Name: Pratham Chellani

Rollno:33

Subject: Object Oriented Computer Programming Practical

Classes, Objects, Functions

1. Write a program to create class Student with student's rollno, name and marks of three subjects (OOCP, DBMS and English) and display the details of student with total marks of all subjects along with the percentage in proper format.(Output should be in descending order of percentage.

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

class student
{
      int roll_no;
      string name;
      float oocp, dbms, eng, total, avg;

      public:
            void getdata(int r, string n, float o, float d, float
e)
            {
                  roll_no = r;
                  name = n;
                  oocp = o;
                  dbms = d;
                  eng = e;
                  total = o + d + e;
                  avg = (total/300)*100;
            }
            float gettotal()
            {
                  return total;
```

```
            }
            float getavg()
            {
                  return avg;
            }
            void display()
            {
                  cout<<endl<<roll_no<<" "<<name<<" "<<total<<"
"<<avg;
            }
};

int main()
{
      student s1,s2,s3;
      int i,j;
      s1.getdata(1, "tanix", 50, 75, 25);
      s2.getdata(1, "rishi", 80, 70, 50);
      s3.getdata(1, "varun", 90, 100, 95);

      student stud[]={s1,s2,s3};

      for(i=0;i<3;i++){
            for(j=i+1;j<3;j++){
                  if(stud[j].getavg()>stud[i].getavg()){
                        student temp = stud[j];
                        stud[j]=stud[i];
                        stud[i]= temp;
                  }
      }
}

for(i=0;i<3;i++){
      stud[i].display();
```

```
    }

    return 0;
}
```

2. Write a program to create class Num (int n1, int n2, int n3, int n4). Display total and average of n1, n2, n3 and n4.

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

class num{
    float number1,number2,number3,number4;
    float total,avg;
    public:
        void getdata(){
            cout<<endl<<"enter number1: ";
            cin>>number1;
            cout<<endl<<"enter number2: ";
            cin>>number2;
            cout<<endl<<"enter number3: ";
            cin>>number3;
            cout<<endl<<"enter number4: ";
            cin>>number4;
        }
        void display(){
            cout<<endl<<"total "<<number1+number2+number3+number4;

            cout<<endl<<"avg "<<(number1+number2+number3+number4)/4;
        }
```

```
};

int main(){
   num n1;
   n1.getdata();
   n1.display();
   return 0;
}
```

3. Write a program to create class Time (int h, int m). Read a value as minutes from user to display new time after adding the value to minutes in Time.

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

class time{
      int h,m,extra;
      int a,b;
      public:
            void getdata(){
                  cout<<endl<<"enter minutes: ";
                  cin>>m;
                  cout<<endl<<"enter minutes to add: ";
                  cin>>extra;
            }
            void display(){
                  cout<<endl<<h<<":"<<m;
            }
            void time1(){
                  m=m+extra;
                  h=h+(m/60);
                  m %= 60;
                  h %= 24;
```

```
            }
};

int main(){
      time t1;
      t1.getdata();
      t1.time1();
      t1.display();
      return 0;
}
```

4. Write a program to create class Date (int day, int month, int year). Read a value as day from user to display new date after adding the value to day in Date.

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

class date{
   int day,month,year;
   int eday;
   public:
        void getdata(){
             cout<<endl<<"enter day: ";
             cin>>day;
             cout<<endl<<"enter month: ";
             cin>>month;
             cout<<endl<<"enter year: ";
             cin>>year;
             cout<<endl<<"enter extra days to add: ";
```

```
            cin>>eday;
            cout<<endl<<"present date "<<day<<"-
"<<month<<"-"<<year;
        }
        void display(){
            cout<<endl<<"new date "<<day<<"-"<<month<<"-
"<<year;
        }
        void add(){
            day=day+eday;
            month=month+(day/30);
            year=year+(month/12);

            day %= 30;
            month %= 12;

        }
};

int main(){
   date d1;
   d1.getdata();
   d1.add();
   d1.display();
   return 0;
}
```

5. Write a program to create class employee with employee's id, name and basic salary. Calculate gross salary for each employee(HRA 20%, DA 30%, OA 10%).

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

class employee{
     int id;
     string name;
     float b_salary;
     float g_salary, hra, da, oa;
     public:
          void getdata(){
               cout<<endl<<"enter id: ";
               cin>>id;
               cout<<endl<<"enter name: ";
               cin>>name;
               cout<<endl<<"enter salary: ";
               cin>>b_salary;
          }
          void display(){
               cout<<endl<<id;
               cout<<endl<<name;
               cout<<endl<<b_salary;
               hra=(b_salary*0.2);
               da=(b_salary*0.3);
               oa=(b_salary*0.1);
               g_salary=b_salary+hra+da+oa;
               cout<<endl<<g_salary;
          }
};
```

```
int main(){
    employee e1;
    e1.getdata();
    e1.display();
    return 0;
}
```

6. Write a program to define a class called book. Write a program to read information about 10 books and display books details in ascending order of price in proper format.

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

class book{
    int price;
    string name;
    public:
        void getdata(int p, string n){
            price=p;
            name=n;
        }
        int getprice(){
            return price;
        }
        void display(){
            cout<<endl<<"name: "<<name;
            cout<<endl<<"price: "<<price;
        }
};
```

```
int main(){
     book b1,b2,b3,b4,b5,b6,b7,b8,b9,b10;
     b1.getdata(1000,"c++");
     b2.getdata(1500,"dbms");
     b3.getdata(2000,"frontend");
     b4.getdata(200,"backend");
     b5.getdata(900,"maths");
     b6.getdata(500,"DE");
     b7.getdata(3500,"linux");
     b8.getdata(2000,"open office");
     b9.getdata(200,"eng");
     b10.getdata(1900,"evs");
     book books[]={b1,b2,b3,b4,b5,b6,b7,b8,b9,b10};
     int i,j;
     for(i=0;i<10;i++){
          for(j=i+1;j<10;j++){

     if(books[j].getprice()>books[i].getprice()){
                    book temp = books[i];
                    books[i] = books[j];
                    books[j] = temp;
               }
          }
     }

     for(i=0;i<10;i++){
          books[i].display();
     }
     return 0;
```

```
}
```

7. Demonstrate the use of static variables in a class by using it to count the number of times the value is being inputted in the program.

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

class student{
     int value;
     int count;
     public:
          void inputvalue(){
               cout<<"enter value: ";
               cin>>value;
               count++;
          }
          void display(){
               cout<<endl<<"total times: "<<count;
          }
};
int main(){
     student s1;
     s1.inputvalue();
     s1.inputvalue();
     s1.inputvalue();
     s1.display();
     return 0;
}
```

8. Create class STUDENT having rollno, name and age as data members, also take subject with three subjects and initialize their value with minimum passing marks. Using member function, modify marks of student with specific rollno which is given by user.

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

class Student{
     int rollno;
     string name;
     int age;
     int marks[3];

     public:
          Student(int rn,string sname,int a,int m1=35,int m2=35,int m3=35){
               rollno=rn;
               name=sname;
               age=a;
               marks[0]=m1;
               marks[1]=m2;
               marks[2]=m3;
          }

          void change(int r,int new_marks[]){
               if(rollno=r){
                    marks[0]=new_marks[0];
```

```
                    marks[1]=new_marks[1];
                    marks[2]=new_marks[2];
                    cout << "Marks updated for roll
number " << rollno << "." << endl;
        }
          else {
            cout << "Roll number not found." << endl;
          }
          }


     void display(){
          cout << "Roll No: " << rollno << endl;
        cout << "Name: " << name << endl;
        cout << "Age: " << age << endl;
        cout << "Marks: " << marks[0] << ", " << marks[1]
<< ", " << marks[2] << endl;
     }
};


int main() {
    Student student1(2, "Rahul", 45);

    student1.display();


    int new_marks[3] = {75, 80, 90};
    student1.change(2, new_marks);
```

```
        student1.display();

        return 0;}
```

9. Define a class to represent a bank account. Include the following members :

DATA MEMBERS MEMBER FUNCTIONS

---------- ----------

Name of depositor (1) To assign initial values

Account Number (2) To Deposit the amount

Type of Account (3) To withdraw an amount after checking the

Balance amount in account (4) To display name and balance

Write C++ program to handle 10 customers.

```
#include <iostream>
#include <string>

using namespace std;
class Bank{
     string name;
     int acc_no;
     string acc_type;
     double balance;

     public:
          Bank(string depname,int acc_number,string
atype,double starting_balance){
               name=depname;
               acc_no=acc_number;
```

```
            acc_type=atype;
            balance=starting_balance;
        }

        void deposit(double amount){
            if(amount > 0){
                balance+=amount;
                cout<<"Deposited :"<<amount<<" New balance
: "<<balance<<endl;
            }
            else{
                cout<<"Invalid deposit amount. Deposit
amount must be positive"<<endl;
            }
        }

        void withdraw(double amount){
            if (amount >0){
                if (amount<=balance){
                    balance-=amount;
                    cout << "Withdrew " << amount << ".
New balance: " << balance << endl;
                }
                else{
                    cout << "Insufficient balance." <<
endl;
                }}
            else{
                    cout<<"Invalid withdrawl amount.
Withdraw amount must be positive"<<endl;
                }
```

```
                }


          void display() const {
        cout << "Name: " << name << endl;
        cout << "Balance: " << balance << endl;
    }
};



int main(){
     Bank account("Rishi Gangwani", 123456, "Savings",
20000.0);
     account.deposit(2000.0);
     account.withdraw(5000.0);
     account.display();
     return 0;
}
```

10. Write a program to create class ‘Search’ having data members (int a[ ], x) and define member functions as void input(), void output(), void search(int position), void add(int value) to display result.

```
include<iostream>
using namespace std;

class Search{
    int a[5];
    int x;
    public:
```

```
void input(){
    for (int i = 0; i < 5; i++)
    {
        int temp;
        cout << "enter a value: " << endl;
        cin >> temp;
        a[i] = temp;
    }
    int xVal;
    cout << "enter value for x: " << endl;
    cin >> xVal;
    x = xVal;
}
void output(){
    for (int i = 0; i < 5; i++)
    {
        cout << a[i] << endl;
    }
    cout << x << endl;
}
void search(int pos){
    if (pos > 5)
    {
        cout << "out of index" << endl;
    }
    else{
        for (int i = 0; i < 5; i++)
        {
            if (pos-1 == i)
```

```
                    {
                        cout << a[pos-1] << endl;
                    }
                }
            }
        }
        void add(int value){
            x += value;
            cout << x << endl;
        }
};

int main(){
    Search s1;
    s1.input();
    s1.search(3);
    s1.add(5);
    s1.output();
    return 0;
}
```

Inheritance

(1) Define a class Employees. Also define classes of MaleEmp and FemaleEmp inheriting from that. Define classes Officers, Clercks and peons again inheriting from Employee class. Define an array which contains 10 different types of employees. Define a function ReadDetails() in all above classes. All array elements should be able to be accessed in the same routine irrespective of their type.

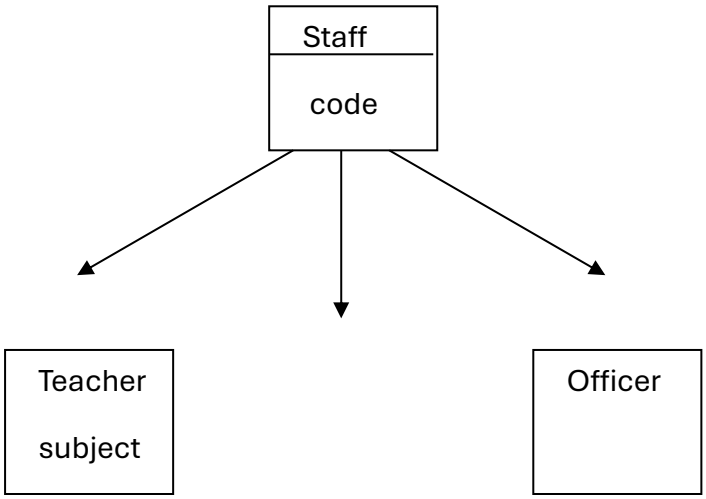
```
#include <iostream>
#include <string>
using namespace std;

class Employee {
protected:
    string name;
    int emp_id;

public:
    Employee(string name, int emp_id) : name(name), emp_id(emp_id) {}
    virtual void ReadDetails() {
        cout << "Employee ID: " << emp_id << ", Name: " << name << endl;
    }
    virtual ~Employee() {}
};

class MaleEmp : public Employee {
protected:
```

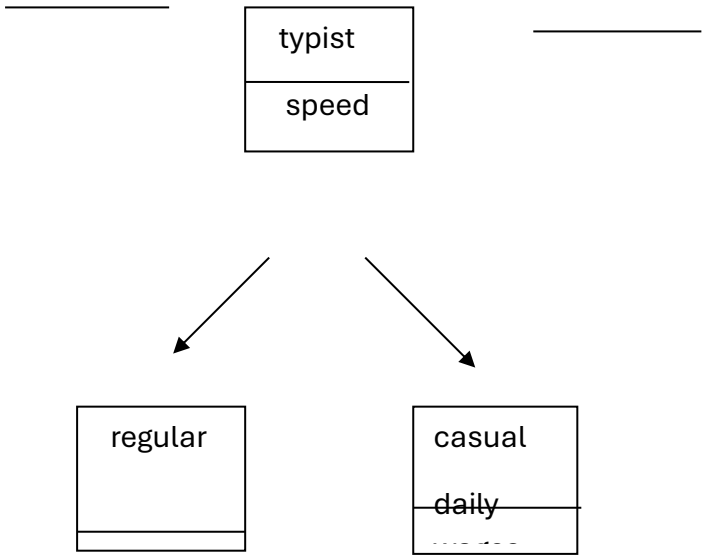
```
    string gender;

public:
    MaleEmp(string name, int emp_id) : Employee(name, emp_id), gender("Male") {}
    void ReadDetails() override {
        cout << "Employee ID: " << emp_id << ", Name: " << name << ", Gender: " << gender << endl;
    }
};

class FemaleEmp : public Employee {
protected:
    string gender;

public:
    FemaleEmp(string name, int emp_id) : Employee(name, emp_id), gender("Female") {}
    void ReadDetails() override {
        cout << "Employee ID: " << emp_id << ", Name: " << name << ", Gender: " << gender << endl;
    }
};

class Officer : public Employee {
public:
    Officer(string name, int emp_id) : Employee(name, emp_id) {}
    void ReadDetails() override {
        cout << "Officer - Employee ID: " << emp_id << ", Name: " << name << endl;
```

```
    }
};

class Clerk : public Employee {
public:
    Clerk(string name, int emp_id) : Employee(name, emp_id)
{}
    void ReadDetails() override {
        cout << "Clerk - Employee ID: " << emp_id << ", Name:
" << name << endl;
    }
};

class Peon : public Employee {
public:
    Peon(string name, int emp_id) : Employee(name, emp_id)
{}
    void ReadDetails() override {
        cout << "Peon - Employee ID: " << emp_id << ", Name:
" << name << endl;
    }
};

int main() {
    Employee* employees[10] = {
        new MaleEmp("Arjun Sharma", 1001),
        new FemaleEmp("Priya Mehra", 1002),
        new Officer("Ravi Kumar", 1003),
        new Clerk("Sunita Singh", 1004),
        new Peon("Rajesh Yadav", 1005),
        new MaleEmp("Amit Joshi", 1006),
```

```
            new FemaleEmp("Anjali Verma", 1007),
            new Officer("Vikas Pandey", 1008),
            new Clerk("Manoj Gupta", 1009),
            new Peon("Suresh Reddy", 1010)
        };

        for (int i = 0; i < 10; i++) {
            employees[i]->ReadDetails();
        }

        for (int i = 0; i < 10; i++) {
            delete employees[i];
        }
        return 0;
    }
```

(2) An educational institution wishes to maintain a database of its employees. The database is divided into a number of classed whose hierarchical relationships are shown in fig-1. The figure also shows the minimum information required for each class. Specify all the classes and define function to create the database and retrieve individual information as and when required. Write parameterized constructor for each class in the hierarchy.





(Fig – 1)

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

class Staff {
protected:
    int code;
    string name;

public:
    Staff(int c, string n) : code(c), name(n) {}

    virtual void displayDetails() const {
```

```
        cout << "Staff Code: " << code << endl;
        cout << "Name: " << name << endl;
    }

    virtual ~Staff() {}
};

class Officer : public Staff {
protected:
    string grade;

public:
    Officer(int c, string n, string g) : Staff(c, n),
    grade(g) {}

    void displayDetails() const override {
        Staff::displayDetails();
        cout << "Grade: " << grade << endl;
    }
};

class Teacher : public Staff {
protected:
    string subject;
    string publication;

public:
    Teacher(int c, string n, string subj, string pub)
```

```
        : Staff(c, n), subject(subj), publication(pub) {}

    void displayDetails() const override {
        Staff::displayDetails();
        cout << "Subject: " << subject << endl;
        cout << "Publication: " << publication << endl;
    }
};

class Typist : public Staff {
protected:
    int speed;

public:
    Typist(int c, string n, int spd) : Staff(c, n),
    speed(spd) {}

    void displayDetails() const override {
        Staff::displayDetails();
        cout << "Typing Speed: " << speed << " wpm" <<
    endl;
    }
};

class CasualTypist : public Typist {
protected:
    float daily_wages;
```

```
public:
    CasualTypist(int c, string n, int spd, float wages)
        : Typist(c, n, spd), daily_wages(wages) {}

    void displayDetails() const override {
        Typist::displayDetails();
        cout << "Daily Wages: $" << daily_wages << endl;
    }
};

class RegularTypist : public Typist {
public:
    RegularTypist(int c, string n, int spd) : Typist(c,
    n, spd) {}

    void displayDetails() const override {
        Typist::displayDetails();
        cout << "Type: Regular Typist" << endl;
    }
};

int main() {
    Officer officer1(1, "Ankur Bhatia", "A");
    Teacher teacher1(2, "Priya Rai", "Mathematics",
    "Calculus Fundamentals");
    Typist typist1(3, "Rohit Sharma", 78);
    CasualTypist casualTypist1(4, "Virat Kohli", 56,
    50.5);
```

```
    RegularTypist regularTypist1(5, "Liam Livingston",
    85);

    cout << "Officer Details:" << endl;
    officer1.displayDetails();
    cout << endl;

    cout << "Teacher Details:" << endl;
    teacher1.displayDetails();
    cout << endl;

    cout << "Typist Details:" << endl;
    typist1.displayDetails();
    cout << endl;

    cout << "Casual Typist Details:" << endl;
    casualTypist1.displayDetails();
    cout << endl;

    cout << "Regular Typist Details:" << endl;
    regularTypist1.displayDetails();
    cout << endl;

    return 0;
}
```

(3) The database created in Ex-2 does not include educational information of the staff. It has been decided to add this

information to teachers and officers (and not for typists) which will help the management in decision making with regards to training, promotion , etc. Add another data class called education that holds two pieces of educational information , namely highest qualification in general education and highest professional qualification. This class should be inherited by the class teacher and officer. Modify the program of Ex 2 to incorporated these additions.

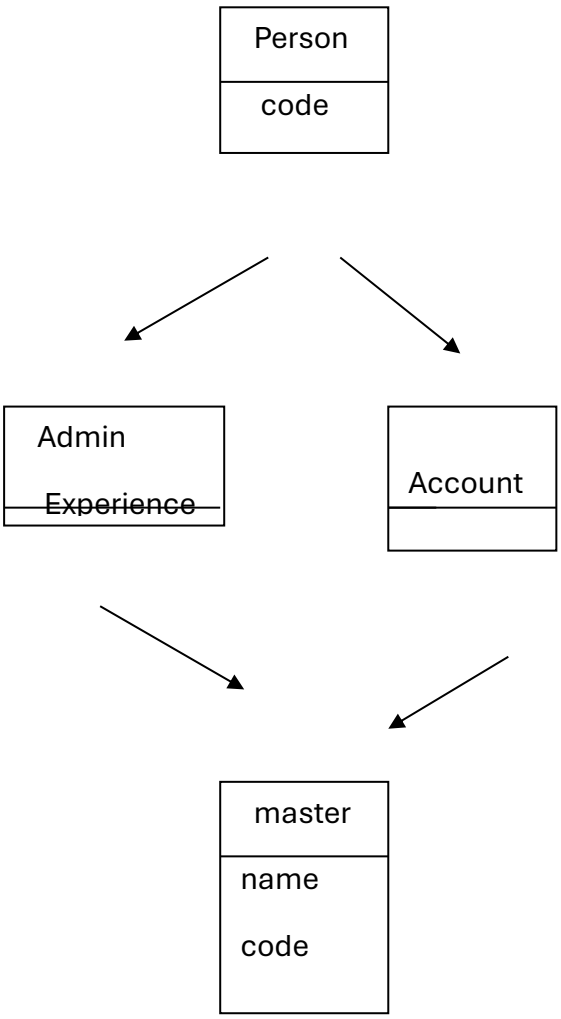
```
#include <iostream>
#include <string>
using namespace std;

class Staff {
protected:
    int code;
    string name;

public:
    Staff(int c, string n) : code(c), name(n) {}

    virtual void displayDetails() const {
        cout << "Staff Code: " << code << endl;
        cout << "Name: " << name << endl;
    }

    virtual ~Staff() {}
};

class Education {
protected:
    string highestQualification;
    string professionalQualification;

public:
    Education(string hQual, string pQual)
        :                       highestQualification(hQual),
professionalQualification(pQual) {}

    void displayEducation() const {
        cout  <<  "Highest  General  Qualification:  "  <<
highestQualification << endl;
        cout << "Highest Professional Qualification: " <<
professionalQualification << endl;
    }
};

class Officer : public Staff, public Education {
protected:
    string grade;
```

```
public:
    Officer(int c, string n, string g, string hQual, string
pQual)
        : Staff(c, n), Education(hQual, pQual), grade(g) {}

    void displayDetails() const override {
        Staff::displayDetails();
        cout << "Grade: " << grade << endl;
        displayEducation();
        cout << endl;
    }
};

class Teacher : public Staff, public Education {
protected:
    string subject;
    string publication;

public:
    Teacher(int c, string n, string subj, string pub, string
hQual, string pQual)
        :    Staff(c,    n),    Education(hQual,    pQual),
subject(subj), publication(pub) {}

    void displayDetails() const override {
        Staff::displayDetails();
        cout << "Subject: " << subject << endl;
        cout << "Publication: " << publication << endl;
        displayEducation();
        cout << endl;
    }
};

class Typist : public Staff {
protected:
    int speed;

public:
    Typist(int  c,  string  n,  int  spd)  :  Staff(c,  n),
speed(spd) {}

    void displayDetails() const override {
        Staff::displayDetails();
        cout << "Typing Speed: " << speed << " wpm" << endl
<< endl;
    }
};

class CasualTypist : public Typist {
protected:
    float daily_wages;
```

```
public:
    CasualTypist(int c, string n, int spd, float wages)
        : Typist(c, n, spd), daily_wages(wages) {}

    void displayDetails() const override {
        Typist::displayDetails();
        cout << "Daily Wages: $" << daily_wages << endl;
    }
};

class RegularTypist : public Typist {
public:
    RegularTypist(int c, string n, int spd) : Typist(c, n,
spd) {}

    void displayDetails() const override {
        Typist::displayDetails();
        cout << "Position: Regular Typist" << endl << endl;
    }
};

int main() {
    Staff* employees[5] = {
        new  Officer(101,  "Ankur  Bhatia",  "A",  "MBA",
"Certified Accountant"),
        new Teacher(102, "Priya Rai", "Physics", "Nature
Publishing", "PhD", "Research Fellow"),
        new CasualTypist(103, "Rohit Sharma", 50, 15.0f),
        new RegularTypist(104, "Virat Kohli", 60),
        new Officer(105, "Liam Livingston", "B", "B.Sc",
"Project Management")
    };

    cout << "Employee Details: " << endl << endl;

    for (int i = 0; i < 5; i++) {
        employees[i]->displayDetails();
        delete employees[i];
    }

    return 0;
}
```

(4) Consider a class network of fig . The class master derives information from both account and admin classes which in turn derived derive information from the class person. Define all the four classes and write a program to create

, update and display the information contained in master objects.



(Fig – 2)

```
#include <iostream>
#include <cstring> // for strcpy and other string
    operations
using namespace std;


class Person {
```

```
protected:
    int code;
    char name[50];

public:
    Person(int c = 0, const char* n = "") : code(c) {
        strcpy(name, n);
    }

    virtual void inputDetails() {
        cout << "Enter code: ";
        cin >> code;
        cout << "Enter name: ";
        cin.ignore();
        cin.getline(name, 50);
    }

    virtual void displayDetails() const {
        cout << "Code: " << code << endl;
        cout << "Name: " << name << endl;
    }
};

class Admin : virtual public Person {
protected:
    int experience;

public:
```

```
    Admin(int c = 0, const char* n = "", int exp = 0) :
    Person(c, n), experience(exp) {}

    void inputDetails() override {
        Person::inputDetails();
        cout << "Enter experience (in years): ";
        cin >> experience;
    }

    void displayDetails() const override {
        Person::displayDetails();
        cout << "Experience: " << experience << " years"
    << endl;
    }
};

class Account : virtual public Person {
protected:
    float pay;

public:
    Account(int c = 0, const char* n = "", float p =
    0.0f) : Person(c, n), pay(p) {}

    void inputDetails() override {
        Person::inputDetails();
        cout << "Enter pay: ";
        cin >> pay;
    }
```

```
    void displayDetails() const override {
        Person::displayDetails();
        cout << "Pay: $" << pay << endl;
    }
};

class Master : public Admin, public Account {
public:
    Master(int c = 0, const char* n = "", int exp = 0,
    float p = 0.0f)
        : Person(c, n), Admin(c, n, exp), Account(c, n,
    p) {}

    void inputDetails() override {
        cout << "Enter details for Master:" << endl;
        Person::inputDetails();
        cout << "Enter experience (in years): ";
        cin >> experience;
        cout << "Enter pay: ";
        cin >> pay;
    }

    void displayDetails() const override {
        cout << "Details of Master:" << endl;
        Person::displayDetails();
        cout << "Experience: " << experience << " years"
    << endl;
        cout << "Pay: $" << pay << endl << endl;
```

```
    }
};

int main() {
    Master master;
    cout << "Creating Master details..." << endl;
    master.inputDetails();

    cout << "\nDisplaying Master details:" << endl;
    master.displayDetails();

    cout << "Updating Master details..." << endl;
    master.inputDetails();

    cout << "\nDisplaying Updated Master details:" <<
    endl;
    master.displayDetails();

    return 0;
}
```

(5) In Exercise 4, the classes teacher , Officer and typist are derived from the class staff. As we know, we can use container classes in place of inheritance in some situations. Redesign the program of Ex-8.21 such that the classes teacher, officer and typist contain the objects of staff. Write parameterized constructor for each class in the hierarchy.

```
#include <iostream>
#include <string>
```

```
using namespace std;

class Staff {
private:
    int code;
    string name;

public:
    Staff(int c = 0, string n = "") : code(c), name(n) {}

    void input() {
        cout << "Enter Code: ";
        cin >> code;
        cout << "Enter Name: ";
        cin.ignore();
        getline(cin, name);
    }

    void display() const {
        cout << "Code: " << code << endl;
        cout << "Name: " << name << endl;
    }
};

class Teacher {
private:
    Staff staff;
    string subject;
    string publication;
```

```
public:
    Teacher(int code = 0, string name = "", string sub = "",
string pub = "")
        : staff(code, name), subject(sub), publication(pub)
{}

    void input() {
        staff.input();
        cout << "Enter Subject: ";
        cin.ignore();
        getline(cin, subject);
        cout << "Enter Publication: ";
        getline(cin, publication);
    }

    void display() const {
        cout << "\nTeacher Details:" << endl;
        staff.display();
        cout << "Subject: " << subject << endl;
        cout << "Publication: " << publication << endl;
    }
};

class Officer {
private:
    Staff staff;
    int experience;

public:
```

```
    Officer(int code = 0, string name = "", int exp = 0)
        : staff(code, name), experience(exp) {}

    void input() {
        staff.input();
        cout << "Enter Experience (in years): ";
        cin >> experience;
    }

    void display() const {
        cout << "\nOfficer Details:" << endl;
        staff.display();
        cout << "Experience: " << experience << " years" <<
endl;
    }
};

class Typist {
private:
    Staff staff;
    int speed;

public:
    Typist(int code = 0, string name = "", int spd = 0)
        : staff(code, name), speed(spd) {}

    void input() {
        staff.input();
        cout << "Enter Typing Speed (words per minute): ";
        cin >> speed;
```

```
    }

    void display() const {
        cout << "\nTypist Details:" << endl;
        staff.display();
        cout << "Typing Speed: " << speed << " wpm" << endl;
    }
};

int main() {
    Teacher t(101, "Ankur Bhatia", "Mathematics", "Math
Journal");
    Officer o(102, "Priya Rai", 5);
    Typist typ(103, "Virat Kohli", 80);

    cout << "Input Teacher Details:" << endl;
    t.input();
    cout << "\nInput Officer Details:" << endl;
    o.input();
    cout << "\nInput Typist Details:" << endl;
    typ.input();

    cout << "\nDisplaying All Details:" << endl;
    t.display();
    o.display();
    typ.display();

    return 0;
}
```

Operator Overloading

(1) Overload all the four arithmetic operators to operate on a vector class and also the overload the * operator to multiply scalar values to the vector class. Overload the >> operator to input a vector and the << operator to display the vector in the form (10,20,.....). Also overload the [] operator to access the individual member of the vector. Use Dynamic memory allocation to achieve the solution. Write appropriate constructor and destructure for the class.

```
#include<iostream>

using namespace std;


class vector{


   int *element;


   int size;


   public:


   vector(int size)


   {


        {


        }


   }
```

```
{
      {
            delete[] element;
            {
            }
      }
      return *this;
}
void display()
{
      {
            cout<<"\n"<<element[i];
      }
}
vector operator+(vector &v)
```

```
{
     vector temp(size);
     {
     }
     return temp;
}
vector operator-(vector &v)
{
     vector temp(size);
     {
     }
     return temp;
}
vector operator*(vector &v)
```

```
{
      vector temp(size);
      {
      }
      return temp;
}
vector operator/(vector &v)
{
      vector temp(size);
      {
      }
      return temp;
}
vector operator*(int a)
{
```

```
      vector temp(size);

      {

      }

      return temp;

}

int operator[](int index)

{

      {

            return element[index];

      }

}

friend istream& operator>>(istream&,vector&);

friend ostream& operator<<(ostream&,vector&);

~vector()
```

```
   {
         delete[] element;
   }
};
istream& operator>>(istream &din , vector &v)
{
   {
         din>>v.element[i];
   }
   return din;
}
ostream& operator<<(ostream &dout,vector &v)
{
   {
         dout<<"\n"<<v.element[i];
```

```
    }

    return dout;

}

int main()

{

    vector v1(3);

    vector v2(3);

    vector v3(3);

    cout<<"\n constructor v1 \n";

    v1.display();

    cout<<"\n constructor v2 \n";

    v2.display();

    cout<<"\n enter the value of v1 object \n";

    cin>>v1;
```

```
cout<<"\n enter the value of v2 object \n";

cin>>v2;

cout<<"\n the value of v1 object : "<<v1;

cout<<"\n the value of v2 object : "<<v2;

cout<<"\n addition : "<<v3;

cout<<"\n subtraction : "<<v3;

cout<<"\n multiplication : "<<v3;

cout<<"\n division : "<<v3;

cout<<"\n scalar : "<<v3;

int index;

cout<<"\n enter the index : ";

cin>>index;

cout<<"\n index : "<<index<<" element : "<<v1[index];

{

     cout<<v2[i]<<"\n";
```

```
   }

   return 0;

}
```

output:-

```
 constructor v1

0

0

0

 constructor v2

0

0

0

 enter the value of v1 object

 enter the value of v2 object
```

```
 the value of v1 object :

10

20

30

 the value of v2 object :

5

4

3

 addition :

15

24

33

 subtraction :

5

16
```

27

multiplication :

50

80

90

division :

2

5

10

scalar :

20

40

60

enter the index :

```
 index : 1 element : 20

5

4

3
```

```
#include<iostream>

using namespace std;

class mystring{

   char *data;

   int length;

   public:

   mystring(char *s1)

   {

         {

         }
```

```
        {

        }

}

{

        return false;

        int i;

        {

                {

                return false;

                }

        }

        return true;

}

{

        int i;
```

```
	{
		if(data[i] < v.data[i])
		{
			return true;
		}
		else if(data[i] > v.data[i])
		{
			return false;
		}
	}
}
char operator[](int index)
{
	{
```

```
            return '\0';

        }

        return data[index];

    }

    void copy(mystring &v)

    {

        delete[] data;

        int i;

        {

        }

    }

    void reverse()

    {

        int i;

        {
```

```
        }
}
mystring operator+(mystring &v)
{
        {
        }
        {
        }
        return mystring(concate);
}
void display()
{
        cout<<"\n string is : "<<data;
}
```

```
};

int main()

{

  mystring s1(" "),s2(" "),s3(" ");

  char temp[100];

  int choice;

  do{

       cout<<"\n 0 : exit "

             "\n 3 : Copy the string to another "

             "\n 4 : Extract a character from the string
(Overload []) "

             "\n 5 : Reverse the string "

             "\n 6 :  Concatenate two strings (+ operator)
";

       cin>>choice;

       switch(choice)
```

```
{
        case 1:{
                cout<<"\n enter the value of s1 : ";
                cin>>temp;
                cout<<"\n enter the value of s2 : ";
                cin>>temp;
                s1.display();
                s2.display();
                {
                        cout<<"\n both string are same";
                }
                else
                {
                        cout<<"\n both are not same";
                }
```

```
        break;

    }

    case 2:{

        cout<<"\n enter the value of s1 : ";

        cin>>temp;

        cout<<"\n enter the value of s2 : ";

        cin>>temp;

        s1.display();

        s2.display();

        {

            cout<<"\n string 2 is greater";

        }

        else

        {
```

```
            cout<<"\n string 1 is greater";

        }

        break;

    }

    case 3:{

        cout<<"\n enter the value of s1 : ";

        cin>>temp;

        s2.copy(s1);

        cout<<"\n the string s1 : ";

        s1.display();

        cout<<"\n the string s2 : ";

        s2.display();

        break;

    }

    case 4:{
```

```
                cout<<"\n enter the value of s1 : ";

                cin>>temp;

                int index;

                cout<<"\n enter the index : ";

                cin>>index;

                cout<<"\n index : "<<index<<" character :
"<<s1[index]<<endl;

                break;

            }

            case 5:{

                cout<<"\n enter the value of s1 : ";

                cin>>temp;

                s1.reverse();

                s1.display();

                break;
```

```
            }

            case 6:{

                cout<<"\n enter the value of s1 : ";

                cin>>temp;

                cout<<"\n enter the value of s2 : ";

                cin>>temp;

                s3.display();

                break;

            }

        }

   return 0;

}
```

output:-

```
enter the value of s1 : hello

enter the value of s2 : hello
```

```
string is : hello

string is : hello

both strings are same

enter the value of s1 : apple

enter the value of s2 : banana

string is : apple

string is : banana

string 2 is greater

enter the value of s1 : test

the string s1 : string is : test

the string s2 : string is : test

enter the value of s1 : example

enter the index : 3

index : 3 character : m
```

enter the value of s1 : hello

string is : olleh

enter the value of s1 : good

enter the value of s2 : morning

string is : goodmorning

(5) Create a date class with the following capabilities :

(a) Output the date in multiple formats such as

DD MM YYYY

MM/DD/YY

May 14, 2001

(b) Use overloaded constructors to create Date objects initialized with date format in section (a)

(c) Overload operators for testing equality of two dates and for comparing dates to determine if one date is prior to, or after, another date.

(d) Create a member function nextday to increment the day by day. The date object function should always remain in consistant state. Be sure to test following cases : (i)

Incrementing to next month (ii) Incrementing into the next year.

```
#include <iostream>

using namespace std;

class Date {

private:

    int day, month, year;

    bool isValidDate(int d, int m, int y) {

        if (m < 1 || m > 12 || d < 1 || y < 0) return false;

        }

    }

public:

    Date(int d, int m, int y) : day(d), month(m), year(y) {

        if (!isValidDate(day, month, year)) {

            cout << "Invalid date! Setting default date (01/01/2000).\n";
```

```
        }

    }

    Date(int d, const char* m, int y) {

                break;

            }

        }

        if (!isValidDate(day, month, year)) {

            cout << "Invalid date! Setting default date
(01/01/2000).\n";

        }

    }

    void displayFormat1() {

        cout << day << " " << month << " " << year << endl;

    }

    void displayFormat2() {
```

```
        cout << month << "/" << day << "/" << (year % 100)
<< endl;

    }

    void displayFormat3() {

        cout << months[month - 1] << " " << day << ", " <<
year << endl;

    }

    }

    bool operator<(const Date& d) {

        return day < d.day;

    }

    bool operator>(const Date& d) {

    }

    void nextDay() {

        }

        day++;
```

```
        if (day > daysInMonth[month - 1]) {
            month++;
            if (month > 12) {
                year++;
            }
        }
    }
};
int main() {
    int d, m, y;
    cout << "Enter date (day month year): ";
    cin >> d >> m >> y;
    Date date1(d, m, y);
    cout << "\nDate in format DD MM YYYY: ";
    date1.displayFormat1();
```

```
    cout << "Date in format MM/DD/YY: ";

    date1.displayFormat2();

    cout << "Date in format Month Day, Year: ";

    date1.displayFormat3();

    cout << "\nDate after incrementing a day:\n";

    date1.nextDay();

    date1.displayFormat1();

    Date date2(14, "May", 2001);

    cout << "\nComparing with another date (14 May
2001):\n";

    date2.displayFormat3();

        cout << "The dates are equal.\n";

    } else if (date1 < date2) {

        cout << "The first date is earlier.\n";

    } else {
```

```
        cout << "The first date is later.\n";

    }

    return 0;

}
```

output:-

```
Enter date (day month year): 28 2 2000

Date in format DD MM YYYY: 28 2 2000

Date in format MM/DD/YY: 2/28/00

Date in format Month Day, Year: February 28, 2000

Date after incrementing a day:

29 2 2000

Comparing with another date (14 May 2001):

May 14, 2001

The first date is earlier.
```

(7) Design a class date which sets date of object to dd, mm ,yyyy format . Overload + and _ operators for the class date to add given no of is set using a constructor to some valid date and to find difference between two given date. Display the original dates , after addition and the difference. The date objects created should be validated in the constructor itself.

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

class Date {

    int day, month, year;

    bool isLeapYear(int y) {

    }

    int daysInMonth(int m, int y) {

        return 31;

    }

    void normalizeDate() {

        while (day > daysInMonth(month, year)) {

            if (++month > 12) {
```

```
                year++;

            }

        }

    }

public:

    Date(int d, int m, int y) : day(d), month(m), year(y) {

        if (day < 1 || day > daysInMonth(month, year) ||
month < 1 || month > 12 || year < 0) {

            cout << "Invalid date!\n";

            exit(1);

        }

    }

    Date operator+(int days) {

        result.normalizeDate();

        return result;

    }
```

```
    friend Date operator+(int days, Date d) {
        return d + days;
    }
    int operator-(Date &d) {
        }
        }
        return abs(dayCount1 - dayCount2);
    }
    void display() {
        cout << day << "/" << month << "/" << year;
    }
};
int main() {
    Date d1(12, 10, 2023), d2(25, 12, 2023);
```

```
    cout << "Original date d1: ";

    d1.display();

    cout << "\nOriginal date d2: ";

    d2.display();

    cout << "\n\nAfter adding " << daysToAdd << " days to
d1: ";

    d3.display();

    cout << "\n\nAfter adding " << daysToAdd << " days
(using days + d1): ";

    d4.display();

    cout << "\n\nDifference between d1 and d2: " <<
difference << " days\n";

    return 0;

}
```

output:-

```
Original date d1: 12/10/2023

Original date d2: 25/12/2023
```

```
After adding 10 days to d1: 22/10/2023

After adding 10 days (using days + d1): 22/10/2023

Difference between d1 and d2: 74 days
```

(8) Define a singly linked list class, which is a made up objects of node class. Provide addition, deletion of nodes, with operator overloading.

```
#include<iostream>

using namespace std;

struct Node{

   int data;

   Node *next;

};

class linkedlist{

   public:

   Node *head;

   void inti()
```

```
{

}

{

      {

      }

      else

      {

            {

            }

      }

}

{

      {

            return ;
```

```
        }

        {

             delete temp;

             return;

        }

        {

        }

        {

             return;

        }

        delete current;

   }

   friend ostream& operator<<(ostream &dout,linkedlist
&list)

   {

        {
```

```
            dout<<"\t"<<temp->data;

        }

        return dout;

    }

};

int main()

{

    linkedlist list;

    list.inti();

    cout<<"\n\t"<<list;

    cout<<"\n\t"<<list;

    cout<<"\n\t"<<list;

    cout<<"\n\t"<<list;

    return 0;
```

```
}
```

output:-

```
1       2       3

2       3

3
```

(10) Define a class of your own choice. Overload all the operators supported by C++, which can be overloaded for your class. List all the operators which cannot be overloaded for your class, with proper justification.

```
#include <iostream>
using namespace std;
class ComplexNumber {
    double real;
    double imaginary;

public:

    ComplexNumber operator+(const ComplexNumber &c) const {
        return ComplexNumber(real + c.real, imaginary +
c.imaginary);
    }
    ComplexNumber operator-(const ComplexNumber &c) const {
        return ComplexNumber(real - c.real, imaginary -
c.imaginary);
```

```
    }

    ComplexNumber operator*(const ComplexNumber &c) const {
        return ComplexNumber(real * c.real - imaginary *
c.imaginary,    real * c.imaginary + imaginary * c.real);
    }

    ComplexNumber operator/(const ComplexNumber &c) const {
        return ComplexNumber((real * c.real + imaginary *
c.imaginary) / denominator,
                             (imaginary * c.real - real *
c.imaginary) / denominator);
    }
    }
    }

    ComplexNumber& operator++() {
        ++real;
        ++imaginary;
        return *this;
    }

    ComplexNumber operator++(int) {
        real++;
        imaginary++;
        return temp;
    }

    void operator()(double r, double i) {
    }
```

```
    void display() const {
        cout << real << " + " << imaginary << "i" << endl;
    }

    friend ostream& operator<<(ostream &out, const
ComplexNumber &c) {
        out << c.real << " + " << c.imaginary << "i";
        return out;
    }

    friend istream& operator>>(istream &in, ComplexNumber
&c) {
        cout << "Enter real part: ";
        in >> c.real;

        cout << "Enter imaginary part: ";
        in >> c.imaginary;

        return in;
    }

};

int main() {
    ComplexNumber c1(3, 4), c2(1, 2), c3;
    cout << "Initial Complex Numbers:" << endl;
    cout << "c1: " << c1 << endl;
    cout << "c2: " << c2 << endl;
```

```
    cout << "\nAddition (c1 + c2): ";

    cout << c3 << endl;

    cout << "Subtraction (c1 - c2): ";

    cout << c3 << endl;

    cout << "Multiplication (c1 * c2): ";

    cout << c3 << endl;

    cout << "Division (c1 / c2): ";

    cout << c3 << endl;

    cout << "\nPrefix Increment (++c1): ";

    ++c1;

    cout << c1 << endl;

    cout << "Postfix Increment (c1++): ";

    c1++;

    cout << c1 << endl;

    cout << "\nFunction call operator to reset values (c1(5,
6)): ";
```

```
    c1(5, 6);

    cout << c1 << endl;

    return 0;

}
```

output:-

```
Enter real part: 3

Enter imaginary part: 4

Enter real part: 1

Enter imaginary part: 2

Initial Complex Numbers:

c1: 3 + 4i

c2: 1 + 2i

Addition (c1 + c2): 4 + 6i

Subtraction (c1 - c2): 2 + 2i
```

Multiplication (c1 * c2): -5 + 10i

Division (c1 / c2): 2.2 + 0.4i

Prefix Increment (++c1): 6 + 7i

Postfix Increment (c1++): 7 + 8i

Function call operator to reset values (c1(5, 6)): 5 + 6i

Type Conversions

(1) Construct a class distance having member variables int feets and int inches. Design the class to make the following possible :

(i) to convert this class into the basic data type int which will represent the total no. of inches of the class.

(ii) to convert a basic data type int to distance class

(iii) to convert distance class to length class having member variables int meters and int centimeters

Write a C++ program to test your class.

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

class Distance{
   int inches;
   int feets;
   public:
   Distance()
   { }
```

```
   Distance(int i,int f)
   {}
   operator int()
   {
        return(feets * 12 + inches);
   }
   int getfeet()
   {
        return feets;
   }

   int getinches()
   {
        return inches;
   }
   void frominches(int sum_marksinches)
   {}

   void display()
   {
        cout<<"\n feet : "<<feets<<"\n inches : "<<inches;
   }
};

class length{
   int meters;
   int centimeters;

   public:
```

```
   length()
   {}

   length(Distance &d)
   {}

   void display()
   {
        cout<<"\n meters : "<<meters<<"\n centimeters :
"<<centimeters;
   }
};

int main()
{
   int i,f;
   cout<<"\n enter the inches : ";
   cin>>i;

   cout<<"\n enter the feets : ";
   cin>>f;

   Distance d(i,f);
   cout<<"\n sum_marks inches : "<<sum_marksinches;
   Distance d1;
   d1.frominches(sum_marksinches);
   d1.display();
   length l;
   l.display();
   return 0;
```

```
}
```

output:-

```
Enter the inches: 8

Enter the feets:

Total inches: 68

Feet: 5

Inches: 8

Meters: 1

Centimeters: 20
```

(2) Define a class Integer containing an int variable and also two more variables, UpperBound and LowerBound. Provide a function Validate to the same class such that when that function is called, it checks to see if the value is between upper and lower bound. Integer class should be defined such that when we write following code,

```
Integer Int1(5);

int int1;

int1 = Int1;
```

Should work properly; i.e. it casts int to Integer object if validated.

```
#include<iostream>

using namespace std;


class Integer{


   int value;

   public:

   Integer()

   {}
```

```
   Integer(int num)
   {}

   int getdata()
   {
        return value;
   }

   operator int()
   {
        {
             return value;
        }
        else
        {
             return 0;
        }
   }
};

int main()
{
   int num;
   cout<<"\n enter the number : ";
   cin>>num;
   Integer Int1(num);
   int int1;
   if(int1)
   {
```

```
        cout<<int1<<" is in range";
   }
   else
   {
        cout<<"\n out of range";
   }
}
```

output:-

```
Enter the number: 7
7 is in range
Enter the number: 12
Out of range
```

(3) Create two classes DM and DB which store the value of distances. DM stores distances in metres and centimeters and DB in feet and inches. Write a program that can read values for the class objects and add one object DM with another object DB. Use a friend function to carry out the addition operation. The object that stores the results may be a DM object or DB object, depending on the units in which the results are required. The display should be in the format of feet and inches or meters and centimeters depending on the object on display.

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

class DB;
class DM{
   int meters;
   int centimeters;
   public:
```

```
   DM()
   {}

   DM(int m,int cm)
   {
        {
             meters++;
        }
   }
   void print()
   {
        cout<<"\n centimeters : "<<centimeters<<"\n meters :
"<<meters;
   }
   friend DM operator+(const DM &,const DB &);
};

class DB{
   int feets;
   int inches;

   public:
   DB(int f,int i)
   {
        {
             feets++;
        }
   }
   DB(int i)
   {
```

```
        {
             feets++;
        }
   }
   void print()
   {
        cout<<"\n inches : "<<inches<<"\n feets : "<<feets;
   }
   friend DM operator+(const DM &,const DB &);
};
DM operator+(const DM &m,const DB &b)
{
   return DM(m.meters,m.centimeters+sum_markscentimeters);
}

int main()
{
   m1.print();
}
```

output:-

centimeters : 85

meters : 7

Assignment-3

Run Time Polymorphism

(1) Create a base class called shape. Use this class to store two double type values that could be used to compute the area of figures. Derive three specific classes calledtriangle, rectangle and circle from the base shape. Add to the base class, a member function get_data( ) to initialize base class data members and another member function display_area( ) to compute and display the area of figures.Make display_area( ) as a virtual function and redefine this function in derived classes to suit their requirements. Using these three classes design a program that will accept dimensions of a triangle or rectangle interactively and store it in one array . After having read all the input display the area of all the figures whose area has been read in the program. Remember the two values given as input will be treated as lengths of two sides in the case of rectangle and as base and height in case of triangle. In case of circle only one value should be accepted which will be taken as the radius and the default value of the next parameter should be 0.

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

class shape{
   protected:
   double dime1;
   double dime2;

   public:
   shape():dime1(0),dime2(0){}
   virtual void get_data()
   {
        cout<<"\n enter the dimension 1 : ";
```

```
        cin>>dime1;

        cout<<"\n enter the dimension 2 : ";
        cin>>dime2;
   }
};

class triangle : public shape{
   public:
   void display_area()
   {
        cout<<"\n area of triangle is : "<<area;
   }
};
class rectangle : public shape{
   public:
   void display_area()
   {
        cout<<"\n area of rectangle is : "<<area;
   }
};
class circle : public shape{
   public:
   void display_area()
   {
        cout<<"\n area of circle is : "<<area;
   }
};
```

```
int main()
{
   shape *sh[5];
   int choice;
   int i;
        cout<<"\n 1 : triangle "
              "\n 2 : rectangle "
              "\n 3 : circle ";
        cin>>choice;
        switch(choice)
        {
             case 1:{
                  sh[i]->get_data();
                  sh[i]->display_area();
                  break;
             }
             case 2:{
                  sh[i]->get_data();
                  sh[i]->display_area();
                  break;
             }

             case 3:{
                  sh[i]->get_data();
                  sh[i]->display_area();
                  break;
             }

             default:{
```

```
                cout<<"\n invalid \n";
                break;
            }
        }
   }
   return 0;
}
```

output:-

```
1 : triangle

2 : rectangle

3 : circle

1
enter the dimension 1 : 5
enter the dimension 2 : 10
area of triangle is : 25
1 : triangle
2 : rectangle
3 : circle

2
enter the dimension 1 : 4
enter the dimension 2 : 8
area of rectangle is : 32
1 : triangle
```

```
2 : rectangle
3 : circle

3
enter the dimension 1 : 7
area of circle is : 153.86
1 : triangle
2 : rectangle
3 : circle

1
enter the dimension 1 : 3
enter the dimension 2 : 6
area of triangle is : 9

1 : triangle

2 : rectangle

3 : circle

2

enter the dimension 1 : 10

enter the dimension 2 : 12

area of rectangle is : 120
```

(3) ABC publishing company markets both book and audio cassette versions of its work. Create a class called publication that stores the title(a string) and price(type float) of a publication. From this class derive two classes : book , which adds a page count (type int); and tape, which adds playing time in minutes (type float). Write a main program that reads both book and tape information in one array. When the user has finished entering data for all books and tapes, displays the resulting data for all the books and tapes entered. Also count no of book and cassette entries in the array using runtime identification feature of C++.

```
#include<iostream>

using namespace std;

class publication{

  protected:

  string title;

  float price;

  public:

};

class book : public publication{

  protected:
```

```
int page_count;

public:

void getdata()

{

     cout<<"\n enter the title               : ";

     cin>>title;

     cout<<"\n enter the price               : ";

     cin>>price;

     cout<<"\n enter the page number of book : ";

     cin>>page_count;

}

void display()

{

     cout<<"\n title      : "<<title;

     cout<<"\n price      : "<<price;
```

```
        cout<<"\n page count : "<<page_count;
    }
};
class tape : public publication{
    protected:
    float playing_time;
    public:
    void getdata()
    {
        cout<<"\n enter the title              : ";
        cin>>title;
        cout<<"\n enter the price              : ";
        cin>>price;
        cout<<"\n enter the playing_time       : ";
```

```
        cin>>playing_time;

    }

    void display()

    {

        cout<<"\n title        : "<<title;

        cout<<"\n price        : "<<price;

        cout<<"\n playing_time : "<<playing_time;

    }

};

int main()

{

    publication *p[5];

    int i;

    int choice;

    {
```

```
cout<<"\n 1 : book "

       "\n 2 : tape ";

cin>>choice;

switch(choice)

{

     case 1:{

          p[i]->getdata();

          p[i]->display();

          book_count++;

          break;

     }

     case 2:{

          p[i]->getdata();

          p[i]->display();
```

```
                    tape_count++;

                    break;

                }

            }

    }

    cout<<"\n sum_marks count of book : "<<book_count;

    cout<<"\n sum_marks count of tape : "<<tape_count;

    return 0;

}
```

output:-

```
1 : book

2 : tape

1

enter the title              : C++ Programming

enter the price              : 300
```

```
enter the page number of book : 350

 title      : C++ Programming

 price      : 300

 page count : 350

1 : book

2 : tape

2

enter the title               : Learn C++

enter the price               : 100

enter the playing_time        : 120.5

 title        : Learn C++

 price        : 100

 playing_time : 120.5

1 : book
```

```
2 : tape

1

enter the title               : Data Structures

enter the price               : 250

enter the page number of book : 400

 title      : Data Structures

 price      : 250

 page count : 400

1 : book

2 : tape

2

enter the title               : Advanced C++

enter the price               : 150

enter the playing_time        : 95.5

 title        : Advanced C++
```

```
 price        : 150

 playing_time : 95.5

1 : book

2 : tape

1

enter the title               : Algorithms

enter the price               : 350

enter the page number of book : 500

 title      : Algorithms

 price      : 350

 page count : 500

 sum_marks count of book : 3

 sum_marks count of tape : 2
```

Templates and Exception Handling

(1) Write a generic function that will sort a character string, integer and float value. Create a menu with appropriate options and accept the values from the user.

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

template <typename s>
void sort(s a[],int size)
{
   s temp;
   {
        {
             if(a[j] > a[j+1])
             {}
        }
   }
}
template <typename s>
void display(s a[],int size)
{
   {
        cout<<a[i]<<"\t";
   }
}

int main()
{
   int a[5];
```

```
float b[5];
string c[5];
int choice;

do{

     cout<<"\n 0 : exit"

            "\n 1 : integer sorting "

            "\n 2 : float sorting "

            "\n 3 : string sorting ";

     cin>>choice;

     switch(choice)

     {

          case 1:{

               cout<<"\n enter the value of integer :
\n";

               {

                    cin>>a[i];

               }
```

```
        sort(a,5);

        cout<<"\n sorted value of integer \n";

        display(a,5);

        break;

    }

    case 2:{

        cout<<"\n enter the value of float : \n";

        {

            cin>>b[i];

        }

        sort(b,5);

        cout<<"\n sorted value of float \n";

        display(b,5);

        break;
```

```
            }

            case 3:{

                cout<<"\n enter the string \n";

                {

                    cin>>c[i];

                }

                sort(c,5);

                cout<<"\n sorted value of character string
\n";

                display(c,5);

                break;

            }

        }

   return 0;

}
```

(2) Implement binary search as a generic function (function template). The function should take arguments as array name, the size and the element to be searched.

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

template <typename s>
int search(s a[],int size,s element)
{
   int mid;
   {
        {
             return 1;
        }
        else if(a[mid] > element)
        {
}
        else
        {
}
   }
   return 0;
}

template <typename s>
void sort(s a[],int size)
{
   int i,j;
```

```
      s temp;
      {
            {
                  if(a[j] > a[j + 1])
                  {

}
            }
      }
   }

   int main()
   {
      int size;
      cout<<"\n enter the size of an array : ";
      cin>>size;
      int a[size];
      int i;
      int choice;
      float b[size];
      char c[size];
            do{
            cout<<"\n 0 : exit"
                  "\n 1 : integer sorting "
                  "\n 2 : float sorting "
                  "\n 3 : string sorting ";
            cin>>choice;
            switch(choice)
            {
```

```
            case 1:{
                 cout<<"\n enter the integer element of an
array \n";
                 {
                      cin>>a[i];
                 }
                 sort(a,5);
                 int element;
                 cout<<"\n enter the element : ";
                 cin>>element;
                 if(search(a,size,element))
                 {
                      cout<<"element found !!!";
                 }
                 else
                 {
                      cout<<"\n not found \n";
                 }
                 break;
            }
            case 2:{
                 cout<<"\n enter the float element of an
array \n";
                 {
                      cin>>b[i];
                 }
                 sort(b,5);
                 float element;
                 cout<<"\n enter the element : ";
                 cin>>element;
```

```
                if(search(b,size,element))
                {
                     cout<<"element found !!!";
                }
                else
                {
                     cout<<"\n not found \n";
                }

                break;

           }

           case 3:{

                cout<<"\n enter the character of an array
\n";

                {

                     cin>>c[i];

                }

                sort(c,5);

                char character;

                cout<<"\n enter the character : ";
```

```
                cin>>character;

                if(search(c,size,character))

                {

                     cout<<"character found !!!";

                }

                else

                {

                     cout<<"\n not found \n";

                }

                break;

            }

        }

   return 0;

}
```

(3) Write a template function called find(). This function searches an array for an object. It returns either the index of the matching object (if one is found) or -1 if no match is found.

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

template<typename f>

int find(f a[] , int size , f element)
{
   int i;
   {
        {
             return 1;
        }
   }
   return 0;
}

int main()
{
   int choice;
   int size;
   cout<<"\n enter the size : ";
   cin>>size;
   int a[size];
   float b[size];
```

```
char c[size];
do{
     cout<<"\n 1 : integer "
            "\n 2 : float "
            "\n 3 : character string ";
     cin>>choice;
     switch(choice)
     {
          case 1:{
               cout<<"\n enter the integer elements \n";
               {
                    cin>>a[i];
               }
               int element;
               cout<<"\n enter the element : ";
```

```
        cin>>element;

        if(find(a , size , element))

        {

             cout<<"\n element found!!!! \n";

        }

        else

        {

             cout<<"\n element not found \n";

        }

        break;

   }

   case 2:{

        cout<<"\n enter the float elements \n";

        {

             cin>>b[i];
```

```
        }

        float element;

        cout<<"\n enter the element : ";

        cin>>element;

        if(find(b , size , element))

        {

             cout<<"\n element found!!!! \n";

        }

        else

        {

             cout<<"\n element not found \n";

        }

        break;

    }
```

```
case 3:{

     cout<<"\n enter the integer elements \n";

     {

          cin>>c[i];

     }

     char character;

     cout<<"\n enter the character : ";

     cin>>character;

     if(find(c , size , character))

     {

          cout<<"\n character found!!!! \n";

     }

     else

     {

          cout<<"\n character not found \n";
```

```
                    }

                    break;

                }

            }

    return 0;

}
```

4) Write a object oriented program to implement a generic Stack. Incorporate all the possible operation on Stack in the program. Rework stack class so that stack overflows are handled as exceptions.

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

template <typename s , int size>

class stack{

    s a[size];

    public:

    void push(int value)
```

```
{
      {
            cout<<" stack overflow ";
      }
      else
      {
            top++;
            cout<<value<<" is inserted\n";
      }
}
void pop()
{
      {
            cout<<" stac underflow ";
```

```
        }

        else

        {

              cout<<a[top]<<" is deleted";

              top--;

        }

}

void peek()

{

        {

              cout<<" stac underflow ";

        }

        else

        {

              cout<<a[top]<<" is top of the stack element";
```

```
      }

}

void isfull()

{

      {

            cout<<" stack is full ";

      }

}

void isempty()

{

      {

            cout<<" stack is empty ";

      }

}
```

```
};

int main()

{

   stack <int,10> st;

   int value;

   int choice;

   do{

        cout<<"\n 0 : exit "

                "\n 1 : push "

                "\n 2 : pop "

                "\n 3 : peek "

                "\n 4 : isfull "

                "\n 5 : isempty ";

        cin>>choice;

        switch(choice)
```

```
{
        case 0:{
                cout<<"\n exiting...\n";
                break;
        }
        case 1:{
                        cout<<"\n enter the value : ";
                        cin>>value;
                        st.push(value);
                        break;
                }
        case 2:{
                st.pop();
                break;
```

```
            }

            case 3:{

                  st.peek();

                  break;

            }

            case 4:{

                  st.isfull();

                  break;

            }

            case 5:{

                  st.isempty();

                  break;

            }

      }

return 0;
```

```
}
```

(5) Write a object oriented program to implement a generic Queue. Incorporate all the possible operation on Queue in the program.

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

template<typename q,int size>

class queue{

   q que[size];

   public:

   int insertion(q value)

   {

        {

             return 0;

        }

        {
```

```
            front++;

      }

      end++;

      return 1;

}

int deletion()

{

      {

            return 0;

      }

      {

            return 1;

      }

      front++;

      return 1;
```

```
    }

    void display()

    {

          {

                cout<<"\n queue is empty \n";

          }

          else

          {

                {

                      cout<<"\t"<<que[i];

                }

          }

    }

};
```

```
int main()

{

   queue<int,10> q;

   int value;

   int choice;

   do{

        cout<<"\n 0 : exit "

               "\n 1 : insertion "

               "\n 2 : deletion "

               "\n 3 : display ";

        cin>>choice;

        switch(choice)

        {

             case 0:{

                  cout<<"\n exiting.........";
```

```
        break;

    }

    case 1:{

        cout<<"\n enter the value : ";

        cin>>value;

        if(q.insertion(value))

        {

            cout<<"\n element inserted";

        }

        else

        {

            cout<<"\n queue is full";

        }

        break;
```

```
}

case 2:{

     if(q.deletion())

     {

          cout<<"\n element deleted ";

     }

     else

     {

          cout<<"\n queue is empty";

     }

     break;

}

case 3:{

     cout<<"\n elements in queue \n";

     q.display();
```

```
                    break;

                }

            }

    return 0;

}
```

(6) Write a program to create template class called "Safearray". Rules for this class are as follows :

(1) Size is equal to 100

(2) Index from 0 to size -1

(3) If array is sought outside bound the program aborts

(4) Function safeput() is used to assign value to an array element.

(5) Function safeget() is used to return the array element.

Make necessary provisions so the program terminates gracefully when unsafe action is attempted.

```
#include<iostream>

#include<stdexcept>

using namespace std;

template<typename s , int size>
```

```
class safearray{

   s array[size];

   public:

   void safeput(int index , s value)

   {

        {

             cout<<"invalid index";

             abort();

        }

   }

   s safeget(int index)

   {

        {

             cout<<"invalid index";

        }
```

```
        return array[index];
    }
};
int main()
{
    safearray<int,100> sf;
    int index,value,choice;
    do{
        cout<<"\n 0 : exit "
            "\n 1 : safeput"
            "\n 2 : safeget";
        cin>>choice;
        switch(choice)
        {
```

```
case 0:{

     cout<<"\n exiting.........\n";

     break;

}

case 1:{

     cout<<"\n enter the index : ";

     cin>>index;

     cout<<"\n enter the value : ";

     cin>>value;

     sf.safeput(index,value);

     cout<<" value is inserted.\n";

     break;

}

case 2:{

     cout<<"\n enter the index : ";
```

```
                    cin>>index;

                    cout<<"value at index : "<<index<<" value
is : "<<returned_value<<endl;

                    break;

                }

        }

   return 0;

}
```

(9) Design a stack class so that it can store pairs of different type of objects on the stack. Demonstrate your solution. Rework stack class so that stack overflows are handled as exceptions.

```
#include<iostream>
#include<stdexcept>

using namespace std;

template <typename t1 , typename t2>

struct Pair{
```

```
    t1 first_type;

    t2 second_type;

};

template <typename t1 , typename t2 , int size>

class stack{

    Pair<t1,t2> a[size];

    public:

    void push(t1 &value1 , t2 &value2)

    {

        {

            throw overflow_error("stack overflow");

        }

        else

        {

            top++;
```

```
            cout<<value1<<" "<<value2<<" inserted"<<endl;

        }

    }

    void pop()

    {

        {

            throw underflow_error("stack underflow");

        }

        else

        {

            cout<<a[top].first_type<<"
"<<a[top].second_type<<" is deleted "<<endl;

            top--;

        }

    }
```

```
    void peek()

    {

          {

                throw underflow_error("stack underflow");

          }

          else

          {

                cout<<a[top].first_type<<"
"<<a[top].second_type<<" is top of the stack";

          }

    }

    void isfull()

    {

          {

                cout<<"stack is full"<<endl;

          }
```

```
      else

      {

            cout<<"stack is not full"<<endl;

      }

}

void isempty()

{

      {

            cout<<"stack is empty"<<endl;

      }

      else

      {

            cout<<"stack is not empty"<<endl;

      }
```

```
    }
};
int main()
{
    stack<int , string , 5> st;
    int choice;
    int value;
    string str;
    do{
         cout<<"\n 0 : exit "
                "\n 1 : push "
                "\n 2 : pop "
                "\n 3 : peek "
                "\n 4 : isfull "
                "\n 5 : isempty ";
```

```
cin>>choice;

switch(choice)

{

     case 0:{

          cout<<" exiting .........";

          break;

     }

     case 1:{

          cout<<"enter the value : ";

          cin>>value;

          cout<<"enter the string : ";

          cin>>str;

          try{

               st.push(value,str);
```

```
        }
        catch(const exception &e)
        {
            cout<<"exception : "<<e.what()<<endl;
        }
        break;
    }
    case 2:{
        try{
            st.pop();
        }
        catch(const exception &e)
        {
            cout<<"exception : "<<e.what()<<endl;
        }
```

```
        break;
    }
    case 3:{
        try{
            st.peek();
        }
        catch(const exception &e)
        {
            cout<<"exception : "<<e.what()<<endl;
        }
        break;
    }
    case 4:{
        st.isfull();
```

```
                    break;

                }

                case 5:{

                    st.isempty();

                    break;

                }

            }

    return 0;

}
```

(11) Use Time class to provide overloaded -. Here the time query is also to be recorded in file. Use C text file to store the query. If a calling function provides expression Time1 – Time2, then operator – function should throw an exception if Time2 is a later time then Time1. Before throwing exception, though, the operator – function should close the file.

```
#include <iostream>

#include <fstream>

class Time {
```

```
private:
    int hours, minutes, seconds;
    FILE *file;
public:
    }
    int toSeconds() const {
        return hours * 3600 + minutes * 60 + seconds;
    }
    Time operator-(const Time &t) {
        if (toSeconds() < t.toSeconds()) {
            fclose(file);
            throw "Time2 is a later time than Time1";
        }
        fclose(file);
        return Time(h, m, s);
```

```
    }

    void display() {

        printf("%02d:%02d:%02d\n", hours, minutes, seconds);

    }

};

int main() {

    Time t1(5, 45, 30), t2(3, 20, 15);

    try {

        result.display();

    } catch (const char *msg) {

        std::cerr << msg << std::endl;

    }

    return 0;

}
```

Binary Search Tree

```
#include<iostream>
using namespace std;
typedef struct Node{
   int data;
   Node *left;
   Node *right;
}node;

typedef struct Stack{
   node *Node;
   Stack *next;
}stack;

void push(node *root,stack **top)
{
   stack *new1=new stack();
   new1->Node=root;
   new1->next=*top;
   *top=new1;
}

node *pop(stack **top)
{
   if(*top == NULL)
   {
        return NULL;
```

```
   }
   stack *temp=*top;
   *top=(*top)->next;
   node *new1=temp->Node;
   delete temp;
   return new1;
}

bool isempty(stack *top)
{
   return top==NULL;
}

void inorderiterative(node *root)
{
   stack *top=NULL;
   node *current=root;
   while(current != NULL || !isempty(top))
   {
        while(current != NULL)
        {
             push(current,&top);
             current=current->left;
        }
        current=pop(&top);
        cout<<"\t"<<current->data;

        current=current->right;
   }
```

```
}

void preorderiterative(node *root)
{
   stack *top=NULL;
   push(root,&top);
   while(!isempty(top))
   {
        node *current=pop(&top);
        cout<<"\t"<<current->data;

        if(current->left)
        {
             push(current->left,&top);
        }
        if(current->right)
        {
             push(current->right,&top);
        }
   }
}

void postorderiterative(node *root)
{
   stack *top1=NULL;
   stack *top2=NULL;
   push(root,&top1);
   while(!isempty(top1))
   {
```

```
        node *current1=pop(&top1);
        push(current1,&top2);
        if(current1->left)
        {
             push(current1->left,&top1);
        }
        if(current1->right)
        {
             push(current1->right,&top1);
        }
   }
   while(!isempty(top2))
   {
        node *current2=pop(&top2);
        cout<<"\t"<<current2->data;
   }
}


node *createnode(int value)
{
   node *new1=new node();
   new1->data=value;
   new1->left=new1->right=NULL;
   return new1;
}

node *insert(node *root,int value)
{
```

```
node *new1=createnode(value);
if(root == NULL)
{
     root = new1;
}
else
{
     node *parent=NULL;
     node *current=root;
     while(current != NULL)
     {
          parent=current;
          if(current->data > value)
          {
               current=current->left;
          }
          else
          {
               current=current->right;
          }
     }
     if(parent->data > value)
     {
          parent->left=new1;
     }
     else
     {
          parent->right=new1;
     }
```

```
   }
   return root;
}

void searching(node *root,int value)
{
   node *current=root;
   while(current != NULL)
   {
        if(current->data == value)
        {
             cout<<"\t"<<"value is found : "<<value;
             return ;
        }
        else if(current->data > value)
        {
             current=current->left;
        }
        else
        {
             current=current->right;
        }
   }
}

node *delelement(node *root,int value)
{
   node *parent=NULL;
   node *current=root;
```

```
while(current != NULL && current->data != value)
{
     parent=current;
     if(current->data > value)
     {
          current=current->left;
     }
     if(current->data < value)
     {
          current=current->right;
     }
}
if(current == NULL)
{
     return root;
}
if(current->left == NULL || current->right == NULL)
{
     node  *newcurr=NULL;
     if(current->left == NULL)
     {
          newcurr=current->right;
     }
     else
     {
          newcurr=current->left;
     }
     if(parent == NULL)
     {
```

```
            return newcurr;
        }
        if(parent->left == current)
        {
            parent->left=newcurr;
        }
        else
        {
            parent->right=newcurr;
        }
        delete current;
    }
    else
    {
        node *p=NULL;
        node *temp=current->right;
        while(temp->left != NULL)
        {
            p=temp;
            temp=temp->left;
        }
        if(p != NULL)
        {
            p->left=temp->right;
        }
        else
        {
            current->right=temp->right;
        }
```

```
        current->data=temp->data;
        delete temp;
   }
   return root;
}

void inorder(node *root)
{
   if(root != NULL)
   {
        inorder(root->left);
        cout<<"\t"<<root->data;
        inorder(root->right);
   }
}

void preorder(node *root)
{
   if(root != NULL)
   {
        cout<<"\t"<<root->data;
        preorder(root->left);
        preorder(root->right);
   }
}

void postorder(node *root)
{
   if(root != NULL)
```

```
   {
        postorder(root->left);
        postorder(root->right);
        cout<<"\t"<<root->data;
   }
}

int main()
{
   int choice;
   int value;
   node *root=NULL;
   do{
        cout<<"\n 0 : exit "
              "\n 1 : insert "
              "\n 2 : inorder "
              "\n 3 : preorder "
              "\n 4 : postorder "
              "\n 5 : searching "
              "\n 6 : delete "
              "\n 7 : inorder interative "
              "\n 8 : preorderiterative "
              "\n 9 : postorderiterative";

   cout<<"\n================================================
=======\n";
        cin>>choice;

   cout<<"\n================================================
=======\n";
        switch(choice)
```

```
        {
                case 0:{
                        cout<<"\n exit \n";
                        break;
                }
                case 1:{
                        cout<<" \n enter the value : ";
                        cin>>value;

   cout<<"\n=============================================
=======\n";
                        root=insert(root,value);
                        break;
                }
                case 2:{
                        inorder(root);
                        break;
                }
                case 3:{
                        preorder(root);
                        break;
                }
                case 4:{
                        postorder(root);
                        break;
                }
                case 5:{
                        cout<<" \n enter the value : ";
                        cin>>value;
```

```
   cout<<"\n================================================
=======\n";
                searching(root,value);
                break;
            }
            case 6:{
                cout<<" \n enter the value : ";
                cin>>value;

   cout<<"\n================================================
=======\n";
                root=delelement(root,value);
                break;
            }
            case 7:{
                inorderiterative(root);
                break;
            }
            case 8:{
                preorderiterative(root);
                break;
            }
            case 9:{
                postorderiterative(root);
                break;
            }
            default:{
                cout<<"\n invalid \n";
                break;
            }
```

```
        }
   }while(choice != 0);
   return 0;
}
Output:
0 : exit
 1 : insert
 2 : inorder
 3 : preorder
 4 : postorder
 5 : searching
 6 : delete
 7 : inorder interative
 8 : preorderiterative
 9 : postorderiterative
=======================================================
1

=======================================================

 enter the value : 50

=======================================================

 0 : exit
 1 : insert
 2 : inorder
 3 : preorder
 4 : postorder
```

```
 5 : searching
 6 : delete
 7 : inorder interative
 8 : preorderiterative
 9 : postorderiterative
=========================================================
1

=========================================================

 enter the value : 30

=========================================================

 0 : exit
 1 : insert
 2 : inorder
 3 : preorder
 4 : postorder
 5 : searching
 6 : delete
 7 : inorder interative
 8 : preorderiterative
 9 : postorderiterative
=========================================================
1

=========================================================
```

```
 enter the value : 70

==========================================================

 0 : exit
 1 : insert
 2 : inorder
 3 : preorder
 4 : postorder
 5 : searching
 6 : delete
 7 : inorder interative
 8 : preorderiterative
 9 : postorderiterative
==========================================================
1

==========================================================

 enter the value : 20

==========================================================

 0 : exit
 1 : insert
 2 : inorder
 3 : preorder
 4 : postorder
 5 : searching
```

```
 6 : delete
 7 : inorder interative
 8 : preorderiterative
 9 : postorderiterative
=========================================================
1

=========================================================

 enter the value : 40

=========================================================

 0 : exit
 1 : insert
 2 : inorder
 3 : preorder
 4 : postorder
 5 : searching
 6 : delete
 7 : inorder interative
 8 : preorderiterative
 9 : postorderiterative
=========================================================
1

=========================================================

 enter the value : 60
```

```
==========================================================

 0 : exit
 1 : insert
 2 : inorder
 3 : preorder
 4 : postorder
 5 : searching
 6 : delete
 7 : inorder interative
 8 : preorderiterative
 9 : postorderiterative
==========================================================
1

==========================================================

 enter the value : 80

==========================================================

 0 : exit
 1 : insert
 2 : inorder
 3 : preorder
 4 : postorder
 5 : searching
 6 : delete
```

```
 7 : inorder interative
 8 : preorderiterative
 9 : postorderiterative
=======================================================
2


=======================================================
   20   30   40   50   60   70   80
 0 : exit
 1 : insert
 2 : inorder
 3 : preorder
 4 : postorder
 5 : searching
 6 : delete
 7 : inorder interative
 8 : preorderiterative
 9 : postorderiterative
=======================================================
3


=======================================================
   50   30   20   40   70   60   80
 0 : exit
 1 : insert
 2 : inorder
 3 : preorder
 4 : postorder
 5 : searching
```

```
 6 : delete
 7 : inorder interative
 8 : preorderiterative
 9 : postorderiterative
=======================================================
4

=======================================================
  20   40   30   60   80   70   50
 0 : exit
 1 : insert
 2 : inorder
 3 : preorder
 4 : postorder
 5 : searching
 6 : delete
 7 : inorder interative
 8 : preorderiterative
 9 : postorderiterative
=======================================================
5

=======================================================

 enter the value : 60

=======================================================
  value is found : 60
 0 : exit
```

```
 1 : insert
 2 : inorder
 3 : preorder
 4 : postorder
 5 : searching
 6 : delete
 7 : inorder interative
 8 : preorderiterative
 9 : postorderiterative
=========================================================
6

=========================================================

 enter the value : 70

=========================================================

 0 : exit
 1 : insert
 2 : inorder
 3 : preorder
 4 : postorder
 5 : searching
 6 : delete
 7 : inorder interative
 8 : preorderiterative
 9 : postorderiterative
=========================================================
```

```
2

=======================================================
  20   30   40   50   60   80
 0 : exit
 1 : insert
 2 : inorder
 3 : preorder
 4 : postorder
 5 : searching
 6 : delete
 7 : inorder interative
 8 : preorderiterative
 9 : postorderiterative
=======================================================
0

=======================================================

 Exit

Graph BFS
#include<iostream>
#define max 4

using namespace std;

void bredth_first_search(int adj[][max], int visited[max],
int start)
{
```

```
int queue[max], i, j;
int front = -1, end = -1;

queue[++end]=start;

visited[start]=1;

while(end != front)
{
     start=queue[++front];

     if(start == 4)
     {
          cout<<"5\t";
     }

     else
     {
          cout<<char(65+start)<<"\t";
     }

     for(i=0; i<max; i++)
     {
          if(adj[start][i] == 1 && visited[i] ==  0)
          {
               queue[++end]=i;
               visited[i]=1;
          }
     }
```

```
   }
}

int main()
{
   int adj[max][max], i, j;
   int visited[max]={0};

   cout<<endl<<"Enter the adjacency matrix:"<<endl;

   for(i=0;i<max;i++)
   {
        cout<<endl<<"Row "<<i+1<<endl;

        for(j=0; j<max; j++)
        {
             cin>>adj[i][j];
        }
   }

   bredth_first_search(adj,visited,0);
}
```

Output:

```
Enter the adjacency matrix:

Row 1
0 1 1 0

Row 2
```

```
0 0 0 1

Row 3
0 0 0 1

Row 4
0 0 0 0
A B    C    D
```

GRAPH DFS

```
#include<iostream>
#define max 4

using namespace std;

void depth_first_search(int adj[][max], int visited[max],
int start)
{
    int stack[max], top = -1, i;

    // Push the start node onto the stack
    stack[++top] = start;
    visited[start] = 1;

    cout<<char(65+start)<<"\t";

    while(top!=-1)
   {
        // Peek the top node of the stack
```

```
        start = stack[top];

        // Check if there are any unvisited adjacent nodes
        for(i=0; i<max; i++)
        {
            if(adj[start][i]==1 && visited[i]==0)
             {
                stack[++top]=i; // Push the adjacent node onto the stack
                visited[i]=1;   // Mark it as visited

                 cout<<char(65+i)<<"\t"; // Print the visited node
                break;
            }
        }

        // If no adjacent unvisited nodes were found, pop the stack
        if(i==max)
        {
            top--;
        }
    }
}

int main()
{
    int adj[max][max];
    int visited[max] = {0};
```

```
    int i, j;

    // Reading the adjacency matrix
    cout<<endl<<"Enter the adjacency matrix:"<< endl;

    for(i=0; i<max; i++)
   {
        cout<<endl<<"Row"<<i+1<<endl;

        for(j=0; j<max; j++)
        {
            cin>>adj[i][j];
        }
    }

    // Perform DFS starting from node 0
    cout << "DFS Traversal:" << endl;

   depth_first_search(adj, visited, 0);
}
```

Output:

Enter the adjacency matrix:

Row1

0 1 1 0

Row2

0 0 0 1

```
Row3
0  0  0  1

Row4
0  0  0  0
DFS Traversal:
A  B    D    C
```

Hashing Collision Resolution Technique

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

struct node
{
   int data;
   struct node *next;
};

typedef struct node node;

void insert(node arr[], int size, int key, int pos)
{
   node *current=new node();
   node *temp=NULL;
   int i;

   current->data=key;
```

```
   current->next=NULL;

   if(arr[pos].next==NULL)
   {
        arr[pos].next=current;
   }

   else
   {
        temp=arr[pos].next;

        while(temp->next!=NULL)
        {
             temp=temp->next;
        }

        temp->next=current;
   }
}

void display(node arr[], int size)
{
   node *temp=NULL;
   int i;

   for(i=0; i<size; i++)
   {
        cout<<endl<<i<<" :\t\t";
```

```
        if(arr[i].next==NULL)
        {
              cout<<"Empty";
        }

        else
        {
              temp=arr[i].next;

              while(temp!=NULL)
              {
                    cout<<temp->data<<"\t";
                    temp=temp->next;
              }
        }
   }
}

main()
{
   node arr[9];
   int no, i, key, pos;

   for(i=0; i<9; i++)
   {
        arr[i].data=i;
        arr[i].next=NULL;
   }
```

```
   cout<<endl<<"Enter number of data points:\t";
   cin>>no;

   for(i=0; i<no; i++)
   {
        cout<<endl<<"Enter data:\t";
        cin>>key;

        pos=key%9;

        insert(arr, 9, key, pos);
   }

   display(arr, 9);
}
```

Output:

```
Enter number of data points:    5

Enter data:  5

Enter data:  14

Enter data:  23

Enter data:  32

Enter data:  41

0 :          Empty
```

```
1 :          Empty
2 :          Empty
3 :          Empty
4 :          Empty
5 :          5    14   23   32   41
6 :          Empty
7 :          Empty
8 :          Empty
```