) – to specify the required field size for displaying an output value.

precision (int no) – to specify the number of digits to be displayed after the decimal point

Fill (char)-to specify a character that is used to fill the unused portion of a field.

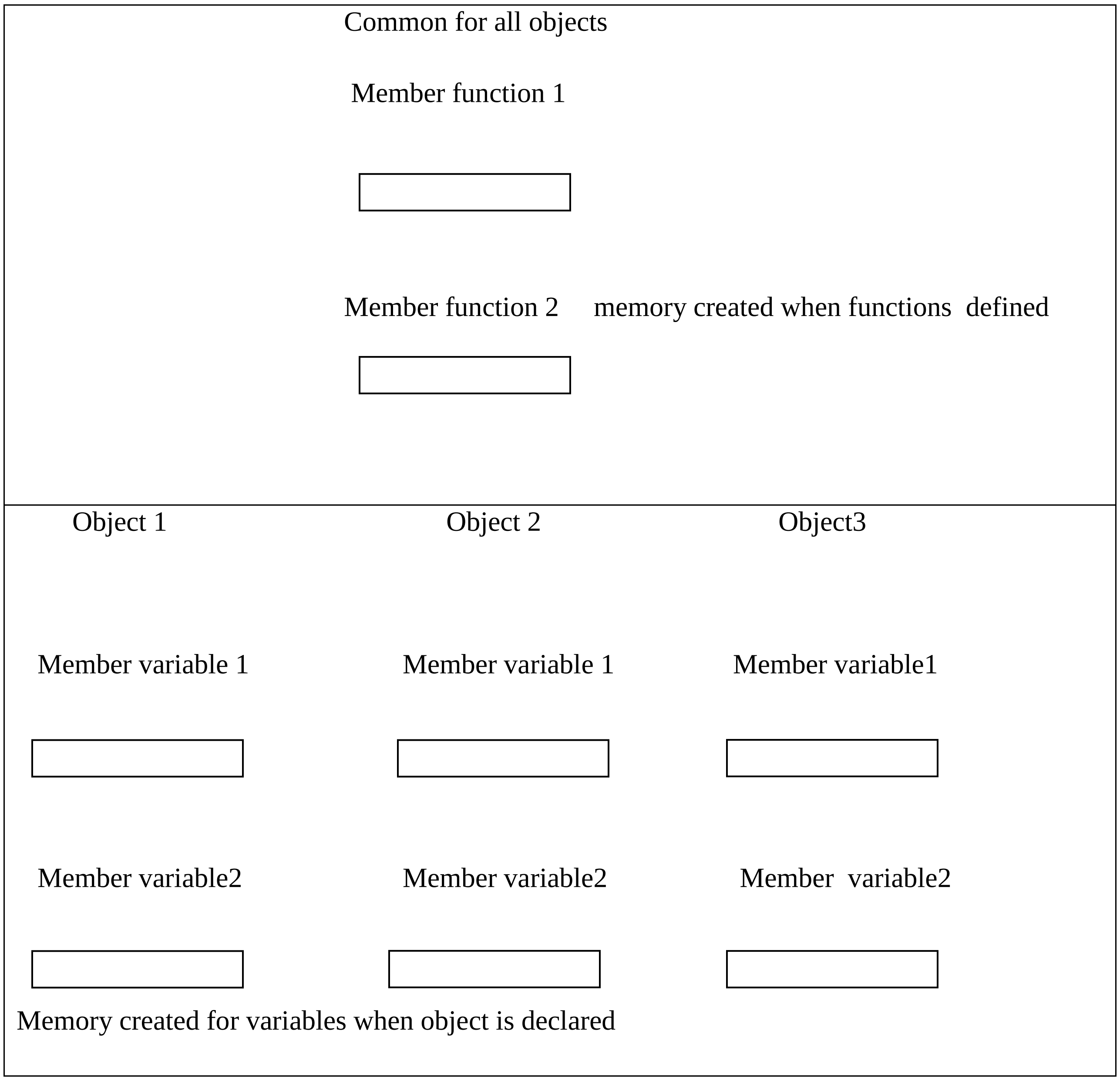
Setf()-to specify the format flags that can control the form of output display.

Unsetf()-to clear the flags specified. (½ Mark Each)

Q2 a) Memory Allocation for Objects: (Dia. 1 Mark, Exp 3 Marks)

The memory space for object is allocated when they are declared & not when the class is specified. This statement is partly true.

Actually, the member functions are created &placed in memory space only once when they are defined as a part of a class definition since all the objects belonging to that class use the same member functions, no separate space is allocated for member functions when the objects are created .only space for member variable is allocated separately for each object. Separate memory locations for the objects are essential because the member variables will hold different data values for different objects this is shown in fig



Memory Allocation of objects

Q.2 b) Multiple Inheritance: (Dia. 1 Mark, Exp 1 Mark , Example 2 Marks)

A class can inherit the attributes of two or more classes as shown in fig .this is known as multiple inheritance.

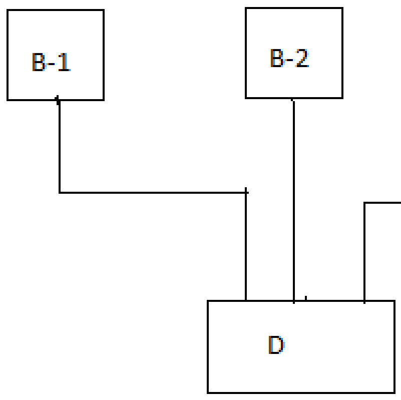


Fig. Multiple Inheritance

Example:-

```
#include<iostream.h>

#include<conio.h>

class M

{

protected:

 int a;

 public :

 void get_a(int);

 };

 class N

{

protected:

 int b;

 public :
```

```
 void get_b(int);
 };
 class P :public M,public N
 {
 public :
 void display();
 };
 void M::get_a(int x)
 {
 a=x;
 }
 void N::get_b(int y)
 {
b=y;
 }
 void P::display()
 {cout<<"a="<<a<<"\n";
   cout<<"b="<<b<<"\n";
   cout<<"a*b="<<a*b<<"\n";
   }
   void main()
   {
```

```
  P z;

  z.get_a(10);

  z.get_b(20);

  z.display();

  getch();

  }
```

Class P is derived from base classes M & N. P inherits the properties of M & N (multiple inheritance).

Q.2 c) Overloaded Constructor: (Example 3 Marks, Exp 1 Marks)

When more than one constructor are present in a class then it is a example of constructor overloading.

By using this concept in program user can create different object with various methods & initialization.

Example:-

```
#include<iostream.h>

#include<conio.h>

class integer

{

int m,n;

public:

integer(){m=0;n=0;}           //constructor1

integer(int a, int b)         //constructor2

{m=a;n=b;}

integer(integer &i)    //constructor3

{m=i.m;
```

```
 n=i.n;
}
void display()
{
cout<<"m="<<m<<"\t";
 cout<<"n="<<n<<"\n";
}
};
void main()
{
 integer I1;
 integer I2(10,20);
 integer I3(I2);
 cout<<"object I1";
 I1.display();
 cout<<"\nobject I2";
 I2.display();
cout<<"\nobject I3";
 I3.display();
 getch();
 }
```

In class integer three constructors are present because of this it is example of constructor overloading. here objects are initialized to I1(m=0,n=0),I2(m=10,n=20),I3(copies values of object I2 i.e. (m=10,n=20) .

Q.2 d) Program: To declare a class ‘Student’ .Accept and display this data for one object.

(Class 2 Marks, Main 2 Marks)

```
#include<iostream.h>

#include<conio.h>

class student

 {

protected:

 int roll_no;

 char name[20];

 public :

 void get()

 {

 cout<<"\nenter rollno & name";

 cin>>roll_no>>name;

 }

 void put()

 {cout<<"\nroll_no\t" <<roll_no;

  cout<<"\nname\t"<<name;

  }

  };

  void main()

  {

  student s1;
  s1.get();

  s1.put();

  getch();

  }
```

Q.2 e) Virtual base class:- (Dia. 1 Mark, Exp 3 Marks)

Consider a situation where all three kinds of inheritance, namely, multilevel, multiple, hierarchical inheritance, are involved. This illustrated in fig a. the child has two direct base classes 'parent1' & parent2 which themselves have a common base class 'grandparent'. The child inherits the traits of 'grandparent' via two separate path .it can also inherit directly as shown by broken line. The 'grandparent' sometimes referred to as indirect base class.

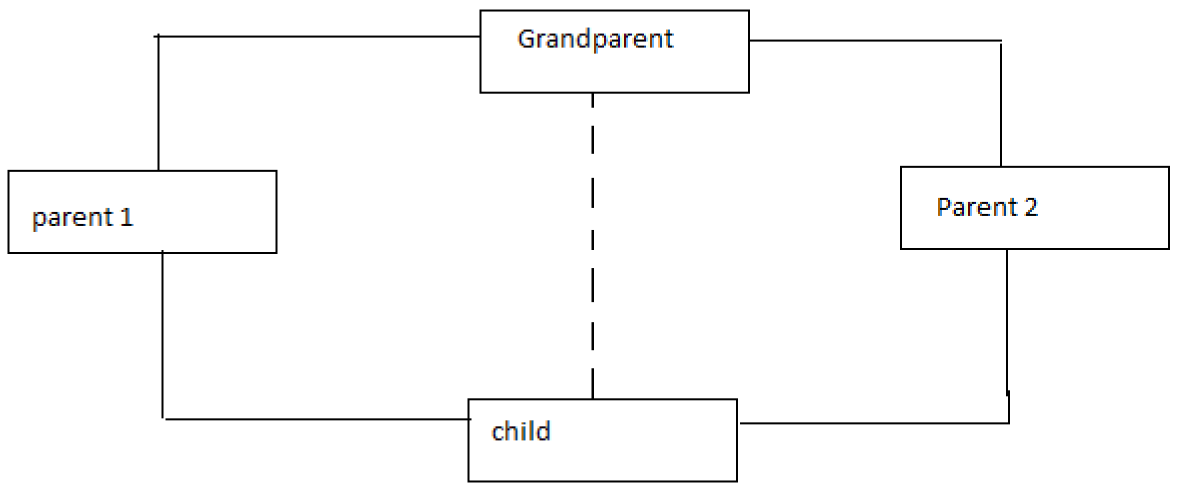


fig a

inheritance by the 'child' as shown in fig a might pose some problems. All the public & protected members of 'grandparent' are inherited into 'child' twice, first via 'parent1' & again via 'parent 2'.This means, 'child' would have duplicate sets of the members inherited from 'grandparent'. This introduces ambiguity & should be avoided.

The duplication of inherited members due to these multiple paths can be avoided by making the common base class as virtual base class while declaring the direct or intermediate base classes as shown below.

```
Class A                    //grandparent

{
```

```
//body of class

};

Class B1:virtual public A                    //parent1

{

//body of class

};

Class B2:public virtual A                    //parent2

{

//body of class

};

Class C:public B1,public B2                    //child

{

//only one copy of A will be inherited

//body of class

};
```

Q.2 f) Function Overriding: (Example 2 Marks, Exp 2 Marks)

Polymorphism refers to the property by which objects belonging to different classes are able to respond to the same message. when prototype of the function in base & derived class is same then which function to be executed for given function call is decided at the time of execution is called as function overriding. This is achieved by using the concept of virtual function.

When we use same function name in both the base & derived classes, the function in base class is declared as virtual using the keyword virtual preceding its normal declaration. When a function is made virtual, C++ determines which function to use at run time based on the type of object pointed to by the base pointer, rather than type of pointer.

Example:-

```
#include<iostream.h>

#include<conio.h>

class Base

{

public :

 void display()

 {cout<<"\n display base";

 }

 virtual void show(){cout<<"\nshow base";}

 };

 class Derived

{

public :

 void display()

 {cout<<"\n display derived";

 }

 virtual void show(){cout<<"\nshow derived";}

 };

 void main()

 {

 Base b,*bptr;

 Derived d;
 cout<<"\n\n bptr points to base\n";

 bptr=&b;

 bptr->display();

 bptr->show();

 cout<<"\n\n bptr points to derived\n";

 bptr=&d;

 bptr->display();

 bptr->show();

 getch();

 }
```

Q.3 a) Rules for Overloading operators : (Any four rules, 1 mark for each rule)

1. Only existing operators can be overloaded. operators can not be created.
2.The overloaded operator must have at least one operand that is of user defined type.
3. We cannot change the basic meaning of an operator i.e. we cannot redefine the plus(+) operator to subtract one value from the other.
4. Overloaded operators follow the syntax rules of the original operators. They cannot be overridden.
5. There are some operators that cannot be overloaded. for e.g. sizeof, . , .* , : : , ?: .
6.We cannot use friend functions to overload certain operators (=,( ),[ ],->).However member functions can be used to overload them.
7.Unary operators overloaded by means of a member function, take no explicit arguments and return no explicit values, but those overloaded by means of a friend function, take one reference argument.
8.Binary operators overloaded through a member function take one explicit argument and those which are overloaded through a friend function take two explicit arguments.

Q.3b) Differences between Call by value and Call by reference (1 mark for each point)

Sr.No	Call by value	Call by reference
1	Whenever a portion of program invokes a function with formal arguments, control will be transferred from main to calling function and value of actual arguments is copied to function	When a function is called by portion of program the address of actual arguments are copied on to formal arguments, though they may be refereed by different variable name.

2	When control is transferred back from function to calling function of program, the altered values are not transferred back.	Any change i.e. made to data item will be recognized in both function and calling portion of program.
3	In this formal arguments are pass to function	In this arguments are passed through references of address.
4	Suitable example program	Suitable example program

Q.3 (c)In C++ Access specifiers are also called as visibility mode. They are as follows: (1 mark for Listing)

1. Public
2. Private
3. Protected

1. Public:- When access specifier is public i.e. base class is publicly inherited by derived class all public member of base class become public members of derived class and all protected members become protected in derived class. (1 mark)

2. Private:- All public and protected members of base class become private members of derived class. (1 mark)

3. Protected:- In this all public and protected members of base class become protected of derived class. (1 mark).

Q. 3d) Program : To find number of vowels in each word of given text using pointer.

(Logic: 2Marks Syntax**:** 2Marks)

```
#include <iostream.h>

#include<conio.h>

#include<string.h>

Void main()

{

  int vow_cnt=0;

Char name[20];

Cout<<"Enter name" <<endl;

Cin>>name;

For(int i = 0; i<strlen(name); i++)

{
 if(name[i]==’a’ || name[i]== ‘e’ || name[i]==’i’ || name[i] == ‘o’ || name[i]== ‘u’)
  {
        Vow_cnt++;
}
}
   cout<< “Total vowels in the string are =”<<vow_cnt<<endl;
 getch();
}
```

```
o/p : Enter name=> student
      total vowels in string are => 2
```

Q.3 e) Run-time polymorphism: (2Marks explanation, 2 Marks example)

In some situation where the function name & prototype is same in both base & derived classes, since prototype of function is same in both places ,the function is not overloaded & therefore

static binding does not apply. In such cases, the appropriate member function is selected while program is running. This is known as run-time polymorphism.

Virtual functions are used to achieve run-time polymorphism. At run-time when it is known what class objects are consideration , the appropriate version of function is called. Since the function is linked with particular class much later after the compilation, this process is called as late binding. It is also called as dynamic binding.

The main advantage of late binding is flexibility at run-time,

Note: *Any suitable Example*

Q.3 (f) ios:: in, ios::out : (2 Marks for each)

When one is using old library to open a stream for input & output using fstreame, one must explicitly specify both ios::in & ios::out mode values.

No default value for mode is specified.

Ios::in is used to open a file for reading only

Ios::out is used to open a file for writing only

Eg. If we want to perform both input & output in the file in binary mode

We can use in following open command.

Fstream File;

File.open(filename,ios::in/ios::out/ios::binary);

Q.4. a) Program: Compile time polymorphism using function overloading: (Logic: 2Marks Syntax**:** 2Marks)

```
#include<iostream.h>

#include<conio.h>

Class Example

{  private:

  char ch;

  int num;

 public:

void show(char a)

{

     ch = a;

cout<< " The character is " <<num << endl;

}

Void show(int  b)
{
   num = b;
cout<<"the integer is" << num << endl;
}};
 Void main()
{
    clrscr();
Example ob;  // object of class example created
   ob.show(12);
   ob.show('x');
getch();
}
```

O/p :

```
The integer is : 12
The character is : x
```

In above program class example has two functions with same name show() but different parameters. Hence we can say that show() function has been overloaded. So it is called as function overloading to implement compile time polymorphism.

Q. 4. b) Program : To swap two integers values using call by references. (Logic: 2Marks Syntax**:** 2Marks)

```
#include<iostream.h>
#include<conio.h>
  Void swap(int &a, int &b)
{
```

```
Int temp;

  Temp = a;

        a= b;

      b = temp;

}

  main()

{     clrscr();

      Int I = 5, j = 10;

Cout<< "Before swapping I=" <<i<<"J=" << j<< endl;

     swap( i,j)

  cout << "After swapping I=" <<i <<"J="<<j<<endl;

}
```

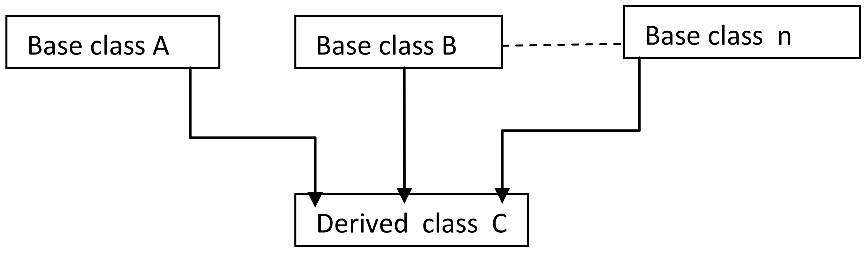
O/p : Before swapping I=5, J=10

After swapping I=10, J=5

Q.4 c) Multiple inheritance with example. (Definition 1 mark , 1 mark Dia.& 2 marks for example)

A derived class more than one base class is called as multiple inheritance. A class can inherit the attributes of two or more classes as shown :

Diag.:



Multiple inheritance

Example:

```
class baseA  {

---------

-----------

};

class baseB {

____

_____



};
```

In this example ,class deivedC is derived from both classes baseA & baseB.

Multiple inheritance can combine the behavior of multiple base classes in a single derived class.

Q.4 d) Describe array of objects in c++ with ex. (definition 2 marks & example 2 marks)

Array is a collection of similar data elements. An array simplifies the code of a program .It allows us to perform operations using loops.

To create more than one object of particular class the concept of array is combined with the definition of the case ,Such concept is called as "Array of object"

example:

```
class student

{

private:

int roll_no;

float percentage;
public:

void getdata();

void putdata();

}S[5];
```

S[5] is an array of objects of class student which creates 5 objects.

Q4 e) Define polymorphism .State its two types. (2 marks for definition & 1 mark for each type)

The ability of object to take more than form is known as polymorphism.

It is process of defining a number of objects of different classes into group & call the methods to carry out the operation of the objects using different function call.

There are two types:

1.Compile -time polymorphism or early-binding or static binding

2.Run-time polymorphism or late binding or dynamic binding.

Q4 f) Program: To overload operator '-'to negate value of variables. (Logic: 2Marks Syntax**:** 2Marks)

)

```
#include<iostream.h>

#include<conio.h>

class  sample

{

int x;

int y;

int z;

public:
```

```
void  getdata(int a,int  b,int c);

void display(void);

void operator-();

};

void  sample ::  getdata(int a,int b,int c)

{

x=a;

y=b;

z=c;

}

void sample :: display(void)

{

cout<<x<<endl;

cout<<y<<endl;

cout<<z<<endl;

}

void sample : : operator-()

{

x=-x;

y=-y;

z=-z;

}
```

```
void main()

{ clrscr();

sample S;

S.getdata(20,-30,10);

S.display();

-S;

S.display();

}
```

o/p:

```
20

-30

10

-20

30

-10
```

Q5 a)Program: To find whether the string is palindrome using pointer to string. (Logic: 4Marks Syntax**:** 4Marks)

```
#include<iostream.h>
#include<conio.h>
#include<string.h>
void main()
{
char str[20],str1[20],*ptr,*ptr1;
int t=0;
clrscr();
cout<<"\n Enter a string : ";
cin>>str;
strcpy(str1,str);
strrev(str);
ptr=str1;
ptr1=str;
while(*ptr!='\0' && *ptr1 !='\0')
{
 if(*ptr!=*ptr1)
  {
   t=1;
   break;
 }
 ptr++;
 ptr1++;
}
if(t==0)
cout << "\nString is palindrome.";
else
cout << "\nString is not palindrome.";
getch();
}
```

Q. 5 b) Program: To open existing file in read mode and count the number of spaces, new lines and characters in file.

(Logic: 4Marks Syntax**:** 4Marks)

```
#include<iostream.h>

#include<conio.h>

#include<fstream.h>

void main()

{

 ifstream file;
```

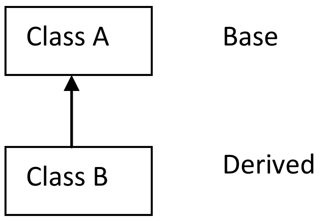
```
char ch;

int s=0,n=0,c=0;

clrscr();

file.open("PQR.txt");

while(file.eof())

{

 file.get(ch);

 if(ch=='\n')

 {

  n++;

 }

 else if(ch==' ')

 {

  s++;

 }

 else if((ch<=65 && ch>=90)||(ch<=65 && ch>=90))

 {

  c++;

 }

}

cout<<"\nNumber of spaces:"<<s;

cout<<"\nNumber of new lines:"<<n;
```

```
cout<<"\nNumber of characters:"<<c;

file.close();

getch();

}
```

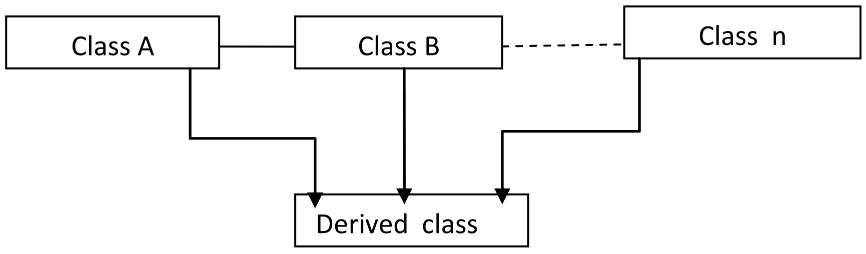
Q. 5 c) Different types of inheritance with Diagram: (Any 4 types, 2M arks for each type)

1. Single inheritance:

It includes single base class which can allow only one derived class to inherit its properties.

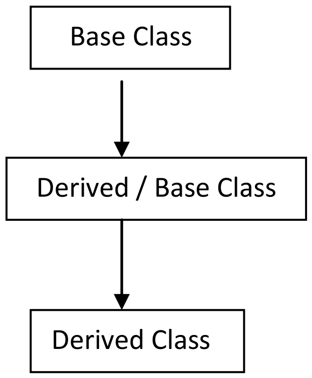


2. Multiple inheritance: In this inheritance, a single derived class can inherit properties of more than one base class.

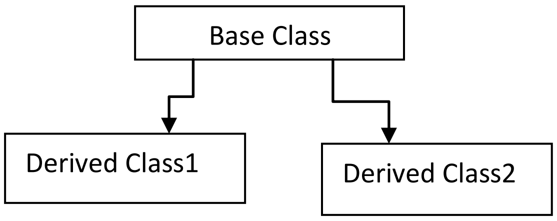


Multiple inheritance

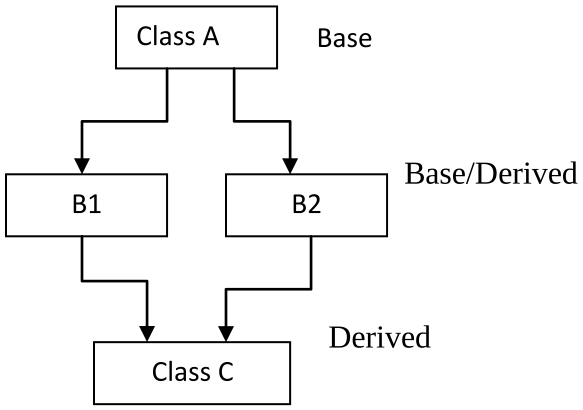
3. Multi-level inheritance: A single class can be derived from a single base class. We can derive a new class from as already derived class. It includes different levels for defining class. A child class can share properties of its base class (parent class) as well as grandparent class.



4. Hierarchical inheritance: In this inheritance, multiple classes can be derived from one single base class. All derived classes inherit properties of single base class.



5. Hybrid Inheritance: In this inheritance, it combines single inheritance, multiple inheritance, multi – level inheritance & hierarchical inheritance.



Q.6 a) Command Line Argument with example: (Explanation-2Marks Example-2Marks)

C++ provides a facility to pass arguments to the main function .These arguments are supplied or passed at the time of invoking the program. This facility is used for passing the names of data files as arguments to the main().

For e.g. C:\program.cc even odd

In above example, program.cc is the name of the file that contains the program to be executed. 'even' and 'odd' are the file names passed to the main() as command line arguments. The command line arguments are typed by the user and delimited by a space. The first argument is always the name of the file containing program and others are arguments required in program execution.

Syntax : main(int argc, char *argv[ ])

```
        {
```

```
        -----

        -----

        }
```

The first argument 'argc' known as argument counter represents the no of arguments in the command line. The second argument 'argv' , known as argument vector is an array of character type pointers that point to command line arguments.

argv[0] – program.cc

argv[1] – even

argv[2] – odd

Example :-

```
#include<iostream.h>

#include<conio.h>

#include<fstream.h>

#include<stdlib.h>

int main(int argc,char *argv[])

{

 int a[5]={1,2,3,4,5};

 ofstream fout1,fout2;

 clrscr();

 fout1.open(argv[1]);

 fout2.open(argv[2]);

 for(int i=0;i<5;i++)

 {
```

```
 if(a[i]%2==0)

 fout1<<a[i]<<" ";

 else

 fout2<<a[i]<<" ";

}

fout1.close();

fout2.close();

ifstream fin;

char ch;

for(i=1;i<argc;i++)

{

fin.open(argv[i]);

cout<<"Contents of "<<argv[i]<<"\n";

do

{

 fin.get(ch);

 cout<<ch;

}

while(fin);

cout<<"\n\n";

fin.close();

}
```

```
 getch();

 return 0;

}

/* Output

Contents of even

2 4

Contents of odd

1 3 5

*/
```

Note: ***Any other suitable example can be considered.***

Q. 6 b) Give Syntax and Use:

i. get()

get() function reads a single character from the associated stream. 1Mark

Syntax:

object.get(ch); 1Mark

e.g. cin.get('c');

ii. put() 1Mark

put() function writes a single character to the associated stream.

Syntax: 1Mark

object.put(ch);

e.g. cout.put('c');

Q.6 c) Difference between structure and class: (Any 4 points , 1 Mark for each)

Structure	Class
a) It contains logically related data items which can be of similar type or different type.	a) Data and functions that operate on the data are tied together in a data structure called class.
b) The data is not hidden from external use	b) It allows data and functions to be hidden from external use
c) Body of structure contains declaration of variables	c) Body of class contains declaration of variables and functions

d) Syntax- struct structure_name
{
Datatype variable_name;
 	
Datatype variable_n;
} Structure_variable;
	d) Syntax- class class_name
	{
	Access specifier:
	Declare data members;
	Declare member functions;
	};

e) Structure does not contain function declaration	e) Class contains function declaration
f) Structure does not provide data hinding	f) The basic purpose of class is to hide data from external use

g) Ex. Struct student
{
int roll_no;
char name[20];
} s;
	g) Ex. Class student
	{
	Private:
	int roll_no;
	char name[20];
	public:
	void getdata();
	void putdata();
	};

Q. 6 d) Need of virtual function: (Any 4 points , 1 Mark for each)

1. Polymorphism refers to the property by which objects belonging to different classes are able to respond to the same message, but in the different forms.
2. Therefore an essential requirement of polymorphism is the ability to refer to objects without any regard of their classes. This requires the use of a single pointer variable to refer to the objects of different classes.
3. Here, we use the pointer to base class to refer to all the derived objects.
4. The compiler simply ignores the contents of the pointer and chooses the member function that matches the type of the pointer. In this case polymorphism is achieved by using virtual functions.
5. When we use the same function name in both the base and derived classes, the function in the base class is declared as virtual using the keyword virtual preceding its normal declaration.
6. When a function is made virtual, C++ determines which function to use at runtime based on the type of object pointed to by the base pointer, rather than the type of the pointer.
7. Thus, by making the base pointer to point to different objects, different versions of the virtual functions can be executed.
8. Runtime polymorphism is achieved only when a virtual function is accessed through a pointer to the base class.

Q.6 e) Differences between Object oriented Programming, Procedure oriented Programming: (Any 4 points , 1 Mark for each)

Object oriented Programming	Procedure oriented Programming
1. Focus is given on the data	1. Focus is given on doing things.
2. Problem statement is divided into multiple objects	2. The problem statement is divided into multiple functions.
3. It follows bottom-up approach.	3. It follows top-down approach.
4. Data & functions operate on the data are tied together in a data structure called as class	4. Most of the function share global data.s
5. Data is hidden and can’t be accessed by external functions	5. Data moves openly around the system from one function to another function.
6. Objects can communicate with each other through functions.	6. Functions transform data from one form to another.

Q. 6 f) Creation of an object in definition of another class.: (Explanation-2Marks Example-2Marks)

Yes we can create objects of one class in another class but the members of first class should be publicly defined to get an access in second class.

Example:

```
#include<iostream.h>

#include<conio.h>

class student

{

 public:

 int roll_no;

 void get()

 {

  cout<<"Enter the roll no:";

  cin>>roll_no;

 }

};
```

class 1

```
class result

{

 student s;

 public:

 void putr()

 {

  s.get();
```

Object of class 1 in class 2

class 2

```
  cout<<"\nRoll no is "<<s.roll_no;

 }

};



void main()

{

 result r;

 r.putr();

 getch();

}
```

Note: ***Any other suitable example can be considered*.**