```
#include <iostream>
#include "Point.h"
#include "Polygone.h"

int
main()
{
  // Cree un polygone et le remplit
  cout << "==> Definition p" << endl;
  Polygone p;
  p.ajoutSommet( Point( 1.0, 0.0 ) );
  p.ajoutSommet( Point( 0.5, 0.8 ) );
  p.ajoutSommet( Point( -0.2, 0.6 ) );

  // Affiche un de ses points
  // ? Grace a quelle methode ?
  cout << "A l'indice 2 : " << p.getPoint(2) << endl;

  // Un autre polygone : création par copie
  cout << "==> Definition p2" << endl;
  Polygone p2 = p;

  // un autre polygone : création par défaut puis affectation
  cout << "==> Definition p3" << endl;
  Polygone p3;
  p3 = p;

  // Affiche le polygone et son translate
  // ? Grace a quelle methode ?
  cout << "p3 Orig  :" << endl << p3 << endl;
  p3.deplace( 0.3, 0.2 );
  cout << "p3 Deplace :" << endl << p3 << endl;

  // Affiche le polygone p de départ
  cout << "p Debut  :" << endl << p << endl;

  return 0;
}

/*
==> Definition p
Point::Constructeur par defaut
Point::Constructeur par defaut
Point::Constructeur par defaut
Point::Constructeur par defaut
Point::Constructeur par defaut
Polygone::Constructeur par defaut
Point::Constructeur : 1 , 0
Point::Operateur affectation
Point::Destructeur
Point::Constructeur : 0.5 , 0.8
Point::Operateur affectation
Point::Destructeur
Point::Constructeur : -0.2 , 0.6
Point::Operateur affectation
Point::Destructeur
A l'indice 2 : (-0.2,0.6)
==> Definition p2
Point::Constructeur par defaut
Point::Constructeur par defaut
Point::Constructeur par defaut
Point::Constructeur par defaut
Point::Constructeur par defaut
Polygone::Constructeur par copie
Point::Operateur affectation
Point::Operateur affectation
Point::Operateur affectation
```



```
==> Definition p3
Point::Constructeur par defaut
Point::Constructeur par defaut
Point::Constructeur par defaut
Point::Constructeur par defaut
Point::Constructeur par defaut
Polygone::Constructeur par defaut
Polygone::Operateur affectation
Point::Operateur affectation
Point::Operateur affectation
Point::Operateur affectation
p3 Orig    :
(1,0)
(0.5,0.8)
(-0.2,0.6)

p3 Deplace :
(1.3,0.2)
(0.8,1)
(0.1,0.8)

p  Debut   :
(1,0)
(0.5,0.8)
(-0.2,0.6)

Polygone::Destructeur
Point::Destructeur
Point::Destructeur
Point::Destructeur
Point::Destructeur
Point::Destructeur
Polygone::Destructeur
Point::Destructeur
Point::Destructeur
Point::Destructeur
Point::Destructeur
Point::Destructeur
Polygone::Destructeur
Point::Destructeur
Point::Destructeur
Point::Destructeur
Point::Destructeur
Point::Destructeur
*/
```

06 fév 08 15:43 **Polygone.h** Page 1/1

```
// Fichier Polygone.h

#ifndef _POLYGONE_
#define _POLYGONE_

#include "Point.h"

#include <sstream>
#include <string>

using namespace std;

class Polygone {
private:
  static const int MAX=5;
  Point my_tab[MAX];
  int my_taille;
public:
  Polygone();
  ~Polygone();
  Polygone( const Polygone & poly );
  Polygone & operator=( const Polygone & poly );
  void saisie();
  string toString() const;
  void deplace ( float dep_x, float dep_y );
  void ajoutSommet( const Point & p );
  float perimetre() const;
  void litFichier( string nom_fic );
  void ecritFichier( string nom_fic ) const;
  int taille() const;
  // Point getPoint( int ind ) const;
  // ou
  const Point & getPoint( int ind ) const;
};

ostream& operator<<(ostream& out, const Polygone& p);
#endif
```

06 fév 08 15:43 **Polygone.cc** Page 1/3

```
// Fichier Polygone.c

#include <iostream>
#include <sstream>
#include <fstream>
#include <string>
#include <cassert>
#include "Polygone.h"
#include "Point.h"

using namespace std;

ostream& operator<<(ostream& out, const Polygone& p){
  out << p.toString();
  return out;
}

Polygone::Polygone()
{
  cout << "Polygone::Constructeur par defaut" << endl;
  my_taille = 0;
}

Polygone::~Polygone()
{
  cout << "Polygone::Destructeur" << endl;
}

Polygone::Polygone( const Polygone & poly )
{
  cout << "Polygone::Constructeur par copie" << endl;
  my_taille = poly.my_taille;
  for(int i=0; i<my_taille; i++)
    my_tab[i] = poly.my_tab[i];
}

Polygone &
Polygone::operator=( const Polygone & poly )
{
  cout << "Polygone::Operateur affectation" << endl;
  if( this != &poly )
  {
    my_taille = poly.my_taille;
    for(int i=0; i<my_taille; i++)
      my_tab[i] = poly.my_tab[i];
  }
  return *this;
}

void
Polygone::saisie()
{
  float val;
  do {
    cout << "nb de sommets : ";
    cin >> my_taille;
  } while (my_taille <0 || my_taille > MAX);
  for (int i=0; i<my_taille; i++) {
    cout << i << " : abs ? ";
    cin >> val;
    my_tab[i].setX(val);
    cout << i << " : ord ? ";
    cin >> val;
    my_tab[i].setY(val);
  }
}
```

```
string
Polygone::toString() const
{
  string p_s = "";
  for (int i=0; i<my_taille; i++)
      p_s += my_tab[i].toString() + "\n";
  return p_s;
}

void
Polygone::deplace( float dep_x, float dep_y )
{
  for (int i=0; i<my_taille; i++)
    my_tab[i].deplace( dep_x, dep_y );
}

void
Polygone::ajoutSommet (const Point & p)
{
  // A l'execution, message d'erreur si condition non respectee
  assert(my_taille < MAX);

  my_tab[my_taille] = p;
  my_taille ++;
}

float
Polygone::perimetre() const
{
  float res = 0 ;
  if (my_taille > 1) {
    for (int i=0; i<my_taille-1; i++)
      res = res + my_tab[i].distance(my_tab[i+1]) ;
    res = res + my_tab[0].distance(my_tab[my_taille-1]) ;
  }
  return res ;
}


// on suppose que le fichier ne contient pas plus de Point que MAX
void
Polygone::litFichier( string nom_fic )
{
  fstream f;
  Point p;

  f.open( nom_fic.data(), ios::in );
  if ( f.fail() ) {
    cerr << "Pb ouverture fichier en lecture" << endl;
    exit( -1 );
  }

  my_taille = 0;  // polygone vide au depart
  p.litFlux( f );
  while ( !f.eof() ) {
    // PEUT-ETRE TESTER S'IL N'Y EST PAS DEJA...
    // ==> rajout methode bool contient( const Point & p )
    ajoutSommet( p );
    p.litFlux( f );
  }

  f.close();
}

void
Polygone::ecritFichier( string nom_fic ) const
{
```



```
  fstream f;

  f.open( nom_fic.data(), ios::out );
  if ( f.fail() ) {
    cerr << "Pb ouverture fichier en ecriture" << endl;
    exit( -1 );
  }

  for (int i=0; i<my_taille; i++)
      my_tab[i].ecritFlux(f);

  f.close();
}


int
Polygone::taille() const
{
  return my_taille;
}


// Point
// ou
const Point &
Polygone::getPoint( int ind ) const
{
  // A l'execution, message d'erreur si condition non respectee
  assert( ( ind >= 0 ) && ( ind < my_taille ) );

  return my_tab[ ind ];
}
```

```
// Fichier Point.h

#ifndef __POINT__
#define __POINT__

#include <fstream>
#include <sstream>
#include <string>

using namespace std;

class Point {
  private :
    float my_abs, my_ord;
    static const float EPSILON;
  public :
    Point( float x, float y );                  // constructeur
    Point();                                    // constructeur par defaut
    ~Point();                                   // destructeur
    Point( const Point & p );                   // constructeur par copie
    Point & operator=( const Point & p );       // operateur d'affectation
    string toString() const;
    float getX() const;
    float getY() const;
    void setX( float new_x );
    void setY( float new_y );
    // quelques services
    void deplace( float d_x, float d_y );
    float distance() const;
    float distance( const Point & p ) const;
    bool operator==( const Point & p ) const;
    bool operator<( const Point & p) const;
    void litFlux(fstream & f);
    void ecritFlux(fstream & f) const;

};

ostream& operator<<(ostream& out, const Point& p);
#endif
```



```
// Fichier Point.cc

#include <iostream>
#include <sstream>
#include <string>
#include <cmath>
#include "Point.h"

using namespace std;

ostream& operator<<(ostream& out, const Point& p) {
  out << p.toString();
  return out;
}

const float Point::EPSILON=0.00000001; // calcul

Point::Point( float x, float y ) {
  cout << "Point::Constructeur : " << x << " , " << y << endl;
  my_abs = x;
  my_ord = y;
}

Point::Point() {
  cout << "Point::Constructeur par defaut" << endl;
  my_abs = 0;
  my_ord = 0;
}

Point::~Point() {
  cout << "Point::Destructeur" << endl;
}

Point::Point( const Point & p ) {
  cout << "Point::Constructeur par copie" << endl;
  my_abs = p.my_abs;
  my_ord = p.my_ord;
}

Point &
Point::operator=( const Point & p ) {
  cout << "Point::Operateur affectation" << endl;
  if ( this != &p ) {
    my_abs = p.my_abs;
    my_ord = p.my_ord;
  }
  return *this;
}

string
Point::toString() const
{
  ostringstream ostr;

  ostr << "(" << my_abs << "," << my_ord << ")";
  return ostr.str();
}

float
Point::getX() const {
  return my_abs;
}

float
Point::getY() const {
  return my_ord;
}
```

```
void
Point::setX( float new_x ) {
  my_abs = new_x;
}

void
Point::setY( float new_y ) {
  my_ord = new_y;
}

void
Point::deplace( float d_x, float d_y ) {
  my_abs += d_x;
  my_ord += d_y;
}

float
Point::distance() const {
  // distance à l'origine
  return sqrt(my_abs*my_abs + my_ord*my_ord);
}

float
Point::distance( const Point & p ) const {
  float d_x = p.my_abs-my_abs;
  float d_y = p.my_ord-my_ord;
  return sqrt(d_x*d_x + d_y*d_y);
}

bool
Point::operator==( const Point & p ) const {
   return distance(p) < EPSILON ;
}

bool
Point::operator<( const Point & p) const {
  // si points egaux : retourner faux
  if ( *this == p )
    return false ;
  // points pas egaux
  if ( my_abs < p.my_abs )
    return true ;
  if ( my_abs > p.my_abs )
    return false ;
  // abscisses egales
  return  my_ord < p.my_ord ;
}

void Point::litFlux(fstream & f) {
  f >> my_abs >> my_ord;
}

void Point::ecritFlux(fstream& f) const {
  f << my_abs << " " << my_ord << endl;
}
```