

LO22 : Examen Final

24 Juin 2004 (2h)

Documents de cours, TD et TP autorisés

Pour faciliter la lecture de vos algorithmes (1 point sera accordé à la lisibilité), vous devez :

- placer des commentaires explicatifs dans votre code ;
- choisir des noms de variables "parlantes", c'est-à-dire différentes des habituels `toto`, `titi`, ... ;
- respecter une indentation lisible (indentation \equiv décalage à droite des blocs de codes)
- écrire modulairement en définissant des fonctions et/ou des bibliothèques chaque fois que possible.

Exercice 1: Et le C fût

1. Que représente le mot clé `int` ? Quelle caractéristique le distingue des autres ?
2. Expliquez toutes les sémantiques que l'on peut donner à l'instruction `char* a`. En d'autres mots : qu'est-ce que `a` ?
3. Écrivez la fonction `char* strdup(char*)`. Cette fonction duplique en mémoire son paramètre et retourne le résultat. Vous ne vous attacherez pas à la justesse du code C mais à la séquence algorithmique.
4. Définissez le tas, la pile et le segment de code.

5. Soit les types canoniques du C et la structure suivante :

```
struct personne {  
    struct S* pere ;  
    short age ;  
};
```

Donnez le typage correspondant à chacune des expressions suivantes.

Exemple : si `z` est un pointeur sur un caractère, alors la réponse attendue est `char* z`.

- (a) `a` est matrice cubique d'entiers longs non signés ;
- (b) `b` est un tableau de chaînes de caractères ;
- (c) `c` est un pointeur sur l'âge de l'arrière-grand père d'une personne ;
- (d) `d` est une variable capable de stocker tous les caractères de la table ASCII non-étendue (première moitié de la table ASCII étendue ou complète).

Exercice 2: Choix d'algorithmes

Soit les deux algorithmes suivantes :

<pre>#define abs(x) ((x>0)?x:-x) long algo_1(short x,short y){ short i; long resultat=0; for(i=0;i<abs(y);i++) resultat += x; if (y < 0) { resultat = -resultat; } return resultat; }</pre>	<pre>long algo_2(short x,short y) { long resultat = 0; while (x > 0) { if (x % 2 == 0) { y += y; x /= 2; } else { resultat += y; x--; } } return resultat; }</pre>
---	---

Expliquez en quelques mots le rôle de ces deux algorithmes. Lequel choisiriez-vous ? Pourquoi ?

Exercice 3: Gestion mémoire

Les fonctions d'accès à la mémoire sont gourmandes. Pour améliorer les performances de votre programme qui manipule beaucoup de tableaux dynamiques, vous décidez de ne plus réallouer complètement les tableaux à chaque fois que vous voulez y ajouter un élément. Vous décidez que l'allocation mémoire ne se produira que tous les 10 ajouts. Par exemple, la figure illustre un tableau contenant 2 éléments. On constate qu'il y a 10 zones mémoires allouées mais que seules les deux premières sont initialisées. Après avoir réalisé 12 ajouts, on obtient le tableau illustré par la figure . On constate qu'un second bloc de 10 éléments a été alloué pour pouvoir stocker les caractères qui ne pouvaient pas entrer dans le premier bloc.

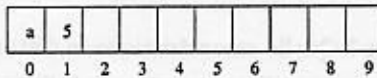


FIG. 1 – tableau de 2 éléments

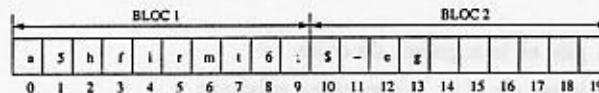


FIG. 2 – tableau de 14 éléments

On supposera que les fonctions suivantes sont déjà écrites :

- `boolean estTabVide(TABLEAU* t)` qui indique si un tableau est vide.
- `unsigned long tailleTab(TABLEAU* t)` qui retourne la taille d'un tableau.
- `char getTab(TABLEAU* t, unsigned long i)` qui retourne le caractère se trouvant à l'indice `i`.

1. Proposez le type `TABLEAU` permettant d'implanter un tableau de caractères avec buffer.
2. Proposez le prototype et écrivez la fonction `initTab` qui initialise le tableau.
3. Proposez le prototype et écrivez la fonction `destruitTab` qui détruit définitivement un tableau.
4. Proposez le prototype et écrivez la fonction `setTab` qui ajoute un élément dans un tableau en ajoutant éventuellement un nouveau bloc.

Exercice 4: Gestion d'un utilisateur

Soit un fichier contenant des instructions BASH respectant la syntaxe suivante :

`nom_commande paramètre_1 paramètre_2 ... paramètre_n`

Les instructions sont séparées par un caractère ; (point-virgule).

1. Écrivez la fonction `char*** litFichier(FILE* f)` (notez les `***` représentant un tableau de tableaux de mots) qui lit l'ensemble des mots du fichier passé en paramètre. Attention, le point virgule est un séparateur de mots. Vous pouvez écrire d'autres fonctions pour avoir une conception plus modulaire.
2. Écrivez la fonction `boolean executeInstruction(char** instruction)` exécutant l'instruction composée de mots qui sont passés en paramètre. Cette fonction retourne une valeur indiquant si l'exécution s'est bien passée. Vous utiliserez la fonction C `int system(char* command)` qui permet d'exécuter la commande BASH donnée en paramètre à partir d'un programme C (la valeur retournée par `system` est identique à celle de la commande BASH).