Programmation et Langages 1

Examen du 3 décembre 2019

(Eléments de correction)

Exercice 1

Q1.

Cette approche ne permet pas de détecter 2 inversions de bits, car la parité sera inchangée. Par contre elle permet de détecter un nombre impaire de fautes (donc 3 inversions).

Q2.

```
void emission(char *S){
  int i=0 ; // compteur
  char b; // bit de controle

  while (S[i] != '\0') {
  if (poids(S[i]) % 2 == 0) b=0 ; else b=1 ;
       /* on ajoute le bit de controle au caractere courant */
  ajoute_bcontrole(&(S[i]), b) ;
  envoyer (S[i]) ;  // on envoie le caratere courant
  i=i+1 ;
  } ;
  // on traite le caractere de fin de chaine ...
  if (poids(S[i]) % 2 == 0) b=1 ; else b=0 ;
  ajoute_bcontrole(&(S[i]), b) ;  // ajout du bit de controle
  envoyer (S[i]) ;  // on envoie le caratere de fin de chaine
}
```

Q3.

```
void reception(char *S1, char *S2){
  int i=0 ;  // compteur
  int fini=0 ;
  int erreur=0;
  char b ;  // bit de controle
  while (!fini) {
  supprime_bcontrole(&(S1[i]), &b);
      /* on verifie que le transfert est correct */
  if (b==0 && poids(S1[i]) % 2 != 1) {
          /* bit de controle incorrect */
  fini=1 ;
          erreur=1 ;
  printf("erreur !\n") ;
  } ;
  if (S1[i] == '\0')
  fini=1 ;  // on a atteint la fin de chaine
  i=i+1 ;
  } ;
  if (!erreur)
  strcpy(S2, S1) ;  // on recopie S1 dans S2
}
```

Q4.

```
unsigned char poids (char x) {
char i, b, ;
    char p=0 ;  // poids de x
for (i=0 ; i<7 ; i++)  {
b = x & 1 ;  // b est le bit de poids faible
x = x>>1 ;  // on decale les bits de x vers la droite
    p = p+b ;   // on met a jour le poids
} ;
return p ;
}


void ajoute_bcontrole(char *x, char b) {
*x = ((*x)*2) + b ;
}


void supprime_bcontrole(char *x, char *b) {
    *b = *x & 1 ;  // b est le bit de poids faible (= bit de controle)
*x = ((*x)/2);
}
```

Exercice 2 (∼ 12 points)

Partie 1.

Q2 et Q3.

contenu du fichier `fap.h`

```
typedef int Prio ;   // le type Prio identique aux entiers

typedef struct cellule {
                char elem ;
                Prio prio ;
                struct cellule *suiv ;
        } Cellule ;    // un element de la FaP

typedef Cellule *FaP ;  // le type FaP est le type pointeur sur Cellule

void FaPVide (FaP *F) ;
int EstVide (FaP F) ;
void Inserer (FaP *F, char c, Prio p) ;
void Extraire (FaP *F, char *c) ;
```

contenu du fichier `fap.c`

```
#include <stdlib.h>
#include "fap.h"

void FaPVide (FaP *F) {
  *F = NULL ;
}

int EstVide (FaP F) {
   return F == NULL ;
}

void Inserer (FaP *F, char c, Prio p) {
  FaP Nouv ;

  /* insertion en tete */
  Nouv = (FaP) malloc (sizeof (Cellule)) ;
  Nouv->elem = c ; Nouv->prio = p ;
  Nouv->suiv = *F ;
  *F = Nouv ;
}
```

```
void Extraire (FaP *F, char *c) {
  /* suppression d'un element de prio max */

  FaP Cour, Prec ; // pointeurs sur courant et precedent
  FaP Max, PMax ;  // pointeur sur element de prio max et son precedent
  Prio pmax ;   // la prio max contenue dans la fap

  /* recheche de l'element de prio max et de son precedent dans la liste */

  Prec = NULL ; PMax = NULL ;
  if (*F != NULL) {
    Max = *F ; pmax = Max->prio ; Cour = (*F)->suiv ; Prec = *F ;
    while (Cour != NULL) {
        if (Cour->prio > pmax) {
           PMax = Prec ; Max = Cour ; pmax = Cour->prio ;
        } ;
    Prec = Cour ;
    Cour = Cour->suiv ;
    }

    /* affectation du prametre c et supression de l'element de prio max */
    *c = Max->elem ;
    if (PMax != NULL) {
      PMax->suiv = Max->suiv ;
    } else {
      *F = (*F)->suiv ;
    } ;
   free(Max) ;
  } ;
}
```

Q4. contenu du fichier `main.c`

```
#include <stdio.h>
#include "fap.h"

int main() {

FaP F ;
        char c, x ;
        Prio p ;

        FaPVide (&F) ;

        scanf("%c", &c) ;
        while (c != 'Q') {
                switch (c) {
                        case 'I':
                                scanf("\n%c", &x) ;
                                scanf("%d", &p) ;
                                Inserer (&F, x, p) ;
                                break ;
```

```
                        case 'E':
                                Extraire (&F, &x) ;
                                printf("%c\n", x) ;
                                break ;

                        default: break ;
                }
                scanf("\n%c", &c) ;
        }

        return 0 ;
```

Q5. contenu du fichier `Makefile`

```
test_fap : fap.o main.o
gcc -Wall -g -o test_fap fap.o main.o

fap.o : fap.c fap.h
gcc -Wall -g -c fap.c

main.o : main.c fap.h
gcc -Wall -g -c main.c
```

Partie 2.

Q6. contenu du fichier `fap.h`

```
#define PMAX 10

typedef int Prio ;

typedef struct cellule {
   char elem ;
   struct cellule *suiv ;
} Cellule ;

    // PCellule est le type pointeur de Cellule
typedef Cellule *PCellule ;

   // FaP est le type tableau sur 0..PMAX-1 de PCellule
typedef PCellule FaP[PMAX] ;

void FaPVide (FaP F) ;
int EstVide (FaP F) ;
void Inserer (FaP F, char c, Prio p) ;
void Extraire (FaP F, char *c) ;
```

Q7. contenu du fichier `fap.c`

```
#include <stdlib.h>
#include "fap.h"

void FaPVide (FaP F) {
   int i ;

   // on initialise a NULL l'ensemble des F[i]
   for (i=0; i<PMAX ; i++) {
      F[i] = NULL ;
   }
}

int EstVide (FaP F) {
   int i=0 ;
   while (i<PMAX && F[i] == NULL) i++ ;
   return i=PMAX ;  // tous les F[i] sont NULL
}

void Inserer (FaP F, char c, Prio p) {
   PCellule Nouv = (PCellule) malloc (sizeof (Cellule)) ;

   /* insertion en tete dans F[p] */
   Nouv->elem = c ;
   Nouv->suiv = F[p] ;
   F[p] = Nouv ;
}


void Extraire (FaP F, char *c) {
   /* suppression d'un element de prio max */
   int prio_max=PMAX-1 ;
   PCellule tmp ;

   while (F[prio_max]>=0 && F[prio_max] == NULL) prio_max=prio_max-1 ;
   if (prio_max >= 0 && F[prio_max] != NULL) {
      *c=F[prio_max]->elem ;
      tmp = F[prio_max] ;
      F[prio_max] = F[prio_max]->suiv ;
      free(tmp) ;
   } ;
}
```

Exercice 3

Q1.

1. une première solution possible consiste à utiliser un (grand) tableau, où chaque élement de ce tableau est une structure contenant :
   — le nom du pays
   — les informations qui lui sont associées

```
    /* la structure Infos */
   typedef struct {
int superficie ;
int nbHabitants ;
int Pib ; } Infos ;

    /* le type nomPays = tableau de MAXCAR caracteres */
   typedef char nomPays[MAXCAR] ;

    /* la structure InfosPays = <nomPays, Infos> */

   typedef struct {
nomPays nom ;
        Infos infos :
   }  InfosPays ;

/* le type TabInfosPays = tableau de NBPays InfosPays */
   typedef struct InfosPays TabInfoPays[NBPays] ;
```

2. une variante de cette solution consiste à utiliser un tableau de couples (nom de pays, pointeur vers les information correspondant à ce pays).

```
    /* la structure Infos */
   typedef struct {
int superficie ;
int nbHabitants ;
int Pib ; } Infos ;

    /* le type nomPays = tableau de MAXCAR caracteres */
   typedef char nomPays[MAXCAR] ;

    /* la structure InfosPays = <nomPays, * Infos> */

   typedef struct {
nomPays nom ;
        Infos *infos :
   }  InfosPays ;

/* le type TabInfosPays = tableau de NBPays InfosPays */
   typedef struct InfosPays TabInfoPays[NBPays] ;
```

3. On peut imaginer aussi une solution avec une liste chaînéee de structures `InfosPays`, etc.

Q2. Pour les 2 solutions proposées, un algorithme possible est le suivant :

`Infos infoPays (char *pays) :`

1. rechercher dans le tableau `T` de type `TabInfoPays` l'indice `i` tel que `T[i].nom` est egal à `pays` (recherche séquentielle)
2. renvoyer en résultat `T[i].infos`