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.

Éléments de correction

Partie I (9 pt)

Soit le programme C suivant :

```c
#include <stdlib.h>
#include <stdio.h>

#define N 16
#define MARQUE_FIN -1

int main(int argc, char ** argv) {
  FILE * f_entree;
  int ec,i;
  int compte[N];

  for (i = 0; i < N; i++) {
    compte[i] = 0;
  }

  f_entree = fopen(argv[1], "r");

  fscanf(f_entree, "%d", &ec);
  while(ec != MARQUE_FIN) {
    compte[ec] = compte[ec] + 1;
    fscanf(f_entree, "%d", &ec);
  }

  for (i = 0; i < N; i++) {
    printf("Nombre de %d : %d\n", i, compte[i]);
  }
  fclose(f_entree);
}
```

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

Ce programme prend en entrée une séquence d'entiers sur l'intervalle $[0, 15]$*, terminée par l'entier* -1*. Il affiche pour chaque entier de cet intervalle le nombre d'occurrence de cet entier dans la séquence d'entrée.*

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

Une entrée de ce programme est un fichier comprenant une séquence d'entiers sur l'intervalle $[0, 15]$*. Cette séquence est terminée par l'entier* -1*, non inclut dans la séquence.*

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

Les critères des tests du jeu de tests sont : la taille de la séquence, les valeurs de la séquence, l'ordre de ces valeurs.

- *Cas limite pour la taille de la séquence : séquence vide,* $[]$
- *Cas limite pour la taille de la séquence : séquence d'un élément*
  - *Cas limites pour les valeurs : bornes de l'intervalle,* $[0]$, $[15]$
  - *Valeur(s) quelconque(s) : par exemple,* $[1]$, $[4]$, *...*
- *Taille de la séquence quelconque, par exemple pour des séquences de taille 3 :*
  - *Cas limites pour les valeurs : bornes de l'intervalle*
    - *Cas limite pour la position des valeurs : en début de séquence (*$[0,3,4]$, $[15,3,4]$*), en fin de séquence (*$[3,4,0]$, $[3,4,15]$*)*
    - *Cas quelconque : valeurs en milieu de séquence,* $[3,0,4]$, *...*
    - *Répétition de la même valeur : permutations de* $[0,0,3]$
  - *Valeurs quelconques :*
    - *Séquences sans répétition de valeur* $[1,2,3]$ *(et ses permutations)*
    - *Séquences avec répétition : 2 fois la même valeur, une seule valeur dans la séquence (cas limite)*
    - *Séquences avec toutes les valeurs de l'intervalle (et permutations)*

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).

Deux exemples :

- *Une séquence comportant un entier en-dehors de l'intervalle* $[0,15]$*, par exemple la séquence* $[16]$*. Le comportement n'est pas prévisible, l'instruction* `compte[ec] = compte[ec] + 1`*, avec* `ec` $= 16$*, modifie un emplacement mémoire arbitraire. Selon l'environnement d'exécution, le programme peut s'arrêter sur «segmentation fault» ou avoir un comportement indéfini.*
  Pour traiter cette erreur, on peut modifier la boucle while (lignes 19 à 22) comme suit :

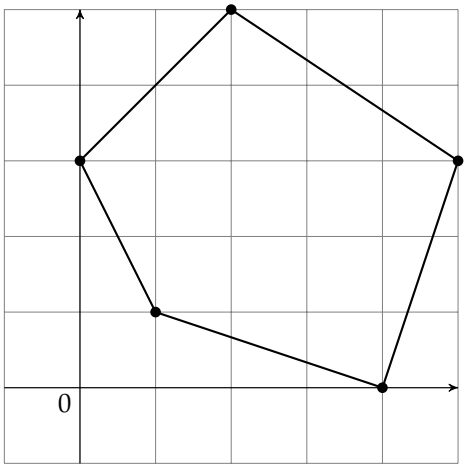
```
1  while(ec != MARQUE_FIN) {
2    if (ec >= N) {
3      fprintf(stderr, "Erreur : valeur %d en-dehors de l'intervalle\n", ec);
4      exit(1);
5    }
6    compte[ec] = compte[ec] + 1;
7    fscanf(f_entree, "%d", &ec);
8  }
```

- *Séquence non terminée par -1. Le programme ne sort pas de la boucle while, la fonction* `fscanf` *une fois arrivée à la fin du fichier ne modifie plus la valeur de la variable* `ec`*. Le programme ne termine pas, et n'affiche rien.*
  Pour traiter cette erreur, on peut utiliser la valeur de retour de `fscanf` *(renvoie le nombre de valeurs effectivement lues).*

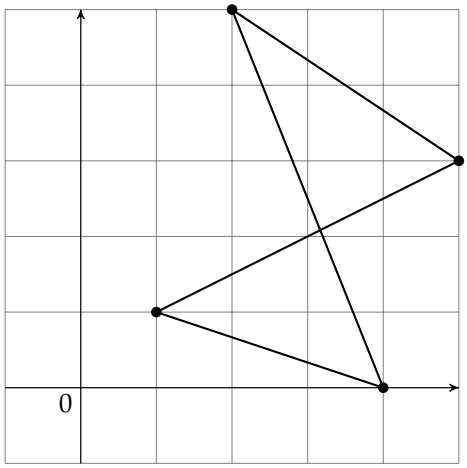
```
1  int nb_val_lues;
2  ...
3  nb_val_lues = fscanf(f_entree, "%d", &ec);
4  while((ec != MARQUE_FIN) && (nb_val_lues == 1)) {
5    compte[ec] = compte[ec] + 1;
6    nb_val_lues = fscanf(f_entree, "%d", &ec);
7  }
8  if (nb_val_lues != 1) {
9    fprintf(stderr,"Erreur de lecture de la séquence d'entrée\n");
10 }
11 ...
```

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.



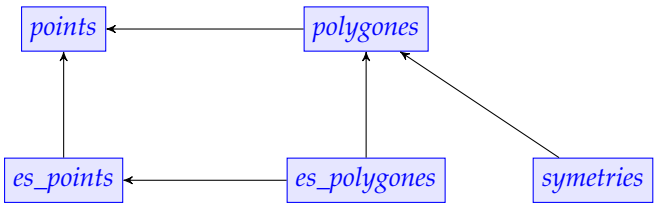
Polygone simple



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.

```
1  #include <stdio.h>
2
3  #include "polygones.h"
4  #include "es_polygones.h"
5  #include "symetries.h"
6
7  int main(int argc, char ** argv) {
8    FILE * f_entree;
9    FILE * f_sortie;
10   Polygone p;
11
12   f_entree = fopen(argv[1], "r");
13   lecture_polygone(f_entree, &p);
14   fclose(f_entree);
15
16   symetrie_x(&p);
17
18   f_sortie = fopen(argv[2], "w");
19   ecriture_polygone(f_sortie, &p);
20   fclose(f_sortie);
21 }
```

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.

```
CC=clang -Wall

all: test_symetrie

%.o: %.c
	$(CC) -c $<

# Règles de compilation

es_points.o: es_points.c es_points.h points.h

es_polygones.o: es_polygones.c es_polygones.h polygones.h es_points.h points.h

symetries.o: symetries.c symetries.h polygones.h points.h

test_symetrie.o: test_symetrie.c symetries.h polygones.h points.h

# Édition de liens

test_symetrie: test_symetrie.o symetries.o es_polygones.o es_points.o
```

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

- *Cas limite : polygones symétriques selon l'axe des abscisses : triangles/carrés/polygones quelconques de tailles et positions sur l'axe variables*
- *Cas quelconques : faire varier le nombre de côtés, la taille et la position des polygones.*

Un exemple de test (triangle) :

```
3
0 0
1 0
0 1
```

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

Par exemple, un nombre de points incorrect :

```
3
0 0
1 0
```

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.

On se retreint à la génération de rectangles, à partir de leur diagonale (permet d'éviter la génération de polygones croisés) :

```c
#include <stdlib.h>
#include <stdio.h>

int main(int argc, char ** argv) {
  FILE * f_sortie;
  int nb_polygones;
  int i;
  int x1, y1, x2, y2;

  /* Initialisation du germe pour la génération aléatoire */
  srand(1);

  nb_polygones = atoi(argv[1]);
  f_sortie = fopen(argv[2], "w");

  for(i = 0; i < nb_polygones; i++) {
    /* Écriture du nombre de côtés */
    fprintf(f_sortie, "4\n");
    /* Tirage aléatoire de deux sommets */
    x1 = rand();
    y1 = rand();
    x2 = rand();
    y2 = rand();
    /* Écriture du rectangle */
    fprintf(f_sortie, "%d %d\n", x1, y1);
    fprintf(f_sortie, "%d %d\n", x2, y1);
    fprintf(f_sortie, "%d %d\n", x2, y2);
    fprintf(f_sortie, "%d %d\n", x1, y2);
  }

  fclose(f_sortie);
}
```

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;
  }
}
```