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

Corrigé

Première partie - Question de cours

1° Que fait la fonction `mystere` :
La fonction `mystere` applique une fonction `f` aux `n` premiers éléments d'un tableau d'entiers `t`. Le prototype de `f` doit être `int f(int)`.

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)`.
Solution :

```
void incremente_tableau(int *t, int n)
{
  mystere(plus_un, t, n);
}
```

Deuxième partie

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

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

Réponse :
`3`

Réécrire le programme précédent en

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

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

```
    for (i = 0; s[i] != '\0'; i++)
      {
        if (s[i] == c)
          resultat++;
      }
    return resultat;
  }
```

La fonction `calcul(char c, char *s)` renvoie le nombre d'occurences du caractère `c` dans la chaîne de caractères `s`.

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

Solution :

```
void
usage(char * nom_commande)
{
  fprintf(stderr, "Usage : %s <caractere> <chaine>\n", nom_commande);
}
```

Il faut de plus rajouter au début de la fonction `main` l'instruction

```
if (argc != 3)
  {
    usage(argv[0]);
    return EXIT_FAILURE;
  }
```

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.

Solution :

dans un fichier nommé `calcul.h`

```
#ifndef CALCUL_H
#define CALCUL_H

int
calcul(char c, char * s);

#endif
```

dans un fichier nommé `calcul.c`

```
#include "calcul.h"

int
calcul(char c, char * s)
{
  ...
}
```

dans le fichier `mystere.c`

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

```
int
main(int argc, char *argv[])
{
  ...
}
```

Troisième partie

7º 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);
```

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

8º Rajouter la fonction `int ens_plein(void)` qui teste si l'ensemble est plein.

Solution :

```
int
ens_plein(void)
{
  return premier_vide() == TAILLE;
}
```

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

Solution :

```
int
ens_present(int element)
{
  return premiere_occurence(element) < TAILLE;
}
```

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

Solution :

```
int
ens_cardinal(void)
{
  int i;
  int cardinal = 0;

  for (i = 0; i < TAILLE; i++)
    {
      if (!vide[i])
        cardinal++;
    }
  return cardinal;
}
```

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.

Solution :

Il suffit de rajouter l'instruction

```
assert(! ens_present(element));
```

au début de la fonction `ens_ajouter(int element)`.

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 ?

Solution :

Il faut tenir compte du fait que l'ensemble ne sera plus jamais plein, du point de vue de l'utilisateur, et augmenter sa taille en cas de besoin. Dans la solution proposée, on choisit de ne jamais diminuer cette taille, et de la doubler quand on souhaite l'augmenter.

Les modifications à apporter à `ens.h` **sont**

– **enlever dans le commentaire de** `ens_ajouter(int element)` ***Provoque une erreur si l'ensemble est plein*****,**

– **supprimer la déclaration de la fonction** `ens_plein(void)`

Dans la nouvelle version de `ens.c`**, toutes les occurences de** `TAILLE` **sont remplacées par** `taille`**, une nouvelle variable statique globale. Les fonctions dont c'est la seule modification ne sont pas récrites entièrement. Les autres sont données dans leur intégralité.**

La définition de la fonction `ens_plein(void)` **est supprimée.**

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

#include "ens.h"

#define TAILLE_INITIALE 10

static int taille = TAILLE_INITIALE;

/* contient les éléments de l'ensemble. La valeur de ens[i] n'est valide
   que si vide[i] est à 0. */
static int *ens = NULL;

/* précise si l'indice correspondant de ens est disponible
   ou non pour accueillir un nouvel élément. */
static int *vide = NULL;

void
ens_initialiser(void)
{
  int i;

  ens = malloc(taille *sizeof(int));
  vide = malloc(taille *sizeof(int));
  for (i = 0; i < taille; i++)
    vide[i] = 1;
}
```

```c
static int
premier_vide() {...}

static void
ajuster()
{
  int i;

  taille *= 2;
  ens = realloc(ens, taille * sizeof(int));
  vide = realloc(vide, taille * sizeof(int));
  for (i = taille/2; i < taille; i++)
    vide[i] = 1;
}

void
ens_ajouter(int element)
{
  int i = premier_vide();
  if (i == taille)
    ajuster();
  ens[i] = element;
  vide[i] = 0;
}

static int
premiere_occurence(int element) {...}

void
ens_supprimer(int element) {...}

int
ens_present(int element) {...}

int
ens_cardinal() {...}
```