Université BORDEAUX 1

Année 2003-04
Session de mai 2004

GM :	MIAS 3'	UE :	302 IP
ETAPE :	INF02	UE :	INF101
ETAPE :	MATH2	UE :	INF101

Épreuve de : Initiation à la programmation

Date : 13 Mai 2004
Durée : 1 heure 30

Aucun document autorisé
Épreuve de O. Baudon et S. Chaumette

Première partie - Question de cours

1° Que fait la fonction `mystere` suivante :

```
void
mystere(int f(int), int *t, int n)
{
  int i;

  for (i = 0; i < n; i++)
    t[i] = f(t[i]);
}
```

2° Ecrire une fonction `void incremente_tableau(int *t, int n)` qui augmente de 1 les n premiers éléments d'un tableau d'entiers en utilisant la fonction `mystere` de la question précédente et la fonction `plus_un(int i)` décrite ci-dessous. *Remarque : la fonction* `incremente_tableau` *ne comporte qu'une seule instruction.*

```
int
plus_un(int i)
{
  return i+1;
}
```

Deuxième partie

Considérons le programme C suivant contenu dans le fichier `mystere.c` :

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

int
calcul(char c, char * s)
{
  int resultat = 0;
```

```
  int i;

  i = 0;
  while (s[i] != 0)
    {
      if (s[i] == c)
        resultat++;
      i++;
    }
  return resultat;
}

int
main(int argc, char *argv[])
{
  printf("%d\n", calcul(argv[1][0], argv[2]));

  return EXIT_SUCCESS;
}
```

3° `mystere` étant le nom de l'exécutable associé à ce programme, quelle est la valeur affichée par la commande suivante :

```
$ ./mystere a abcadfab
```

Réécrire le programme précédent en

4° - remplaçant dans la fonction `calcul` la boucle `while` par une boucle `for`. Que fait la fonction `calcul` ?

5° - rajoutant une fonction `usage`, de prototype `void usage(char *nom_commande)` qui affiche un message explicitant l'utilisation correcte de la commande.
Exemples d'utilisation de la commande `mystere` *:*

```
$ ./mystere
Usage : ./mystere <caractere> <chaine>

$ ./mystere b abcadfab
2
```

6° On souhaite décomposer ce programme en plusieurs fichiers de telle sorte que la définition de la fonction `calcul` soit dans un fichier séparé.
Indiquer les modifications à apporter au programme précédent et écrire les nouveaux fichiers.
Le `Makefile` *n'est pas demandé. Les parties du code inchangées pourront être remplacées par trois points quand cela ne nuit pas à la compréhension de la solution. Par exemple :*

```
int
calcul(...)
{
...
}
```

Troisième partie

On considère le module `ens` composé de deux fichiers `ens.h` et `ens.c`, servant à stocker des entiers dans un ensemble. Un même entier peut être présent plusieurs fois. Le code des deux fichiers est donné en annexe.

Pour répondre aux questions suivantes, ne réécrivez pas tout le code, mais donner uniquement les modifications à apporter !

7° Quel est l'état des tableaux `ens` et `vide` après l'exécution de la séquence d'instructions suivante.

```
ens_initialiser();
ens_ajouter(1);
ens_ajouter(2);
ens_ajouter(3);
ens_supprimer(2);
ens_ajouter(1);
```

Utiliser deux tableaux de cases contigües pour répondre à la question, en suivant le modèle ci-dessous, et en laissant vide les emplacements non initialisés.

	0	1	2	3	4	5	6	7	8	9
`ens`										
`vide`										

8° Rajouter la fonction `int ens_plein()` qui teste si l'ensemble est plein. *Remarque : l'ensemble est plein si et seulement si le tableau* `vide` *ne contient que des zéros.*

9° Rajouter la fonction `int ens_present(int element)` qui teste la présence d'un élément dans l'ensemble.

10° Rajouter la fonction `int ens_cardinal()` qui renvoie le nombre d'éléments présents (un même entier est compté autant de fois qu'il est présent).

11° Modifier la fonction `void ens_ajouter(int element)` pour qu'elle provoque une erreur lorsque l'on cherche à rajouter un élément déja présent.

12° Que faut-il modifier dans le module `ens` pour pouvoir insérer un nombre quelconque d'éléments, la seule limitation étant la capacité en mémoire de l'ordinateur ?

On pourra utiliser pour cela les fonctions
`void *malloc(size_t t)`
qui renvoie l'adresse d'un zone mémoire de taille `t` octets allouée dynamiquement,
`void *realloc(void *p, size_t t)`
qui renvoie l'adresse d'une zone mémoire de taille `t` allouée dynamiquement et initialisée par les `t` premiers octets situés à l'adresse `p`,
`void free(void *p)`
qui libère la zone mémoire située à l'adresse `p` et allouée dynamiquement par `malloc` ou `realloc`.

On rappelle que `malloc` et `realloc` renvoient la valeur `NULL` si l'allocation n'a pas eu lieu, et qu'elles sont déclarées dans le fichier d'entête `stdlib.h`. `size_t` est un type d'entiers positifs défini dans `stdlib.h`.