UNIT II OBJECT ORIENTED PROGRAMMING CONCEPTS

String Handling – Copy Constructor - Polymorphism – compile time and run time polymorphisms –function overloading – operators overloading – dynamic memory allocation - Nested classes - Inheritance – virtual functions

String Handling

1. Correspondence between the C library and the C++ string Class

C Library Functions	C++ string operators/methods
#include <ctype.h>	#include <string>
Strcpy	= (the assignment operator)
Strcat	+= (assign+concat operator)
Strcmp	= =, !=, <, >, <=, >=
strchr, strstr strrchr	.find( ) method .rfind( ) method
Strlen	.size( ) or .length( ) methods
string - Array of chars that is null terminated (‘\0’).	string - Object whose string type is defined in the <string> file
Declaration : char cs[ ] = “Napoleon”;	Declaration : string s = “Napoleon”;
Strings are arrays	Strings are objects

2. a. Formatted Input: Stream extraction operator
   cin >> stringObject;
   a. Unformatted Input: getline function for a string
   getline(cin, s)
   does not skip over whitespace
   delimited by newline reads an entire line of characters into s

3. // This program demonstrates the C++ string class.

```
#include <iostream>
#include <string>
using namespace std;
void main(void)
{
        string str1, str2, str3;
        str1 = "ABC";
        str2 = "DEF";
        str3 = str1 + str2;
        cout << str1 << endl;
        cout << str2 << endl;
        cout << str3 << endl;
        str3 += "GHI";
        cout << str3 << endl;
}
```

Output :

```
ABC
DEF
ABCDEF
ABCDEFGHI
```

4. Comparison Operators for string Objects

- We can compare two strings x and y using the following operators: ==, !=, <, <=, >, >=
- The comparison is alphabetical
- The outcome of each comparison is: true or false
- The comparison works as long as at least x or y is a string object. The other string can be a string object, a C-style string variable, or a double-quoted string.

5. The Index Operator []
   If x is a string object, and you wish to obtain the value of the k-th character in the string, you write: x[k];

```
string x= "high";
char c=x[0]; // c is 'h'
c=x[1]; // c is 'i'
c=x[2]; // c is g
```

   This feature makes string objects appear like arrays of chars.

6. Declaration and concatenation in strings

```
string x= "high";
char y[]= "school";
char z[]= {'w','a','s','\0'};
char *p = "good";
string s= x+y+' '+z+" very"+" "+p+'!';
cout<<"s="<<s<<endl;
cout<<"s="+s<<endl;
```

   Output:
   s=highschool was very good!
   s=highschool was very good!

7. Getting a string Object Length & Checking for Emptiness
   To obtain the length of a string object x, call the method length() or size():

```
string x= "high";

int len=x.length( );
        --or--
int len=x.size( );
```

8. To check of x is empty (that is, has no characters in it):

```
bool x.empty();
```

9. Obtaining Substrings of Strings or copy

```
string y = x.substr(pos,len);

string s2 = "cathode";
```

10. s2.copy(str, 5, 2); //copy 5 characters into str
    //starting at index 2
    Or
    string str = s2.substr(2,5);

ans : thode

The default value of *len* is x.length( )

11. Inserting a String Inside Another

Suppose x is a string object, and let y be another string to be inserted at position *pos* of the string of x

To insert y, do: x.insert(*pos*,y);

```
string text = "This is a test";
string y =”maths”
text.insert(10,"maths");
or
text.insert(10,y);
```

output : "This is a maths test"

The argument y can be: a string object, a C-style string variable, or a double-quoted string

12. Replacing a String Inside Another

x.replace(*pos*,*len*,y);

```
string text = "This is a test";
text.replace (5,2,"was");
```

output : "This was a test"

13. Erasing Substrings

x.erase(*pos*,*len*);
x.clear( ); // To erase the whole string of x

```
string text = "This is a test";
text.erase (5,5);
```

output : "This test"

14. find() and rfind()

To search starting from a position *pos*, do :

```
int startLoc = x.find(y, pos);
```

To search for the rightmost occurrence of y in x, do

```
startLoc = x.rfind(y); // or
startLoc = x.rfind(y, pos);
```

15. string Characteristics - Member functions

s1.size() and s1.length()
Number of characters in a string
s1.capacity()
Number of elements that can be stored without reallocation
s1.max_size()
Maximum possible string size
s1.empty()
Returns true if empty
s1.resize(newlength)
Resizes string to newlength
s1.swap(s2);
Switch contents of two strings

16. String library functions
    toupper() – returns the uppercase equivalent of the argument
    tolower()– returns the lowercase equivalent of the argument

    atoi()- ascii to integer
    atol() – ascii to longint
    atof() – ascii to float

    x = atoi("123") - x value is 123
    y=atol("123456789") - y value is 123456789
    z=atof("3.14") – - z value is 3.14

    returns TRUE(1) or FALSE(0) for the following functions
    isaplha()
    isdigit()
    isalnum()
    isprint()
    ispunct()
    isupper()
    islower()
    isspace()

17. // This program demonstrates some of the character testing functions.

```
#include <iostream.h>
#include <ctype.h>
void main(void)
{
        char input;
        cout << "Enter any character: ";
        cin.get(input);
        cout << "The character you entered is: " << input << endl;
        cout << "Its ASCII code is: " << int(input) << endl;
        if (isalpha(input))
                cout << "That's an alphabetic character.\n";
        if (isdigit(input))
                cout << "That's a numeric digit.\n";
        if (islower(input))
                cout << "The letter you entered is lowercase.\n";
        if (isupper(input))
                cout << "The letter you entered is uppercase.\n";
        if (isspace(input))
                cout << "That's a whitespace character.\n";
}
```

Output:
Enter any character: A [Enter]
The character you entered is: A
Its ASCII code is: 65
That's an alphabetic character.
The letter you entered is uppercase.
Program Output With Other Example input
Enter any character: 7 [Enter]

```
The character you entered is: 7
Its ASCII code is: 55
That's a numeric digit.
```

Types of constructors

1. Default constructor
2. Parameterized constructor
3. Overloaded constructor
4. Copy constructor

1. Default constructors

```
#include<iostream.h>
#include<conio.h>
class rec
{
private:
int l,b,a;
public:
rec(int);
rec(int,int);
void calculate(void);
void putdata(void);
};

void rec::putdata()
{
cout<<"\nlength : "<<l<<endl;
cout<<"\nbreadth  : "<<b<<endl;
cout<<"\nArea :"<<a;
}
void rec::calculate(void)
{
a=l*b;
}

rec::rec(int x)
{
l=x;
b=1;
}
rec::rec(int x,int y)
{
l=x;
b=y;
}

void main()
{
clrscr();

rec r2(2);
r2.calculate();
r2.putdata();
```

```
getch();

rec r3(2,5);
r3.calculate();
r3.putdata();

getch();
}
```

Output:

```
Length : 2
Breadth :1
Area :2

Length : 2
Breadth :5
Area :10
```

Copy Constructor

```
#include<iostream.h>
#include<conio.h>
class simple
{
Int x,y;
public:
simple ()
{
}
simple (int a,int b)
{
x=a;
y=b;
}
simple (simple &o)
{
x=o.x;
y=o.y;
}
void disp()
{
cout<<"\n"<<"The numbers are"<<x<<" "<<y;
cout<<"\n"<<"Multiplication of two given numbers is "<<a*b;
}
};
void main()
{
clrscr();
simple s1(5,2);//call two argument constructor
simple s2(s1);//Implicitlt call copy constructor
simple s3=s1;//call copy constructor
simple s4;
s4=s1;//Explicitly call copy constructor
s1.disp();
```

```
s2.disp();
s3.disp();
s4.disp();
getch();
}
```

Output:

```
The two numbers are 5  2
Multiplication of two given numbers is 10
The two numbers are 5 2
Multiplication of two given numbers is 10
The two numbers are 5 2
Multiplication of two given numbers is 10
The two numbers are 5 2
Multiplication of two given numbers is 10
```

Polymorphism

1. Compile time Polymorphism – Function overloading and operator overloading
2. Run time polymorphisms- Virtual functions

`Function overloading` - functions with same name, but different number of arguments

Eg.
```
void Print(string s);//Print string
void Print(int i);//Print integer
```

`Function overriding` - concept of inheritance. Functions with same name and same number of arguments. Here the second function is said to have overridden the first

Eg.

```
class Stream//A stream of bytes
{
public virtual void Read();//read bytes
}

class FileStream:Stream//derived class
{
public override void Read();//read bytes from a file
}
class NetworkStream:Stream//derived class
{
public override void Read();//read bytes from a network
}
```

1.a. Function overloading

```
#include<iostream.h>
#include<conio.h>
void exchange (float &c,float &d)
{
float t;
t=c;
c=d;
```

```
d=t;
}
void exchange(int &a,int &b)
{
int t;
t=a;
a=b;
b=t;
}
void exchange (char &e,char &f)
{
char t;
t=e;
e=f;
f=t;
}
void main()
{
int p,q;
float r,s;
char t,u;
clrscr();
cout<<"\n Enter two integers : \n";
cout<<" p = ";
cin>>p;
cout<<"\n q = ";
cin>>q;
exchange (p,q);
cout<<"\n Enter two float numbers : \n";
cout<<" r = ";
cin>>r;
cout<<"\n s = ";
cin>>s;
exchange (r,s);
cout<<"\n Enter two Characters : \n";
cout<<" t = ";
cin>>t;
cout<<"\n u = ";
cin>>u;
exchange (t,u);
cout<<"\n AFTER SWAPPING \n";
cout<<" \n INTEGERS p = "<<p<<"\t q =  "<<q;
cout<<" \n FLOAT NUMBERS r= "<<r<<"\t s =  "<<s;
cout<<" \n CHARACTERS t = "<<t<<"\t u =  "<<u;
getch();
}
```

Output:

```
Enter two integers :
 p = 1
 q = 2
 Enter two float numbers :
 r = 2.1
 s = 3.1
 Enter two Characters :
```

```
t = x
u = y
AFTER SWAPPING
INTEGERS p = 2  q =  1
FLOAT NUMBERS r= 3.1    s =  2.1
CHARACTERS t = y       u =  x
```

1.b. Operators overloading

UNARY OPERATOR OVERLOADING (MEMBER FUNCTION)

Program:

```
#include<iostream.h>
#include<conio.h>
class unary
{
private:
int x,y,z;
public:
unary(void)
{
cout<< "Enter Any Three Integer Nos. : ";
cin>>x>>y>>z;
}
void display(void)
{
cout<< endl<< " The Three Nos. Are : "<< x<< " , "<< y<< " , "<< z;
}
void operator --()
{
x = --x;
y = --y;
z = --z;}
void operator ++()
{
x = ++x;
y = ++y;
z = ++z;
}
};
void main()
{
clrscr();
unary s;
s.display();
--s;
cout<< endl<< endl<< endl<< "******* The Decremented Values ********"<< endl;
s.display();
++s;
cout<< endl<< endl<< endl<< "******* The Incremented Values ********"<< endl;
s.display();
getch();
}
```

Output:

```
Enter Any Three Integer Nos. :
7        8        9
 The Three Nos. Are : 7, 8 , 9
******* The Decremented Values ********
 The Three Nos. Are : 6 , 7 , 8
******* The Incremented Values ********
 The Three Nos. Are : 7 , 8 , 9
```

BINARY OPERATOR OVERLOADING (NON MEMBER FUNCTION)

Aim:

To write a c++ program to implement the concept of operator overloading using friend function.

Program:

```
#include<iostream.h>
#include<conio.h>
class complex
{
private:
float real;
float img;
public:
complex()
{
real=0.0;
img=0.0;
}
complex(float p,float q)
{
real=p;
img=q;
}
friend complex operator +(complex,complex);
void show()
{
cout<<"\n"<<real<<"+-i"<<img<<"\n";
}
};
complex operator +(complex s1,complex s2)
{
complex s3;
s3.real=s1.real+s2.real;
s3.img=s1.img+s2.img;
return(s3);
}
void main()
{
clrscr();
complex c1(1.1,2.1);
complex c2(3.1,4.1);
```

```
complex c3;
c3=c1+c2;
cout<<"\n FIRST COMPLEX NUMBER = ";
c1.show();
cout<<"\n SECOND COMPLEX NUMBER = ";
c2.show();
cout<<"\n ADDED COMPLEX NUMBER = ";
c3.show();
getch();
}
```

Output:
FIRST COMPLEX NUMBER = 1.1+-i2.1
SECOND COMPLEX NUMBER = 3.1+-i4.1
ADDED COMPLEX NUMBER = 4.2+-i6.2

Inheritance

1. Single level Inheritance
2. Multi level Inheritance
3. Multiple Inheritance
4. Hierarchical Inheritance
5. Hybrid level Inheritance

Base class visibility mode	Derived class visibility mode		
	private	public	protected
private	Not inherited	Not inherited	Not inherited
public	private	public	protected
protected	private	protected	protected

Multiple Inheritance

```
//multiple inheritance
#include<iostream.h>
#include<conio.h>
class first
{
protected:
int regno;
int f1;
int f2;
int tot1;
float avg1;
public:
void getdata(void)
{
cout <<"\n enter reg.no, 2 marks: ";
cin>>regno>>f1>>f2;
}
void putdata(void)
{
cout <<"\n reg.noand  2 marks of first sem : ";
cout<<regno<<"\t"<<f1<<"\t"<<f2;
tot1=f1+f2;
avg1 = tot1/2;
cout<<"\n 1.total="<<tot1<<"\t 1.Average ="<<avg1;
```

```
}
};

class sec
{
protected:
int s1;
int s2;
int tot2;
float avg2;

public:
void getdata(void)
{
cout <<"\n enter second sem 2 marks: ";
cin>>s1>>s2;
}
void putdata(void)
{
cout <<"\n  marks of second  sem : ";
cout<<"\t"<<s1<<"\t"<<s2;
tot2=s1+s2;
avg2 = tot2/2;
cout<<"\n 2.total="<<tot2<<"\t 2.Average ="<<avg2;
}
};


class third
{
protected:
int t1;
int t2;
int tot3;
float avg3;
public:
void getdata(void)
{
cout <<"\n enter third sem 2 marks: ";
cin>>t1>>t2;
}

void putdata(void)
{
cout <<"\n  marks of third  sem : ";
cout<<"\t"<<t1<<"\t"<<t2;
tot3=t1+t2;
avg3 = tot3/2;
cout<<"\n 3.total="<<tot3<<"\t 3.Average ="<<avg3;
}
};

class overall:public first,public sec,public third
{
int tot;
```

```
float avg;
public:
void getdata()
{
first::getdata();
sec::getdata();
third::getdata();
}
void putdata()
{
first::putdata();
sec::putdata();
third::putdata();
tot=tot1+tot2+tot3;
avg = tot/6;
cout<<"\n overall total="<<tot<<"\t overall Average ="<<avg;

}


};

void main()
{
clrscr();
first p1;
p1.getdata();
p1.putdata();
getch();
sec p2;
p2.getdata();
p2.putdata();
getch();
third p3;
p3.getdata();
p3.putdata();
getch();
overall p4;
p4.getdata();
p4.putdata();
getch();

}
```

Multi level inheritance

```
//multi level inheritance
#include<iostream.h>
#include<conio.h>
class first
{
protected:
char name[10];
int regno;
public:
void getdata(void)
```

```
{
cout <<"\n enter name and reg.no, : ";
cin>>name>>regno;
}
};

class sec:public first
{
protected:
int s1;
int s2;
int tot;
public:
void getdata(void)
{
first::getdata();
cout <<"\n enter  2 marks: ";
cin>>s1>>s2;
tot=s1+s2;
}
void putdata(void)
{

 cout <<"\n "<<name<<"\t"<<regno<<"\t"<<tot;
 //cout <<"\n Address  "<<add;
}
};

class third:public first
{
protected:
char add[10];
public:
void getdata(void)
{

first::getdata();
cout <<"\n enter address: ";
cin>>add;
}
void putdata(void)
{

// cout <<"\n Total = "<<tot;
cout <<"\n "<<name<<"\t"<<regno<<"\t";
 cout <<"\n Address  "<<add;
}
};


void main()
{
clrscr();

sec sp1;
```

```
third tp1;
sp1.getdata();
sp1.putdata();
tp1.getdata();
tp1.putdata();
getch();
}
```

Dynamic memory allocation

```
// pointers - 1
#include<iostream.h>
#include<conio.h>
int main()
{
        clrscr();
        int a =20;
        int *p;

        p=&a;
        cout << " \n value of a= "<<a;
        cout << " \n address of a= "<<&a;
        cout << " \n value of p= "<<p;
        cout << " \n value of *p= "<<*p;
        cout << " \n value of *p= "<<*p/2;
        getch();
        return 0;
}

// pointers - 2
#include<iostream.h>
#include<conio.h>
int main()
{
        clrscr();
        int  a[4] = {10,20,30,40};
        int  *p;
        p=a;
        for (int i=0;i<4;i++)
        {
        cout << " \n value of p= "<<p;
        cout << " \n value of a["<<i<<"]= "<<a[i];
        cout << " \n value of *p= "<<*p;
        p++;
        }
        getch();
        return 0;
}

// pointers - 3
#include<iostream.h>
#include<conio.h>
#include<malloc.h>
int main()
```

```
{
        clrscr();
        int *a=new int[5];
        int *c =a;
        for (int i=0;i<4;i++)
        {
        *a=i;
        cout << " \n value of a= "<<a;
        cout << " \n value of *a= "<<*a;
        a++;
        }
        a=c;
        for (i=0;i<4;i++)
        {
        cout << " \n value of a= "<<a;
        cout << " \n value of *a= "<<*a;
        a++;
        }
        delete []a;
        a=c;
        for (i=0;i<4;i++)
        {
        cout << " \n value of a= "<<a;
        cout << " \n value of *a= "<<*a;
        a++;
        }
        getch();
        return 0;
}


// pointers to objects- 4
#include<iostream.h>
#include<conio.h>
class person
{
private:
char name[20];
int rollno;
int m1;
int m2;
int tot;
public:
void getdata(void);
void calculate(void);
void putdata(void);
};
void person::putdata(void)
{
cout<<"\nName : "<<name<<endl;
cout<<"\nRollNumber : "<<rollno<<endl;
cout<<"\nTotal :"<<tot;
}
void person::calculate(void)
{
```

```
tot=m1+m2;
}
void person::getdata(void)
{
cout<<"\nEnter Name : ";
cin>>name;
cout<<"\nEnter RollNumber : ";
cin>>rollno;
cout<<"\nEnter 2 marks : ";
cin>>m1>>m2;
}
void main()
{
clrscr();
person *s1,*s2;
s1->getdata();
s1->calculate();
s1->putdata();
/*s2.getdata();
s2.calculate();
s2.putdata(); */
getch
();
}
```

This pointer

```
#include<iostream.h>

class Test
{
private:
  int x;
public:
  void setX (int x)
  {
     // The 'this' pointer is used to retrieve the object's x
     // hidden by the local variable 'x'
     this->x = x;
          //x=x;
  }
  void print() { cout << "x = " << x << endl; }
};

int main()
{
  Test obj;
  int x = 20;
  obj.setX(x);
  obj.print();
  return 0;
}
```

// this pointer – for objects

```
# include<iostream.h>
```

```
#include<conio.h>
class Myclass
{
int i;
float f;
public:
Myclass(int x, float y)
{
i=x;
f=y;
}
Myclass operator ++()
{
i=i+1;
f=f+1.0;
return *this;
}
void show()
{
cout<<i<<"    "<<f<<endl;
}
};
 int main()
 {
 Myclass a(4,6.5);
 clrscr();
 a.show();
 ++a;
 a.show();
 getch();
 return 0;
 }
```

virtual functions

```
// VIRTUAL FUNCTIONS
#include<iostream.h>
#include<conio.h>
class Base
{
        int a;
        public:
        Base()
        {
                a = 1;
        }
        void display()
        {
                cout<<"\n Base class display()  is invoked";

        }
        virtual void show()
        {
                cout<<"\n Base class show() is invoked";
                cout << a<<"\n";
```

```
        }
};
class Derived: public Base
{
        int b;
        public:
        Derived()
        {
                b = 2;
        }
        void display()
        {
                cout<<"\n Derived class display() is invoked";

        }
        void show()
        {
                cout<<"\n Derived class show() is invoked";
                cout << b<<"\n";
        }
};
int main()
{
        clrscr();
          Base B;
          Derived D;
          Base *bp;
          B.show();
          B.display();
          D.show();
          D.display();


          bp=&B;
          bp->show();
          bp->display();
          bp = &D;
          bp->show();
          bp->display();
          getch();
          return 0;
}
```

Abstract class and true virtual functions

// TRUE VIRTUAL FUNCTIONS

```
#include<iostream.h>
#include<conio.h>
class Base
{       protected:
          int a;
          public:
          Base()
          {
                  a = 1;
          }
```

```
        void display()
        {
                cout<<"\n Base class display()  is invoked";

        }
        virtual void show() =0;
        };
class Derived: public Base
{
         int b;
         public:
         Derived()
         {
                b = 2;
         }
         void display()
        {
                 cout<<"\n Derived class display() is invoked";

        }
         void show()
         {
                 cout<<"\n No base class show. only Derived class show() is invoked";
                cout << b<<"\n";
         }
};
int main()
{
        clrscr();
          //Base B;  - cannot create objects for base class
          Derived D;
          Base *bp;
          D.show();
          D.display();
          bp = &D;
          bp->show();
          bp->display();
          getch();
          return 0;
}
```

Pointer to functions

```
// pointers to functions
#include<iostream.h>
#include<conio.h>

typedef void (*fnptr) (int,int);
void add(int x, int y)
{
cout<<"\n sum = "<<x+y;
}
void sub(int x, int y)
{
```

```
cout<<"\n difference = "<<x-y;
}

int main()
{
  fnptr p;
  clrscr();
  int a,b;
  cout<<"enter 2 integer values";
  cin>>a>>b;
  p=&add;
  p(a,b);
//   p(8,2);
  p=&sub;
  p(a,b);
 // p(9,4);
  getch();
  return 0;
}
```

Friend function

```
// friend function
#include<iostream.h>
#include<conio.h>
class person
{
private:
char name[20];
int rollno;
int m1;
int m2;
int tot;
public:
void getdata(void);
void calculate(void);
friend void putdata(person p);
};
void person::getdata(void)
{
cout<<"\nEnter Name : ";
cin>>name;
cout<<"\nEnter RollNumber : ";
cin>>rollno;
cout<<"\nEnter 2 marks : ";
cin>>m1>>m2;
}
void person::calculate(void)
{
tot=m1+m2;
}
void putdata(person p)
{
cout<<"\nName : "<<p.name<<endl;
cout<<"\nRollNumber : "<<p.rollno<<endl;
```

```
cout<<"\nTotal :"<<p.tot;
}
void main()
{
clrscr();
person s1,s2;
s1.getdata();
s1.calculate();
putdata(s1);
getch();
}
```