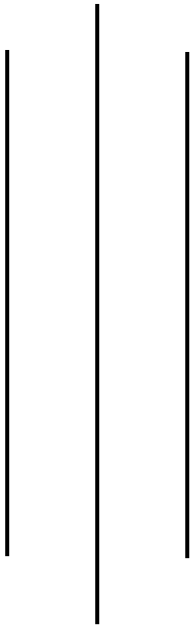
Tribhuvan University

Institute of Science and Technology



LAB SHEET # 3

Submitted by:-

Name:- Sagar Rana Magar

Roll no.:- 023/075

Subject:- Computational Geometry

Year:- 1st Year, 2nd Semester

Submitted to:-

Jagdish Bhatta

Submission date:-……..

PROGRAM

1. To check if polygon is convex or not.
2. For Point Inclusion Test in convex polygon.
3. For Implementation of Ray Casting.

```
#include<iostream>
using namespace std;
class Point{
   public: int x_cor,y_cor;
   void enterPointCoordinate(){
      cout<<"\t\tEnter the X-coordinate: ";
      cin>>x_cor;
      cout<<"\t\tEnter the Y-coordinate: ";
      cin>>y_cor;
   }
   template <class A, class B>
   void setPointCoordinate(A& a, B& b){
       this->x_cor = a;
       this->y_cor = b;
   }
   void displayPoint(){
       cout<<"("<<x_cor<<" ,"<<y_cor<<")";
   }
};

class Line{
   public: Point point1, point2;
   template <class A, class B>
   void line(Point& a, Point& b){
       this->point1 = a;
       this->point2 = b;
       cout<<"\tFor Starting Point: "<<endl;
       point1.enterPointCoordinate();
```

```
        cout<<"\tFor End Point: "<<endl;
        point2.enterPointCoordinate();
    }
    template <class A, class B>
    void setLinePoints(A& a, B& b){
        this->point1 = a;
        this->point2 = b;
        point1.setPointCoordinate(a.x_cor,a.y_cor);
        point2.setPointCoordinate(b.x_cor,b.y_cor);
    }
};

class TurnTest{
    public: int flag=0;
    template <class A, class B, class C>
    int turnTest(A& a, B& b, C& c){
        double area = 0.5*(a.x_cor*(b.y_cor-c.y_cor)+b.x_cor*(c.y_cor-a.y_cor) +
                           c.x_cor*(a.y_cor-b.y_cor));
        if(area<0){
            flag++;
            cout<<"\n\t\t The point lies to RIGHT";
        }
        return flag;
    }
};

class LineIntersection{
    public: int flag=0;
            Line line1, line2;
    template<class A, class B>
    int checkIntersection(A& a, B& b){
        this->line1 = a;
        this->line2 =b;
        Point p1,p2,p3,p4;
```

```
        double p123, p124, p341, p342;
        p1 = line1.point1;
        p2 = line1.point2;
        p3 = line2.point1;
        p4 = line2.point2;
        p123 = computeArea(p1,p2,p3);
        p124 = computeArea(p1,p2,p4);
        p341 = computeArea(p3,p4,p1);
        p342 = computeArea(p3,p4,p2);
        //Check for intersection
        if (((p123 > 0 && p124 < 0) && (p341 > 0 && p342 < 0))|| ((p123 > 0 && p124 < 0)
       && (p341 < 0 && p342 > 0)) || ((p123 < 0 && p124 > 0) && (p341 < 0 && p342 > 0))||
       ((p123 < 0 && p124 > 0) && (p341 > 0 && p342 < 0)))
            flag++;
        return flag;
    }
    template<class A, class B, class C>
    double computeArea(A& a, B& b, C& c){
        return 0.5*(a.x_cor*(b.y_cor-c.y_cor)+b.x_cor*(c.y_cor-a.y_cor)+c.x_cor*(a.y_cor-
                b.y_cor));
    }
};

struct vertex {
    Point info;
    struct vertex *n;
    struct vertex *p;
}*start, *last;

class polygon{
    public:polygon() {
        start = NULL;
        last = NULL;
    }
```

```
vertex *create_vertex(Point p){
    count++;
    struct vertex *t;
    t = new(struct vertex);
    t->info = p;
    t->n = NULL;
    t->p = NULL;
    return t;
};
void insert(){
    Point p;
    p.enterPointCoordinate();
    struct vertex *t;
    t = create_vertex(p);
    if (start == last && start == NULL) {
        start = last = t;
        start->n= last->n = NULL;
        start->p = last->p= NULL;
    } else {
        last->n= t;
        t->p= last;
        last = t;
        start->p = last;
        last->n= start;
    }
};
int checkConvex(){
    int i, flag;
    TurnTest t1;
    struct vertex *s, *p, *n;
    s = start;
    n = s->n;
    p = n->n;
    for (i = 0;i < count;i++) {
```

```
        flag = t1.turnTest(s->info, n->info, p->info);
        p = p->n;
        s = s->n;
        n = n->n;
    }
    return flag;
}
template <class A>
int checkConvex(A& a){
    int i, flag;
    Point point1;
    point1.x_cor = a.x_cor;
    point1.y_cor = a.y_cor;
    TurnTest t1;
    struct vertex *s, *p, *n;
    s = start;
    n = s->n;
    p = n->n;
    for (i = 0;i < count;i++) {
        flag = t1.turnTest(point1, s->info, n->info);
        p = p->n;
        s = s->n;
        n = n->n;
    }
    return flag;
}
template<class A>
int rayCastingValue(A& a){
    LineIntersection intersection;
    Line edge1, edge2;
    Point point111;
    point111.x_cor = 1024;
    point111.y_cor = a.y_cor;
    int flag=0;
```

```
        struct vertex *s, *p, *n;
        s = start;
        n = s->n;
        p = n->n;
        edge1.setLinePoints(a, point111);
        for(int i = 0;i<count;i++){
            edge2.setLinePoints(p->info, n->info);
            flag = intersection.checkIntersection(edge1,edge2);
            p = p->n;
            s = s->n;
            n = n->n;
        }
        return flag;
    }
    void display(){
        int i;
        struct vertex *s;
        s = start;
        cout<<"\n\t The vertices  of given polygon are: ";
        for (i = 0;i < count-1;i++) {
            s->info.displayPoint();
            cout<<",";
            s = s->n;
        }
        s->info.displayPoint();
    };
};

void enterPolygonDetail(){
    int vertexNo;
    polygon pol;
    cout<<"\n\t Enter the number of vertex of a polygon: ";
    cin>>vertexNo;
    }
```

```
    cout<<endl<<"Enter the vertices of the polygon: "<<endl;
    for(int i=0;i<vertexNo;i++){
        cout<<"\tVertex V"<<i<<": "<<endl;
        pol.insert();
    }
};
int main() {
    int choice, convexFlag, isPolygon;
    polygon pol;
    cout<<"\t\t\t   1. To check if polygon is convex or not."<<endl;
    cout<<"\t\t\t   2. For Point Inclusion Test in convex polygon."<<endl;
    cout<<"\t\t\t   3. For Implementation of Ray Casting."<<endl;
    cout<<"\n\t\t Enter the choice(1/2/3): ";
    cin>>choice;
    switch(choice) {
        case 1:enterPolygonDetail();
            pol.display();
            convexFlag = pol.checkConvex();
            if(convexFlag > 0)
                cout<<"\n\t The polygon is non-convex.";
            else
                cout<<"\n\t The polygon is convex.";
        break;
        case 2: Point p1;
            enterPolygonDetail();
            pol.display();
            cout<<"\n\t Enter the point for which point inclusion is to be tested: "<<endl;
            p1.enterPointCoordinate();
            convexFlag = pol.checkConvex(p1);
            if(convexFlag > 0)
                cout<<"\n\t\t The query point lies outside the polygon.";
            else
                cout<<"\n\t\t The query point lies inside the polygon.";
        break;
```

```
        case 3: Point p31;
            enterPolygonDetail();
            pol.display();
            cout<<"\n\n\t Enter the point for which point inclusion is to be tested: "<<endl;
            p31.enterPointCoordinate();
            if(( pol.rayCastingValue(p31) % 2) == 0)
                cout<<"\n\t\t The query point lies outside the polygon.";
            else
                cout<<"\n\t\t The query point lies inside the polygon.";
            }
        break;
        default:cout<<"Invalid choice.\n\tEnter the correct choice number(1/2/3): ";
    }
    return 0;
}
```