END SEM QUESTION BANK

Q1) Define constructor and list the types.

→ A constructor is a special function having the same name as class name.

- A constructor has no return type.
- A constructor may or may not have parameters
- A constructor is called when an object is initialised.
- A class can have more than one constructor.

- The types of constructors are:

(i) Default constructor: A is a constructor with no arguments. If the constructors has parameters then default values are given. Example on page 63

(ii) Parameterized constructor: It contains parameters in the constructor definition and declaration. More than one argument can be passed. Example on page 64

(iii) Copy constructor: It is a member function that initializes an object using another object of the same class. It helps to copy data from one object to another.

(iv) Dynamic constructor: When allocation of memory is done dynamically using dynamic memory allocator 'new' in a constructor, it is called a dynamic constructor.

- Example of copy constructor:

```
#include<iostream.h>
class Rectangle
{
    private:
        int length, breadth;
    public:
        Rectangle(int l, int b) //parameterised
        {                       constructor
            length = l;
            breadth = b;
        }
        Rectangle (Rectangle &obj) // copy constructor
        {
            length = obj.length;      } Copies the values of
            breadth = obj.breadth;    } l and b from
        }                               parameterised constructor
        int calculateArea()             (P.C.)
        {
            return length * breadth;
        }
};
int main()
{
    Rectangle obj(10,6);
    Rectangle obj2 = obj1; //Copies obj of P.C.
    cout<<"Area of rectangle 1 :"<<obj1.calculateArea();
    cout<<"\nArea of rectangle 2:"<<obj2.calculateArea();
    return 0;
```

```
}
```

- Here, the area of both rectangles will be the same as the copy constructor copies the values of the parameterized constructor.

Q2) Why is a constructor called dynamic?

→ When allocation of memory is done dynamically using dynamic memory allocator, 'new' in a constructor, it is called a dynamic constructor.

- Using this, we can dynamically initialize the objects.

eg)

```
#include<iostream.h>
class Dynamic ()
{
    int *p;
    public:
        Dynamic()
        {
            p= new int[3]{1,2,3};
            for(int i=0;i<3;i++)
            {
                cout<<p[i]<<" ";
            }
            cout<<"\n";
        }
};
int main()
```

dynamically allocated variable (has three values - (1,2,3))


```
{
    Dynamic* ptr = new Dynamic[5];
    return 0;
}
```

dynamically created array of objects (5 objects)

⊙ Output:

```
1 2 3
1 2 3
1 2 3
1 2 3
1 2 3
```

⇒ Same variable (3 values) is called for all 5 objects without calling as it is a constructor

- The dynamic constructor does not create the memory of the object but it creates the memory block that is accessed by the object.
- Dynamic constructor allows use of arguments.

Q3) Write a program to calculate area of different shapes using overloaded constructor.

→

```
#include<iostream.h>
class Area
{
    public:
        Area(int r)
        {
            cout<<"The area of the circle is:"<<3.14*r*r;
        }
        Area(float s)
        {
```

```
    cout<<"\nArea of the square is:"<<s*s;
  }
  Area(int l, int b)
  {
      cout<<"\nArea of the rectangle is:"<<l*b;
  }
};
int main()
{
    int r, l, b;
    float s;
    cout<<"Enter the radius of the circle:";
    cin>>r;
    cout<<"Enter the side of the square:";
    cin>>s;
    cout<<"Enter the length and breadth of the
      rectangle:";
    cin>>l>>b;
    Area a(r), ar(s), arr(l,b);
    return 0;
}
```

⊙ Output:

```
Enter the radius of the circle: 5
Enter the side of the square: 4
Enter the length and breadth of the rectangle: 3
4
The area of the circle is: 78.5
Area of the square is: 16
```



```
Area of the rectangle is: 12
```

Q4) Write a program in C++ using overloaded constructor to specify class name "BANK" with the members: Acc. no (int), Branch (string) and balance (float). Derive three objects Bk1, Bk2, Bk3 and assign proper values to each object.

→

```
#include<iostream.h>
class BANK
{
    public:
        BANK(int acc_no)
        {
            cout<<"The account number is:"<<acc_no;
        }
        BANK(string branch)
        {
            cout<<"\nThe branch is:"<<branch;
        }
        BANK(float balance)
        {
            cout<<"\nThe balance is: "<<balance;
        }
};
int main()
{
    int acc_no;
    string branch;
```

```
    float balance;
    cout<<"Enter the account number:";
    cin>>acc_no;
    cout<<"Enter the branch:";
    cin>>branch;
    cout<<"Enter the balance:";
    cin>>balance;
    BANK BK1(int acc_no), BK2(string branch),
            BK3(float balance);
    return 0;
}
```

⊙ Output:

```
Enter the account number: 23456
Enter the branch: gotri
Enter the amount: 100000
The account number is: 23456
The branch is: gotri
The balance is: 100000
```


Q5) Declare a class called Bird having data members name and weight.

→
```
#include<iostream.h>
class Bird
{
    private:
        string name;
        int weight; //can use float also.
    public:
        void getdata()
        {
            cout<<"Enter the name and weight of
                the bird:";
            cin>>name>>weight;
        }
        void display()
        {
            cout<<"Name - "<<name;
            cout<<"Weight - "<<weight;
        }
};
int main()
{
    Bird b1;
    b1.getdata();
    b1.display();
    return 0;
}
```

⊙ Output:

Enter the name and weight of the bird: Pigeon
0.38
Name - Pigeon
Weight - 0.38

Q6) Define the following:

1) Default constructor for reading data member from key node:
(done at last on page number)

2) Overloaded constructor with 2 arguments to be used for initialization of data members.

→

```
class Area
{
    private:
        int l,b;
    public:
        Area(int a, int d)
        {
            l=a;
            b=d;
        }
        Area()
        {
            cout<<"The area of rectangle is:"<<l*b;
```



```
        }
};
int main()
{
    int a,d;
    cout<<"Enter the length and breadth of the
        retangle:";
    cin>>a>>d;
    Area a(a,d), ar();
    return 0;
}
```

⊙ Output:

Enter the length and breadth of the rectangle: 3
4
The area of the rectangle is: 12

Q7) Explain 'this' pointer. ⟹ ??

→ The 'this' pointer in C++ points to the object that invokes the member function.

- This keyword is only accessible within the nonstatic member functions of a class / struct / union type.
- Some ways to use the 'this' pointer are:

① Same class variable and parameter names -
When the parameter name is same as the private variables name, we use 'this' keyword inside the method to distinguish between the two when assignin

values.

eg) class ClassName
{
    private:
        int sameName;
    public:
        void functionName(int sameName)
        {
            this → sameName = sameName;
            cout<<"Here, the variable used with "this", is the class variable that is declared as private.";
        }
};

② Returning the current object :-
We can also use 'this' to return the reference to the calling object.

eg) class ClassName
{
    void functionName()
    {
        //code;
        return *this;
    }
};

③ 'this' in Constructor :-
A 'this' pointer is always passed to the constructor. It holds the object's address that is being constructed during that call.

eg) #include<iostream.h>
class Age
{
    private:
        int a;
    public:
        Age(int a = 0)
        {
            cout<<"Address of object = "<<this;
            this → a = a;
        }
        Age& setAge(int x)
        {
            cout<<"Address of object = "<<this;
            a = x;
            return *this;
        }
        void getAge()
        {
            cout<<"\nAddress of the object : "<<this;
            cout<<"\nThe age is : "<<a;
        }
};
int main()
{
    Age obj;

```
    obj.setAge(20).getAge();
    return 0;
}
```

⊙ Output:

```
Address of object = 0x7ffca02d7654
Address of object = 0x7ffca02d7654
Address of object = 0x7ffca02d7654
The age is 20.
```

Q8) Explain destructor in brief.

→ A destructor is a special member function that destroys the class objects created by the constructor.

- A destructor has the same name as class name preceeded by the tilde '~' symbol.
- It is not possible to define more than one destructor.
- The destructor is only one way to destroy object created by constructor. Hence it can not be over-loaded.
- Destructor neither requires any argument nor returns any value.
- It is automatically called when an object goes out of scope.
- Destructor releases the memory space occupied by the objects created by the constructor.
- If the constructor uses 'new' to store memory then

the destructor should use 'delete' to release the memory.

Eg)
```
#include <iostream.h>
class Test
{
    public:
        Test()
        {
            cout<<"\nConstructor executed...";
        }
        ~Test()
        {
            cout<<"\nDestructor executed...";
        }
};
int main()
{
    Test t;
    return 0;
}
```

⊙ Output:

```
Constructor executed...
Destructor executed...
```

* Q9, Q10 and Q11 on page numbers 63, 64 and 66 respectively.

INHERITANCE

Q1. Explain static member function with example.

→ A static member function is a special member function used to access the static data members or other static member functions.

- A static member function shares the single copy of the member function to any number of the class' objects.
- If the static member function accesses any non-static data member or non-static member function, it gives an error.
- Syntax:

```
static return-type function-name()
{
}
```

eg)
```
#include <iostream.h>
class A
{
    private:
        static int count;
    public:
        static void plus()
        {
            cout << " count = " << count;
```



```
            count++;
            cout << " \n Count after incrementing: " << count;
        }
};
int A :: count = 0;
int main()
{
    A a1;
    a1.plus();
    return 0;
}
```

⊙ Output:

```
count = 0
Count after incrementing : 1
```

Q2. What is inheritance? list out types of inheritance and explain in brief.

→ The process of inheriting the properties of one class to another class is known as inheritance.

- The class whose properties are inherited is called the base class or parent class whereas the class inheriting the properties is called the derived class or child class.
- The private data members and member functions are not inherited.
- There are three modes of inheritance:

(i) Public Mode:

(ii) Protected Mode

(iii) Private Mode

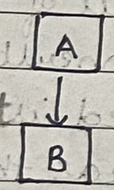
- The functions/use of these modes can be seen in the visibility table below:

Base class Access Specifier	Derived Class Visibility		
	Public	Private	Protected
Public	Public	Private	Protected
Private	Not allowed	Not allowed	Not allowed
Protected	Protected	Private	Protected.

- There are 5 types of inheritance:

① Single Inheritance: In this type of inheritance, there is only one parent class and one child class.

Add syntax in theory part



A: parent class

B: child class.

eg)
```
#include<iostream.h>
class A
{
    public:
        int a,b;
```



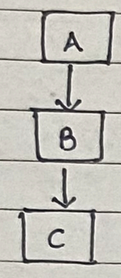
```
    void getdata()
    {
        cout<<"Enter the values of a and b:";
        cin>>a>>b;
    }
};
class B : public A
{
    public:
        void sum()
        {
            cout<<" a+b = "<<a+b;
        }
};
int main()
{
    B b1;
    b1.getdata();
    b1.sum();
    return 0;
}
```

⊙ Output:

```
Enter the values of a and b: 3
4
a+b = 7
```

② Multi Level Inheritance : In this type of inheritance, a derived class is created from another class.

- One class will act as both, a base class and a derived class.



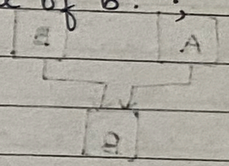
A : parent class for B
B : parent class for C and child class for A
C : child class of B.

- Here class C inherits the properties/characteristics of class A indirectly through class B.
- Syntax :

```
class A
{
};
class B : public A
{
};
class C : public B
{
};
```

- Class C will not inherit the properties of class A if the visibility mode of class B is private,
  i.e., class B: private A.

eg)
```
#include <iostream.h>
Class A
{
public:
    int a;
    void getdata1()
    {
        cout<<"Enter the value of a:";
        cin>>a;
    }
};
class B : public A
{
    public:
    int b;
    void getdata2()
    {
        cout<<"Enter the value of b:";
        cin>>b;
    }
};
class C : public B
{
    public:
        void ~~sum()~~ display()
        {
            cout<<"a+b="a+b;
        }
};
int main()
{
    C c1;
    c1.getdata1();
    c1.getdata2();
```

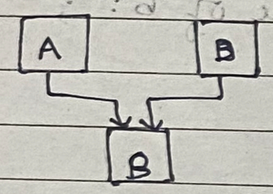


```
    c1.display();
    return 0;
}
```

⊙ Output:

```
Enter the value of a :3
Enter the value of b : 4
a+b = 7
```

③ Multiple Inheritance : In multiple inheritance one child class can inherit the characteristics of multiple parent classes.



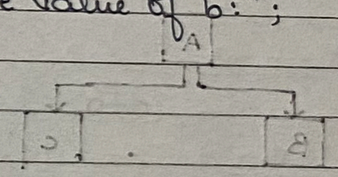
A: parent class
B: child class
D: parent class.

• Syntax :

```
class A
{
};
class D
{
};
class B : public A, public D
{
};
```


eg)

```
class A
{
    public:
        int a;
        void getdata1()
        {
            cout<<"Enter the value of a :";
            cin>>a;
        }
};
class B
{
    public :
        int b;
        void getdata2()
        {
            cout<<"Enter the value of b :";
            cin>>b;
        }
};
```



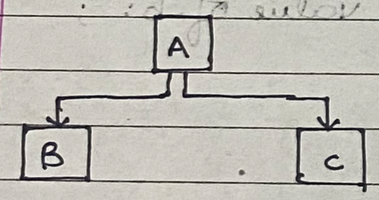

```
class C : public A , public B
{
    public :
        void display()
        {
            cout<<"a+b = "<<a+b;
        }
};
int main()
{
```

```
C cl;
cl.getdata1();
cl.getdata2();
cl.display();
return 0;
}
```

⊙ Output :

```
Enter the value of a : 3
Enter the value of b : 4
a + b = 7
```

④ Hierarchial Inheritance : In this type of inheritance, more than one child class is derived from a single parent class.



A : parent class
B : child class
C : child class

• Syntax :

```
class A
{
};
class B : public A
{
};
class C : public A
{
```



```
};
```

eg)

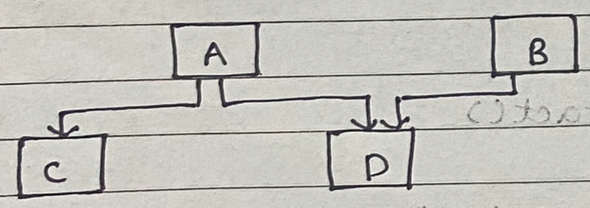
```
#include <iostream.h>
class A
{
    public:
        int a,b;
        void getdata()
        {
            cout<<"Enter the values of a and b";
            cin>>a>>b;
        }
};
class B : public A
{
    public:
        void add()
        {
            cout<<" a + b = "<<a+b;
        }
};
class C : public A
{
    public:
        void subtract()
        {
            cout<<" a-b = "<<a-b;
        }
};
int main()
```

```
{
    B b1;
    C c1;
    b1.getdata();
    b1.add();
    c1.getdata();
    c1.subtract();
    return 0;
}
```

⊙ Output:

```
Enter the values of a and b: 3
4
a+b = 7
Enter the values of a and b: 4
3
a-b = 1
```

(5) Hybrid Inheritance : Hybrid inheritance can be implemented by combining more than one type of inheritance.



Hierarchial Inheritance : A, C, D

Multiple Inheritance : A, B, D

- Syntax : (Different based on different types of inheritance combined. Give syntax of diagram taken).

eg)
```
class A
{
    public:
        int a,b;
        void getdata1()
        {
            cout<<" Enter the values of a and b:";
            cin>>a>>b;
        }
};
class B
{
    public:
        int b2;
        void getdata2()
        {
            cout<<"Enter the value of c :";
            cin>> b2;
        }
};
class c: public A
{
    ~~cout<<" a+b=" <<a+b;~~
    public:
        void sum1()
        {
```

```
    cout<<"a+b = "<<a+b;
    }
};
class D: public A, public B
{
    void sum2()
    {
        cout<<"a+b+c = "<<a+b+b2;
    }
};
int main()
{
    C c1;
    D d1;
    c1.getdata1();
    d1.getdata2();
    c1.sum1();
    d1.sum2();
    return 0;
}
```

⊙ Output :

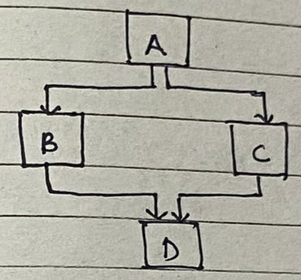
```
Enter the values of a and b: 3
4
Enter the value of c: 1
a+b = 7
a+b+c = 8
```

* Answers of Q3 - Q6 in answer 2 (different types of inheritances).

Q7) Explain virtual base class with example.

→ Virtual base class is a concept used in inheritance as a way of preventing multiple "instances" of a given class appearing in an inheritance hierarchy.

- Consider the following inheritance:



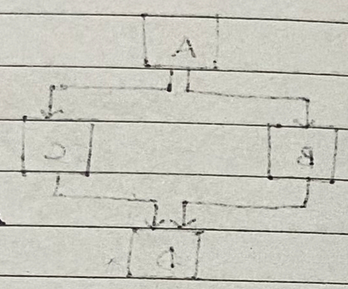
We have a parent class (base class) A whose properties are inherited to two classes B and C. Both these classes are inherited to a new class D.

- Here, the properties of A will be inherited to class D twice, once through B and once through C. When we call a member of class A in class D, the compiler gets confused as to which member to call.
- To prevent this, when class A is inherited in both class B and class C, it is declared as virtual base class, using the keyword 'virtual'.
- Syntax :

```
class B : virtual public A {
};
class C : virtual public A {
};
```

- When we do so, only one copy of data/ member will be copied to class B and C, and class A becomes virtual base class.
- It saves space and the virtual base class can act as an indirect base more than once without the duplication of its data/ function members.

eg)
```
#include<iostream.h>
class A
{
  public: int a;
          void value()
          {
            a=10;
          }
};
class B : virtual public A
{   public:
      void
      ~~public~~ display1()
      {
        cout<<" Class B : a = "<<a;
      }
};
class C : virtual public A
{
  public:
      void display2()
      {
        cout<<" Class C : a = "<<a;
      }
};
class D : public B, public C
{
  public:
      void display3()
      {
        cout<<" Class D : a = "<<a;
```




```
      }
};
int main()
{
  D d1;
  d1.value();
  d1.display1();
  d1.display2();
  d1.display3();
  return 0;
}
```

⊙ Output :

Class B : a = 10
class C : a = 10
Class D : a = 10

Q8) Explain visibility mode.

→ Visibility mode is used in inheritance to show or relate how base classes are viewed with respect to derived class.

- When one class is derived to another, visibility mode is used to inherit all public and protected members of base class.
- The accessibility of base class by the derived class is controlled by visibility mode (public, protected and private).

Draw Visibility table from question 2.

Q9) Write a program to find maximum number from two values using the following diagram.

A	B
Data member : x	Data member : Y
Function member : get-x	Function member : get-y

C (max.)

```
#include<iostream.h>
class A
{
    public :
        int x;
        void get-x()
        {
            cout << " Enter the value of x :";
            cin >> x;
        }
};
class B
{
    public :
        int y;
        void get-y()
        {
            cout << " Enter the value of y :";
            cin >> y;
        }
};
class C : public A, public B
{
```



```
    public :
        void max()
        {
            if (a>b)
            {
                cout << a << " is greater";
            }
            else
            {
                cout << b << " is greater";
            }
        }
};
int main()
{
    C c;
    c.get-x();
    c.get-y();
    c.max();
    return 0;
}
```

Output :

```
Enter the value of x : 3
Enter the value of y : 6
6 is greater
```

Q10) Explain various stream classes available in C++.

OR

Explain C++ stream class with its hierarchy structure.

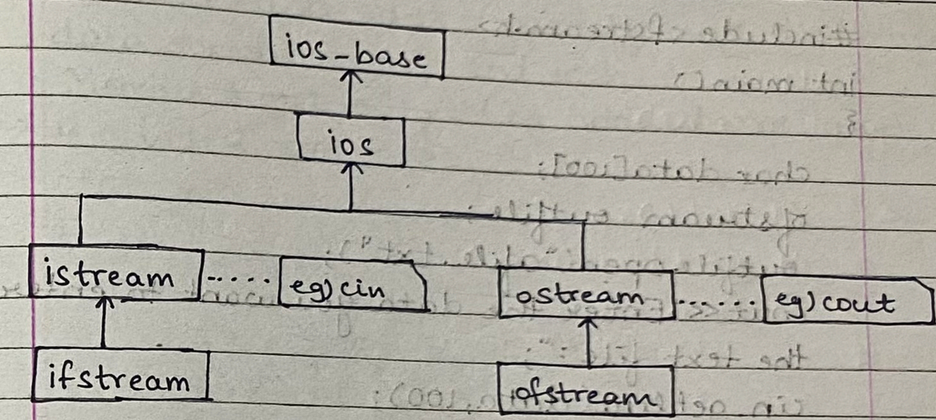
→ A C++ stream is a flow of data into or out of a program, such as the data written to cout or read from cin.

- 4 such streams are:

1) istream: is a general purpose input stream. eg) cin
2) ostream: is a general purpose output stream. eg cout, cerr.
3) instream: is an input file stream. It is a special kind of istream that reads in data from a data file.
4) ofstream: is an output file stream. It is a special kind of ostream that writes data out to a data file.

- In the concept of inheritance, some classes inherit the properties of previously written classes. The derived classes then add on additional properties, making them specialisation of their parent class.
- Similarly, ifstream is a specialization of istream. This means that ifstream is an istream and includes all the properties of istream class, plus additional properties of its own.
- The use of ifstream and ofstream requires the inclusion of the fstream header.

- The hierarchy of stream classes:



⊙ The ifstream class :-

An ifstream is an input file stream, i.e., a stream used for reading input from a file.

- Because an ifstream is an istream, anything you can do to an istream, you can also the same way to an ifstream.

⊙ The ofstream class :-

An ofstream is an output file stream, and works exactly like ifstreams, except for output instead of input.

- Once an ofstream is created, opened and checked for failures, it can be used just like cout.

- Following is a simple program to write to and read from data from a text file:

```
#include <iostream.h>
#include <fstream.h>
int main()
{
    char data[100];
    ofstream outfile;
    outfile.open("afile.txt");
    cout<<"Enter the data you want to enter in
    the text file:";
    cin.getline(data,100);
    outfile<<data;
    outfile.close();
    instream infile;
    infile.open("afile.txt");
    cout<<"\n The data in the file is:\n";
    infile>>data;
    cout<<data;
    infile.close();
    return 0;
}
```

⊙ Output:

Enter the data you want to enter in the file:
Yashasvi - 4136

The data in the file is:
Yashasvi-4136

Q11) Write a program which creates a class named distance and adds two different distance data members feet and inch.
Member function: get(), Adddistance(arg).

→
```
#include <iostream.h>
class gets
{
    public:
        float f1,f2,i1,i2,ft,it;
        void get()
        {
            cout<<"Enter feet of first distance:";
            cin>>f1;
            cout<<endl<<"Enter feet of second distance:";
            cin>>f2;
            cout<<endl<<"Enter inch of first distance:";
            cin>>i1;
            cout<<endl<<"Enter inch of second distance:";
            cin>>i2;
        }
        void Adddistance()
        {
            ft = f1+f2;
            it = i1+i2;
            while(it>=12)
            {
                it=it-12;
                ft++;
            }
            cout<<endl<<"The distance is"<<"ft<<" feet
```

```
and "<<it<<" inches";
}
} A1;
int main()
{
    A1.get();
    A1.Adddistance();
    return 0;
}
```

⊙ OUTPUT :

```
Enter feet of first distance: 3
Enter feet of second distance: 4
Enter inch of first distance : 13
Enter inch of second distance : 7
The distance is 8 feet and 8 inches.
```


Q(12) Write a program to count the total number of created objects.

→

```
#include<iostream.h>
class A
{
    private:
        static int count;
    public:
        A()
        {
            count++;
        }
        static int ans()
        {
            return count;
        }
};
int A::count=0;
int main()
{
    A a1,a2,a3,a4;
    int cnt;
    cnt = A::ans();
    cout<<"The number of objects are :"<<cnt;
    return 0;
}
```

⊙ Output :

```
The number of objects are : 4
```

Q13) Write a class customer to store records of 100 customer's company name "ABC" including following data members:
customer id, customer name, customer address.
Member function : getdata(), display().

→
```
#include<iostream.h>
#include<string.h>
class customer
{
    private:
        int id;
        string name;
        string address;
        string com-name;
    public:
        customer()
        {
            com-name="ABC";
        }
        void getdata()
        {
            cout<<"Enter the customer's name:";
            cin>>name;
            cout<<"Enter the customer's id:";
            cin>>id;
            cout<<"Enter the customer's address:";
            cin>>address;
        }
        void display()
        {
```



```
            cout<<"\n Company name -:"<<com-name;
            cout<<"\n Customer Id - "<<id;
            cout<<"\n Customer name - "<<name;
            cout<<"\n Customer address - "<<address;
        }
};
int main()
{
    customer c[100];
    int i;
    for (i=0; i<100; i++)
    {
        c[i].getdata();
    }
    cout<<"\n The customer details are \n";
    for (i=0; i<100; i++)
    {
        c[i].display();
    }
    return 0;
}
```

Q14) In which case default argument passing is not allowed? Explain with example.

→ ~~When default arguments are passed with values during~~
When values are passed to arguments during function definition, they are called as default arguments.

- These default values are used when other values are not passed to arguments.
- If we are giving default values to a variable, we also have to provide default value to the variable to its right.
- In case where the variable to the right is not provided default value, the concept of default value cannot be used.

eg)
```
#include<iostream.h>
#include<iomanip.h>
void sum (int a=10, b=20)
int main()
{
    int a=1, b=3;
    Sum();
    Sum(a, b);
    sum(a);
    Sum(b);
    return 0;
}

void sum( int a, int b)
{
    cout<<"a+b=" <<a+b<<endl;
}
```

⊙ Output:

a+b = 30

[redacted] b = 4

[redacted] = 21

a+b = 23

- The case sum(, b) will give an error because default argument is being passed to 'a' but not to 'b' which is to the right of 'a'.

Q15) What are the ~~base~~ case of defining member functions? Explain with example.

→ ~~(FUNCTION PROTOTYPING)~~. Inside main function and outside main function.

Q16) Explain following operators: new and delete.

→ The new operator requests for memory allocation in heap. If sufficient memory is available, it initializes the memory to the pointer variable and returns its address.

- A constructor that uses 'new' to store the memory of its objects, it is called as a dynamic constructor.
- The delete operator is used to deallocate the memory.
- Syntax for variable:

  new var_name;

  delete var_name;
- Syntax for block of variable:

  new[] var_name;

  delete[] var_name;

eg)
```
#include <iostream.h>
#include <iomanip.h>
class A
{
    public:
        A()
        {
            cout <<"Constructor" <<endl;
        }
};
int main()
{
    A *a = new A[4];
    delete[] a;
    return 0;
}
```

⊙ Output:
Constructor.
Constructor
Constructor
Constructor

Q17) Explain array of objects with example.
→ Page 58

Q6) (i)
```
#include<iostream.h>
class Add
{
    private:
        int a,b;
    public:
        Add()
        {
            cout<<"Enter the values of a and b:";
            cin>>a>>b;
        }
        void sum()
        {
            cout<<"The sum of values entered
            through the keyboard is:"<<a+b;
        }
};
int main()
{
    Add a1;
    a1.sum();
    return 0;
}
```

⊙ Output:
Enter the values of a and b:3
4
The sum of values entered through the keyword:7