université de BORDEAUX	**ANNEE UNIVERSITAIRE 2015 / 2016** **S1 D'AUTOMNE** **PARCOURS / ETAPE:** Informatique **Code UE :** J1INAW02 **Epreuve :** Programmation 1 **Date :** 05/01/2015 **Heure :** 8H30 **Durée :** 1H30 Documents : autorisés (**uniquement une feuille A4 Recto/Verso manuscrite**) Epreuve de M : Blin Guillaume	**Collège Sciences et technologies**

Indiquez votre code **d'anonymat** : N°

La notation tiendra compte de la clarté de l'écriture des réponses ainsi que de leur précision. Par la suite, vous devez répondre directement sur l'énoncé à toute question étant suivie d'un cadre prévu à cet effet.

Exercice 1. (5 points)

- **Q1 (1 point) :** Quelles sont les valeurs des entiers `a`, `b` et `c` affichés à la fin de l'exécution du programme suivant?

```
1.  #include <stdio.h>
2.  #include <stdlib.h>
3.  int a=5;
4.  int b=6;
5.
6.  int g(int a){
7.   int b;
8.   b=3*a;
9.   return b;
10. }
11.
12. int f(int * p){
13.  *(p)=10*a;
14.  return (*(p))*g(a);
15. }
16.
17.
18. int main(void){
19.  int b=2;
20.  int c=3;
21.  c=f(&b);
22.  printf("%d %d %d\n",a,b,c);
23.  return EXIT_SUCCESS;
24. }
```

Valeur de a	Valeur de b	Valeur de c

- **Q2** (**1 point**) : Expliquer de manière synthétique et avec vos mots les différents modes d'allocation et de stockage mémoire des variables et les conséquences (e.g., différence de localisation, de durée de vie, de visibilité, …).

- **Q3 (1 point) :** Expliquez avec vos mots pourquoi en C le seul mode de passage de paramètre d'une fonction est celui par copie (*aide*: pensez à la pile d'appel de fonction).

Indiquez votre code **d'anonymat** : N°

- **Q4 (1 point) :** Expliquer le résultat inattendu de ce programme qui affiche `sum=32768.000000` (*aide*: on ne demande pas d'expliquer le résultat précisément mais pourquoi on obtient un résultat très éloigné de celui attendu – i.e., `999999.997`)

```
1.  #include <stdio.h>
2.  #include <stdlib.h>
3.  int main(void) {
4.    int i;
5.    float sum=0.f;
6.    for (i=0;i<999999997;i++) {
7.      sum=sum+0.001f;
8.    }
9.    printf("sum=%f\n",sum);
10.   return EXIT_SUCCESS;
11. }
```

- **Q5 (1 point) :** Quelles sont, pour vous, les règles principales à respecter pour garantir une maintenabilité, une réutilisation et une possibilité d'évolution les plus élevées possibles ?

Exercice 2. (10 points)

- Soit la fonction **foo** suivante:

```
1.   #include <stdio.h>
2.   #include <stdlib.h>
3.
4.   char * foo(char * start, char * end){
5.     if(start > end){
6.       fprintf(stderr, "'end' should not appear before 'start'");
7.       return NULL;
8.     }
9.     char * res = start;
10.    char val = *(start);
11.    for(char * tmp = start+1; tmp <= end; tmp++){
12.      if(*(tmp) < val) {
13.        res = tmp;
14.        val = *(tmp);
15.      }
16.    }
17.    return res;
18.  }
19.
20.  int main(void) {
21.    char s[] = {'O','F','F','E','R','T'};
22.    char * m = foo(s,s+5);
23.    if(m != NULL){
24.      printf("foo=%c\n",*(m));
25.    }
26.    return EXIT_SUCCESS;
27.  }
```

- **Q1 (2 point) :** Expliquer ce que vérifie le premier test (lignes 5 à 8) au début de la fonction. De façon générale, indiquer quelles sont les diverses manières de traiter un cas d'erreur.

Indiquez votre code **d'anonymat** : N°

- **Q2 (2 points) :** Expliquer ce que fait la fonction **`foo`**.

- **Q3 (2 points) :** En utilisant la fonction **`foo`** définie précédemment, proposer une fonction **`charsort`** dont le prototype est le suivant :

```
char * charsort(char * w, int len);
```

Le paramètre **`w`** représente le tableau de caractères à trier et **`len`** le nombre de caractères dans le tableau. La fonction retourne **une copie** du tableau **`w`** dont les éléments sont triés du plus petit au plus grand (*aide*: L'évaluation ne tiendra pas compte de la complexité de la solution proposée).

- **Q4 (2 points) :** Nous souhaitons offrir la possibilité de baser le tri sur une fonction quelconque (i.e., pas forcément l'ordre lexicographique). Nous proposons donc de changer la signature de la fonction **`charsort`** en **:**

```
char * charsort (char * w, int len, bool (*comp) (char, char) );
```

Le pointeur de fonction **`comp`**, représente la fonction de comparaison entre deux caractères. Quelles sont les modifications à apporter au code et à la signature de la fonction **`foo`** ainsi qu'au code de la fonction **`charsort`** pour pouvoir trier un tableau de caractères en utilisant la fonction de comparaison pointée par **`comp`** ?
CONSIGNES : Après avoir numéroté les lignes du code source de votre fonction **`charsort`** de la question **Q3**, pour répondre à la question précédente, préciser **uniquement** les changements à apporter en indiquant les lignes où se produisent ces derniers.

- **Q5 (2 point) :** En utilisant la dernière version de la fonction **`charsort`**, écrire les deux fonctions

```
char * increasingSorting(char * w, int len);
```
et
```
char * decreasingSorting(char * w, int len);
```

qui retournent respectivement une copie de **`w`** trié dans l'ordre alphabétique et dans l'ordre inverse (*aide* : vous avez toute liberté de créer d'autres fonctions si nécessaire. Ces dernières ne devront pas être visibles en dehors de votre fichier source).

Exercice 3. (5 points)
BF est un langage de programmation minimaliste, inventé par Urban Müller en 1993. Il suit un modèle de machine simple, consistant en **un tableau d'octets initialisés à 0**, **d'une tête de lecture/écriture unique** (que l'on peut voir comme un simple pointeur sur le tableau) qui est positionnée sur le premier octet du tableau au démarrage et de **deux files d'octets** pour les entrées et sorties.
Les huit instructions du langage, chacune codée par un seul caractère, sont les suivantes :

Caract.	*Signification*
>	*incrémente (augmente de 1) le pointeur.*
<	*décrémente (diminue de 1) le pointeur.*
+	*incrémente l'octet du tableau sur lequel est positionné le pointeur (l'octet pointé).*
−	*décrémente l'octet pointé.*
.	*sortie de l'octet pointé (valeur ASCII).*
,	*entrée d'un octet dans le tableau à l'endroit où est positionné le pointeur (valeur ASCII).*
[	*saute à l'instruction après le] correspondant si l'octet pointé vaut 0.*
]	*retourne à l'instruction après le [si l'octet pointé ne vaut pas 0.*

Le programme suivant affiche le traditionnel "Hello World! " suivi d'un saut de ligne à l'écran.
Noter que, par souci de lisibilité, le code a été divisé en plusieurs lignes et des commentaires ont été ajoutés. *BF* considère comme étant des commentaires tous les caractères ne correspondant pas à une instruction (c.f. tableau plus haut). **Il ne s'agit pas de comprendre le code suivant. Il est donné à titre d'exemple.**

```
++++++++++
[   Boucle initiale qui affecte des valeurs utiles au tableau
   >+++++++>++++++++++>+++>+<<<<-
]
    à la sortie de la boucle le tableau contient: 0 70 100 300 10 0 0 0 0 0 etc
>++.                    'H'    = 72  (70  plus 2)
>+.                     'e'    = 101 (100 plus 1)
+++++++.                'l'    = 108 (101 plus 7)
.                       'l'    = 108
+++.                    'o'    = 111 (108 plus 3)
>++.                    espace = 32  (30  plus 2)
<<+++++++++++++++.      'W'    = 87  (72  plus 15)
>.                      'o'    = 111
+++.                    'r'    = 114 (111 plus  3)
------.                 'l'    = 108 (114 moins 6)
--------.               'd'    = 100 (108 moins 8)
>+.                     '!'    = 33  (32  plus  1)
>.                      nouvelle ligne = 10
```

Nous souhaitons proposer un programme permettant d'interpréter et exécuter un programme décrit en ***BF* simplifié où une seule boucle peut exister dans le code** (comme dans l'exemple Hello World). **En d'autres termes, on ne peut pas avoir plus d'un caractère '[' ni plus d'un caractère ']' dans le code**.

Indiquez votre code **d'anonymat** : N°

Pour ce faire, nous considérerons que

- Le tableau d'octets est représenté par une variable globale définie comme suit en utilisant la constante `MEM_SIZE` (que vous n'avez pas à définir)

```
static unsigned char memory[MEM_SIZE] ;
```

- La tête de lecture/écriture est représentée par une variable `ptr` pointant initialement sur la première case de `memory`

```
static unsigned char* ptr = memory;
```

- Et que les files d'octets pour les entrées et sorties se feront sur les entrées/sorties standard (`stdin` et `stdout`) par des lectures et écritures de caractères.

- **Q2 (2 point) :** Proposer une traduction en langage C de chacune des instructions suivantes du langage BF.

Caractère	Taille possible de la réponse	Traduction en C
>	moins de 20 caractères	
<	moins de 20 caractères	
+	moins de 20 caractères	
–	moins de 20 caractères	
.	moins de 25 caractères	
,	moins de 50 caractères	

- **Q2 (2 points)** : Proposer une implémentation de la fonction suivante

```
void bf_eval(char* code);
```

qui, considérant la chaîne de caractères `code` comme un programme *BF simplifié*, l'exécute. Les réponses à la précédente question peuvent vous aider ! **On considérera que le code fourni est valide. Vous n'avez donc pas à gérer les cas de code mal formé.**

- **Q3 (1 points) :** Proposer une implémentation de la fonction suivante

```
void bf_file(char* filename);
```

qui lit un programme *BF simplifié* depuis un fichier dont le chemin d'accès est représenté par la variable `filename`. Par simplicité, on considérera que le code *BF simplifié* est fourni dans la première ligne du fichier uniquement. **Rien ne vous empêche de répondre à cette question sans fournir une réponse à la précédente et en considérant la question 2 comme résolue.**