UGA - UFR IM^{2}AG **Master 2 CCI**

Programmation et Langages 1

Examen du 3 décembre 2019

Durée : deux heures – **Documents :** autorisés – Barème indicatif (sur 21 pts)

On rappelle qu'en langage C une chaîne de caractères est représentée par un tableau de caractères (donc de type `char *`) terminé par le caractère spécial noté `'\0'`.

Exercice 1 (∼ 5 points)

La transmission de données sur un réseau informatique peut parfois être perturbée par des éléments extérieurs (champs électromagnétiques, variations de tensions électriques, etc.) avec pour effet de "permuter" de la valeur "0" à la valeur "1" (ou inversement !) certains bits transmis. Pour s'assurer que les données transmises sont correctes une solution consiste à ajouter des *informations de contrôle*. Une approche possible consiste à "découper" les données émises en sous-séquences de 7 bits et à ajouter un **bit de contrôle** [1] à la fin de chaque sous-séquence. Plus précisément :

- si la sous-séquence contient un nombre **pair** de bits égaux à 1, le bit de contrôle vaut 1 ;
- si la sous-séquence contient un nombre **impair** de bits égaux à 1, le bit de contrôle vaut 0.

A la réception on vérifie alors que ce bit de contrôle est correct (il vaut 1 si et seulement si la sous-séquence de 7 bits reçue contient un nombre pair de 1). Si c'est le cas on le supprime et on délivre la sous-séquence initiale de 7 bits, sinon on envoie un message d'erreur à l'émetteur.

Exemple : les caractères `'C'` et `'H'` ont pour code ASCII 67 et 72. En base deux, ces valeurs sont codées (sur 7 bits) par `1000011` et `1001000`. Le bit de contrôle associé à `'C'` est donc **1** et celui asocié à `H` est **0**. Pour transmettre ces deux caractères, on transmettra donc `10000111` et `10010000`.

Q1. Cette technique permet-elle de détecter si la sous-séquence contient deux erreurs (2 bits permutés) ? Et trois erreurs ?

En langage C, le type `char` représente une "valeur entière codée en base deux sur 8 bits". Ce type peut donc être utilisé (efficacement !) pour représenter une sous-séquence de 8 bits. On dispose dans la suite des fonctions suivantes :

```
int poids (char x) ;
// poids(x) est le nombre de bits egaux a 1 dans la representation en base 2 de x
// ex : poids(0)=0, poids(2)=1, poids(3)=2, poids(4)=1

void ajoute_bcontrole (char *x, char b) ;
// etat initial :
//    - x est une valeur codee sur 7 bits
//    - b vaut 0 ou 1
// etat final : le bit de controle b a ete ajoute a x
```

1. il s'agira ici d'un **bit de parité**

```
void supprimer_bcontrole (char *x, char *b) ;
// etat initial :
//    x est une valeur codee sur 8 bits contenant un bit de controle
// etat final : le bit de controle b a ete supprime de x

void envoyer(char x) ; // envoie sur le reseau un paquet de 8 bits
```

Q2. Ecrivez une fonction `emission (char *S)` qui prend en paramètre une chaine de caractère `S` et qui envoie sur le réseau – un par un – chacun des caractères de `S` (y compris le caractère `'\0'`) après leur avoir ajouté leur bit de contrôle.

Q3. Ecrivez une fonction `reception (char *S1, char *S2)` qui prend en paramètre une chaine de caractères `S1`, supposée reçue du réseau, et qui vérifie que le bit de contrôle de chacun de ses caractères est correct. Si c'est le cas, `S1` est copiée dans `S2` (sans les bits de contrôle !), sinon un message d'erreur est affiché à l'écran. Vous pouvez utiliser la fonction prédéfinie `strcpy (char *dest, chat *src)` qui copie la chaine `src` dans la chaine `dest`. On supposera que la taille de `S2` est suffisante.

Q4. (question subsidiaire, hors-barême)

On dipose en C des opérateurs suivants :
- l'operateur `>>` permet de décaler vers la droite (diviser par deux) une valeur entière ;
- l'operateur `<<` permet de décaler vers la gauche (multiplier par deux) une valeur entière ;
- l'opération `x & 1` renvoie la valeur du bit de poids faible (le plus à doite) de la valeur entière `x`.

A l'aide de ces opérateurs écrivez les fonctions `poids`, `ajoute_bcontrole` et `supprimer_bcontrole`.

Exercice 2 ($\sim$ 12 points)

On souhaite implémenter en C une structure de données permettant de gérer une **file d'attente à priorité (FaP)** contenant des **caractères**. Le fonctionnement de cette structure est le suivant :
- on dispose d'un type de nom Prio qui représente un ensemble ordonné de *priorités* ;
- lorsque l'on **insère** un caractère dans la FaP, on lui associe une priorité ;
- lorsque l'on **extrait** un caractère de la FaP, le caractère extrait est l'un des caractères de **priorité maximale** présent dans la Fap.

On donne ci-dessous un exemple d'exécution, en supposant que les priorités sont représentées par des entiers :

1. initialement la FaP `F` est vide ;
2. on **insère** le caractère `'x'` avec la priorité `3`
   `F` contient donc le couple `('x',3)`.
3. on **insère** le caractère `'a'` avec la priorité `8`
   `F` contient donc les couple `('a',8)` et `('x',3)`
4. on **insère** le caractère `'z'` avec la priorité `4`
   `F` contient donc les couple `('z',4)` et `('a',8)` et `('x', 3)`
5. on `extrait` un élément :
   - l'élément extrait est `'a'` (sa priorité est maximale dans `F`)
   - `F` contient maintenant les couples `('z',4)` et `('x', 3)`
6. on **insère** le caractère `'b'` avec la priorité `4`
   `F` contient donc les couple `('b',4)` et `('z',4)` et `('x', 3)`
7. on `extrait` un élément :
   - l'élément extrait est `'b'` [2] (sa priorité est maximale dans `F`)
   - `F` contient maintenant les couples `('z',4)` et `('x', 3)`

2. Ce pourrait aussi être 'z', qui a la même priorité que `'b'`

Partie 1.

On considère une première solution dans laquelle une FaP est représentée par une liste chaînée de couples (élément, priorité) selon le lexique suivant :

Prio : le type entier ;
Cellule : un type ;
FaP : le type pointeur de Cellule ;
Cellule : le type { elem : caractère ; prio : Prio ; suiv : FaP } ;

Q2. Ecrire en C le contenu d'un fichier `fap.h` qui implémente le lexique ci-dessus.

Q3. On définit les sous-programmes suivants permettant de manipuler une FaP :

FaPVide : action (résultat F : FaP)
{ etat initial : indifférent ; état final : F est une fap vide}

EstVide : fonction (F : FaP) → booléen
{EstVide(F) vaut vrai ssi F est une fap vide}

Inserer : action (donnée-résultat F : FaP, donnée c : caractère ; donnée p : Prio)
{ état initial : indifférent ;
état final : le couple (c,p) est ajouté à F}

Extraire : action (donnée-résultat F : FaP, résultat c : caractère)
{ état initial : indifférent ;
état final :
- si non EstVide(F), alors un caractère (quelconque) *de priorité maximale dans F* est supprimé de F et affecté à c
- si estVide(F), alors F est inchangée et c n'est pas affecté }

Ecrire en C le contenu des fichiers suivants :
- Dans `fap.h` : en-tête des fonctions **FaPVide**, **EstVide**, **Inserer** et **Extraire**.
- Dans `fap.c` : corps des fonctions **FaPVide**, **EstVide**, **Inserer** et **Extraire**.

Q4. Ecrire un fichier `main.c` contenant un programme principal dont le contenu est le suivant :

1. déclarer (et initialiser) une fap vide `F` ;
2. déclarer deux caractères de nom c et x ;
3. déclarer un Prio p
4. lire la valeur de c au clavier ;
5. tant que c est différent de 'Q'
   (a) selon c
   - c = 'I' : lire les valeur de x et p au clavier ; Insérer (F, x, p)
   - c = 'E' : Extraire (F, x) ; afficher la valeur de x
   - sinon : {ne rien faire ...}

   (b) lire la valeur de c au clavier ;
6. fin

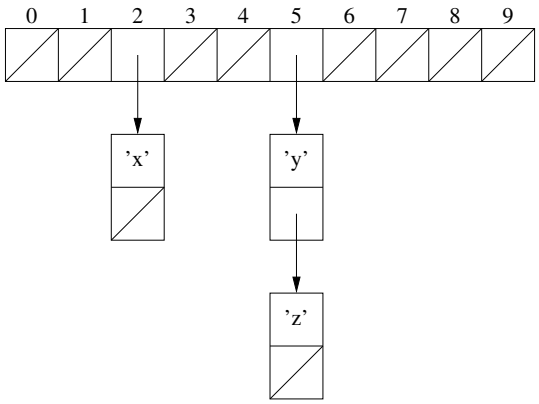
Q5. Ecrire un fichier Makefile permettant de compiler le fichier `main.c` (et les autres fichiers nécessaires) par **compilation séparée** pour produire un exécutable de nom `test_fap`.

Partie 2.

On étudie maintenant une autre implémentation possible de la FaP dans laquelle :

— le type `Prio` est le type `Entier sur 0..Pmax-1`, pour une constante `Pmax=10` ;
— le type `FaP` est un tableau sur `Prio` dont les éléments sont des listes chaînées de caractères : la case d'indice `p` de ce tableau contient la liste des caractères présents dans la FaP avec une priorité `p`

On donne ci-dessous un exemple de FaP contenant les couples `('x',2)` et `('y',5)` et `('z', 5)` :



Q6. Ré-écrivez le contenu du fichier `fap.h` correspondant à cette nouvelle implémentation.

Q7. Ré-écrivez alors en conséquence le code des fonctions **FaPVide**, **EstVide**, **Inserer** et **Extraire**.

Exercice 3 (∼ 4 points)

On souhaite représenter en langage C une structure de données permettant de mémoriser, pour chaque pays du monde, les informations suivantes : superficie, nombre d'habitants, produit intérieur brut. Les pays seront identifiés par leurs noms (des chaînes de caractères), et cette structure de données doit permettre de coder la fonction suivante :
`Infos infoPays (char *pays) ;` {renvoie les informations associées au pays de non `p`}.

Q1. Définissez en langage C (comme on le ferait dans un fichier `.h`) les types nécessaires pour représenter cette structure de données. On supposera que :
— le nombre total de pays est une constante `NBPays`
— la longueur maximale (en nombre de caractères) du nom d'un pays est une constante `MAXCAR`
— la superficie, le produit intérieur brut et le nombre d'habitants sont représentés par un type `Infos` représentant un triplet d'entiers.
Il est conseillé de fournir un **schéma** qui explicite votre structure de données choisie ...

Q2. Expliquer (informellement, en quelques lignes) un algorithme permettant de coder la fonction `infoPays` à l'aide de votre structure de données.