Examen

12 janvier 2017 — Durée 2h

Document autorisé : **Mémento C** vierge de toute annotation

Les deux parties sont indépendantes et peuvent être traitées dans un ordre quelconque.

Partie I (9 pt)

Soit le programme C suivant :

```
1  #include <stdlib.h>
2  #include <stdio.h>
3
4  #define N 16
5  #define MARQUE_FIN -1
6
7  int main(int argc, char ** argv) {
8    FILE * f_entree;
9    int ec,i;
10   int compte[N];
11
12   for (i = 0; i < N; i++) {
13     compte[i] = 0;
14   }
15
16   f_entree = fopen(argv[1], "r");
17
18   fscanf(f_entree, "%d", &ec);
19   while(ec != MARQUE_FIN) {
20     compte[ec] = compte[ec] + 1;
21     fscanf(f_entree, "%d", &ec);
22   }
23
24   for (i = 0; i < N; i++) {
25     printf("Nombre de %d : %d\n", i, compte[i]);
26   }
27   fclose(f_entree);
28 }
```

Exercice 1. (1 pt) Expliquer en quelques phrases ce que fait ce programme.

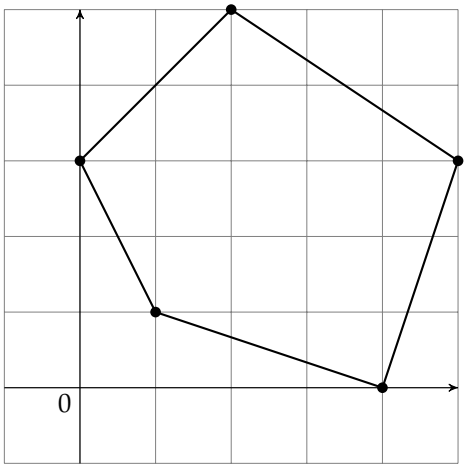
Exercice 2. (2 pt) Quel est le format des entrées de ce programme ? Décrire le domaine de validité des entrées.

Exercice 3. (3 pt) Décrire un jeu de tests fonctionnels pour ce programme.

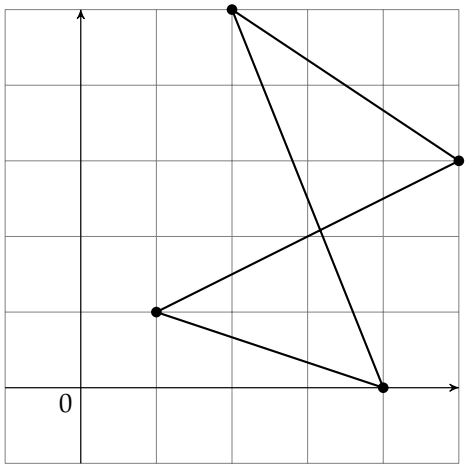
Exercice 4. (3 pt) Donner un exemple de test de robustesse pour ce programme. Quel sera le comportement du programme avec ce test en entrée ? Modifier le programme afin qu'il s'interrompe et affiche une erreur explicite dans le cas où ce test est fourni en entrée (il n'est pas nécessaire de réécrire tout le programme : indiquer quelles lignes sont modifiées ou ajoutées).

Partie II (11 pt)

Un *polygone* peut être représenté par une *séquence de points*, soit la séquence des sommets de ce polygone. On s'intéresse dans cette partie aux *polygones simples*, dont les côtés ne se croisent pas. Par exemple, dans la figure ci-dessous, le polygone de gauche est un polygone simple, et peut être représenté par la séquence $[(1,1),(4,0),(5,3),(2,5),(0,3)]$. Le polygone de droite n'est pas un polygone simple.



Polygone simple



Polygone croisé

Les paquetages `points`, `es_points`, `polygones`, `es_polygones` et `symetries` permettent de manipuler des points et des polygones. Les fichiers sources de ces paquetages sont fournis en annexe.

Exercice 5. (1 pt) Donner un schéma des dépendances entre ces paquetages.

Exercice 6. (2 pt) Écrire un programme qui, en utilisant ces paquetages, lit un polygone dans un fichier, applique une symétrie axiale selon l'axe des abscisses, et écrit le polygone résultant dans un autre fichier. Le nom des deux fichiers est passé en argument de ce programme.

Exercice 7. (2 pt) Écrire un fichier `Makefile` permettant de compiler ces paquetages et le programme écrit pour l'exercice précédent. Ce `Makefile` doit être écrit de telle sorte que la commande `make` sans argument doit générer un exécutable correspondant à ce programme.

Exercice 8. (2 pt) Décrire un jeu de test fonctionnel pour ce programme. Donner un exemple de test fonctionnel.

Exercice 9. (1 pt) Donner un exemple de test de robustesse pour ce programme.

Exercice 10. (3 pt) Écrire un programme permettant de générer de manière aléatoire un fichier contenant N polygones simples. N et le nom du fichier généré sont fournis en argument du programme. Le fichier généré doit pouvoir être lu par la fonction `lecture_polygone`. Si le cas général vous paraît trop compliqué, vous pouvez vous restreindre à une sous-famille de polygones, ou à des polygones ayant certaines propriétés. Dans ce cas, vous devez indiquer précisément quelles sont les propriétés des polygones générés.

Annexes

Fichier points.h

```c
#ifndef _POINTS_H_
#define _POINTS_H_

/* Type point en coordonnée cartésienne */
typedef struct {
  int x;
  int y;
} Point;

#endif
```

Fichier es_points.h

```c
#ifndef _ES_POINTS_H_
#define _ES_POINTS_H_

#include <stdio.h>
#include "points.h"

/* Lecture d'un point p dans le fichier f
   f doit être ouvert en lecture
 */
void lecture_point(FILE * f, Point * p);

/* Ecriture d'un point p dans le fichier f
   f doit être ouvert en écriture
 */
void ecriture_point(FILE * f, Point p);

#endif
```

Fichier es_points.c

```c
#include "es_points.h"

/* Lecture d'un point p dans le fichier f */
void lecture_point(FILE * f, Point * p) {
  fscanf(f, "%d %d", &(p->x), &(p->y));
}

/* Écriture d'un point p dans le fichier f */
void ecriture_point(FILE * f, Point p) {
  fprintf(f, "%d %d\n", p.x, p.y);
}
```

Fichier polygones.h

```c
#ifndef _POLYGONES_H_
#define _POLYGONES_H_

#include <stdio.h>
#include "points.h"

#define NMAX 100

/* Type polygones : séquence de points, représenté dans un tableau
   avec longueur explicite */
typedef struct {
  Point tab[NMAX];
  int nbpoints;
} Polygone;

#endif
```

Fichier es_polygones.h

```c
#ifndef _ES_POLYGONES_H_
#define _ES_POLYGONES_H_

#include <stdio.h>
#include "polygones.h"

/* Lecture d'un polygone p dans un fichier f
   f doit être ouvert en lecture
*/
void lecture_polygone(FILE * f, Polygone * p);

/* Écriture d'un polygone p dans un fichier f
   f doit être ouvert en écriture
 */
void ecriture_polygone(FILE * f, Polygone * p);

#endif
```

Fichier es_polygones.c

```c
#include "es_polygones.h"
#include "es_points.h"

void lecture_polygone(FILE * f, Polygone * p) {
  int i;
  /* Lecture du nombre de sommets */
  fscanf(f, "%d", &(p->nbpoints));
  /* Lecture des coordonnées des sommets */
  for (i = 0; i < p->nbpoints; i++) {
    lecture_point(f, &(p->tab[i]));
  }
}

void ecriture_polygone(FILE * f, Polygone * p) {
  int i;
  /* Ecriture du nombre de sommets */
  fprintf(f, "%d\n", p->nbpoints);
  /* Ecriture des coordonnées des sommets */
  for (i = 0; i < p->nbpoints; i++) {
    ecriture_point(f, p->tab[i]);
  }
}
```

Fichier symetries.h

```h
#ifndef _SYMETRIES_H_
#define _SYMETRIES_H_

#include "polygones.h"

/* Applique au polygone p une symétrie centrale par rapport à
   l'origine */
void symetrie_origine(Polygone * p);

/* Applique au polygone p une symétrie axiale par rapport à l'axe des
   abscisses*/
void symetrie_x(Polygone * p);

/* Applique au polygone p une symétrie axiale par rapport à l'axe des
   ordonnées*/
void symetrie_y(Polygone * p);

#endif
```

Fichier symetries.c

```c
#include "symetries.h"

/* Applique au polygone p une symétrie centrale par rapport à
   l'origine */
void symetrie_origine(Polygone * p) {
  int i;
  for(i = 0; i < p->nbpoints; i++) {
    p->tab[i].x = - p->tab[i].x;
    p->tab[i].y = - p->tab[i].y;
  }
}

/* Applique au polygone p une symétrie axiale par rapport à l'axe des
   abscisses*/
void symetrie_x(Polygone * p) {
  int i;
  for(i = 0; i < p->nbpoints; i++) {
    p->tab[i].y = - p->tab[i].y;
  }
}

/* Applique au polygone p une symétrie axiale par rapport à l'axe des
   ordonnées*/
void symetrie_y(Polygone * p) {
  int i;
  for(i = 0; i < p->nbpoints; i++) {
    p->tab[i].x = - p->tab[i].x;
  }
}
```