```
// Fichier Polygone.h

#ifndef _POLYGONE_
#define _POLYGONE_

#include "Point.h"

#include <sstream>
#include <string>

using namespace std;

// Modifié seulement partie private
class Polygone {
private:
  Point * my_tab; // tableau qui sera alloué dynamiquement
  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
```

```
// 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;
}

// MODIFIEE
Polygone::Polygone()
{
  cout << "Polygone::Constructeur par defaut" << endl;
  // polygone vide
  my_taille = 0;
  my_tab = NULL;
}

// MODIFIEE
Polygone::~Polygone()
{
  cout << "Polygone::Destructeur" << endl;
  if( my_tab!=NULL )  // Attention, si my_tab était vide, rien à libérer
    delete [] my_tab;
}

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

// MODIFIEE
Polygone &
Polygone::operator=( const Polygone & poly )
{
  cout << "Polygone::Operateur affectation" << endl;
  if( this != &poly )
  {
    if( my_tab!=NULL ) // Attention, si my_tab était vide, rien à libérer
      delete [] my_tab;

    my_taille = poly.my_taille;
    my_tab = new Point[my_taille];
    for(int i=0; i<my_taille; i++)
      my_tab[i] = poly.my_tab[i];
  }
  return *this;
}

// MODIFIEE
void
Polygone::saisie()
{
  float valx, valy;
  int taille_reelle;
  my_taille=0;
```



```
  do {
    cout << "nb de sommets : ";
    cin >> taille_reelle;
  } while (taille_reelle <0);
  for (int i=0; i<taille_reelle; i++) {
    cout << i << " : abs ? ";
    cin >> valx;
    cout << i << " : ord ? ";
    cin >> valy;
    ajoutSommet( Point(valx,valy) );
    //incrementationd emy_taille dans
    //ajoutSommet
  }
}

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 );
}

// MODIFIEE
void
Polygone::ajoutSommet (const Point & p)
{
  my_taille++;
  Point * tmp = new Point[my_taille];
  for(int i=0; i<my_taille-1; i++)
    tmp[i] = my_tab[i];
  tmp[my_taille-1] = p;

  if( my_tab!=NULL ) // Attention, si my_tab était vide, rien à libérer
    delete [] my_tab;

  my_tab = tmp;
}

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 ;
}


// MODIFIEE
// Le fichier peut contenir autant de Point qu'on veut maintenant
// On vide le polygone si besoin
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 );
  }

  // polygone vide au depart
  my_taille = 0;
  //if( my_tab!=NULL )
  //  delete [] my_tab;
  // Pas nécessaire : ajoutSommet le réalise

  // lecture
  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 ];
}
```