Durée 2 h Documents non autorisés

Sujet d'examen 1

Pratique du C

Mai 2015

Introduction

Écrivez lisiblement et n'hésitez pas à commenter votre code en langage C. Vous ne pouvez utiliser que les fonctions C dont le prototype est donné dans l'énoncé et celles dont vous donnez la définition dans vos copies.

Les sections sont indépendantes ; lisez l'énoncé complet avant de commencer à le résoudre.

1 Quizz

1. La suite de Mallows $(a_j)_{j\in\mathbb{N}}$ est définie par la récurrence :

$$a_1 = a_2 = 1, \quad a_n = a_{a_{n-2}} + a_{n-a_{n-2}}.$$

Donnez la définition d'une fonction C qui prend en argument un entier n et implante récursivement cette suite ; elle retourne le $n$ième terme de cette suite.

Correction. Cette suite s'implante difficilement séquentiellement et est donc un bon exemple pour une implantation récursive :

```
int
Mallows
(unsigned int n)
{
  if(n<3)
  return 1 ;
  return Mallows(Mallows(n-2))+Mallows(n-Mallows(n-2)) ;
}
```

2. Qu'affiche le programme suivant ?

```
int
main
(void)
{
  int tab[5] = {1,2,3,4,5};
  int *p, *q, i;

  i = 2; tab[i++] = 0;
  p = tab; q = p + i - 1;
  *(q+1) = tab[i-1] + (q-p);

  for(i=0;i<5;i++)
     printf("%d ",tab[i]);
```

```
    return 0 ;
}
```

Correction. Ce code affiche

```
1 2 0 2 5
```

2 Recherche du pgcd de deux entiers positifs sans utiliser l'algorithme d'Euclide

Dans cet exercice, vous pouvez utiliser les fonctions `malloc` et `free` de la librairie standard.

1. Donner une structure de donnée de type `cell_t` permettant de construire une liste chaînée permettant de stocker des entiers positifs.
2. Donner la définition de la fonction de prototype :

   `cell_t * liste_des_diviseurs(unsigned int);`

   qui retourne la liste chaînée des entiers divisant le paramètre.
3. Donner la définition de la fonction de prototype :

   `cell_t * elements_communs(cell_t *, cell_t *);`

   qui retourne une liste composée des éléments communs aux listes passées en paramètres. Cette fonction détruit les listes passées en paramètres.
4. Donner la définition de la fonction de prototype :

   `unsigned int plus_grand_element(cell_t *) ;`

   qui retourne le plus grand élément de la liste passée en paramètre et qui détruit cette dernière.

3 Une implantation sommaire de la fonction scanf

Dans cette section, on se propose d'implanter la fonction `scanf` dans une architecture dans laquelle le passage des paramètres se fait par une pile (cf. fin de la section pour une description de l'architecture).

Par ailleurs, on suppose ne disposer que d'une seule fonction externe dont le prototype est `int getchar(void);` et qui lit depuis l'entrée standard un caractère dont le code ASCII est retourné après conversion en un entier ; en cas d'erreur ou si l'entrée est un fichier dont on a atteint la fin, cette fonction retourne l'entier défini par la macro `EOF`.

Par exemple, pour extraire de l'entrée standard un caractère et le stocker dans une variable de type entier, on peut utiliser le code suivant :

```
#include<stdio.h>

int main(){
  unsigned char c ;
  c = (unsigned char) getchar() ;
  return 0 ;
}
```

L'entrée standard est considérée comme une suite finie ou vide de n octets $o_1, \ldots, o_n$; si elle est vide la fonction `getchar` attend qu'un octet lui soit transmis, sinon après un appel à `getchar`, l'entrée standard est constituée de la suite fini de $n-1$ octets $o_2, \ldots, o_n$.

Objectif. L'objectif est d'implanter la fonction de prototype : `int mscanf(const char *format, ...)` permettant de saisir depuis l'entrée standard :
- des caractères ASCII classique de type `char`.
- des entiers machines positifs dans les bases décimale et binaire.

Cette fonction retourne 0 en cas de problème (mauvaise directive ou problème lors de la saisie i.e. le codage de la suite de caractères de l'entrée standard ne correspond pas aux conventions, cf. la suite) et 1 sinon.

Caractéristique de la fonction mscanf. Cette fonction a un paramètre obligatoire et un nombre variable de paramètres complémentaires.

Le paramètre obligatoire est constitué par une chaîne de caractères composée d'une ou plusieurs directives. Une directive commence par le caractère % et peut être de plusieurs formes :
- la directive %c indique que l'on souhaite saisir un caractère ;
- la directive %s indique que l'on souhaite saisir une chaîne de caractères ;
- la directive %d indique que l'on souhaite saisir un entier positif en base décimale ;
- la directive %b indique que l'on souhaite saisir un entier positif en base binaire.

Convention pour la saisie par mscanf. On convient qu' :
- un entier se termine par tout caractère non numérique ;
- une chaîne de caractères (de type `char *`) se termine par un caractère de code 0 ou `EOF`.

Questions

1. De quels types peuvent être les paramètres complémentaires de la fonction `mscanf` ?
2. Donnez la définition d'une fonction `int ScanString(char *s,unsigned int stringsize)` qui saisit depuis l'entrée standard une chaîne de caractères d'au plus `stringsize` caractères et les place dans l'espace mémoire pointé par `s` et retourne 1 en cas de problème et 0 sinon. On convient que :
   - les opérations d'allocation et de désallocation de `s` incombent à l'utilisateur et non pas à la fonction `ScanString`.
   - la saisie d'une chaîne de caractères s'interrompt par la saisie du caractère de code ASCII 0 ou `EOF` (ce dernier est remplacé par 0 dans la chaîne de caractères).
   - un problème survient lorsque la fonction saisie `stringsize` caractères sans rencontrer de caractère de code ASCII 0 ou `EOF` et dans ce cas le dernier caractère de l'espace mémoire associé à `s` doit être de code ASCII 0.
3. Donnez la définition d'une fonction `int ScanInt(int b, unsigned int *res)` qui saisit depuis l'entrée standard une chaîne de caractères représentant un entier positif exprimé dans la base $b \in \{2, 10\}$ et le transmet par l'intermédiaire de son second paramètre `res`. Cette fonction arrête la saisie au premier caractère de l'entrée standard n'étant pas un chiffre de la base `b` ; ce caractère est converti en entier et retourné.
4. Donnez la définition de la fonction `mscanf`.

Modèle de passage de paramètres aux fonctions. Rappelons que les paramètres d'une fonction sont passés par la pile. Cette pile est composée de cellules dont la taille en octet correspond au type `int`, elle croit vers les adresses décroissantes et elle a la structure suivante :

0000	$\cdots$
	seconde variable locale
	première variable locale
	ancien pointeur de contexte
	adresse de retour
	premier paramètre
	second paramètre
FFFF	$\cdots$

Ainsi si `int foo` est la première variable locale automatique définie dans la fonction appelée et que `ptr` est un pointeur sur la cellule correspondante, `ptr+1` pointe sur l'ancien pointeur de contexte.

De plus, lors de l'appel d'une fonction, ses paramètres sont empilés du dernier au premier et on note qu'un pointeur occupe une cellule dans tous les cas.

Correction.

1. Ces paramètres doivent être des pointeurs de type `int*` ou `char*` selon le format.
2. ```
   int
   ScanString
   (char *s,unsigned int stringsize)
   {
     stringsize-- ; /* pour pouvoir placer un 0 en fin */
      while(1) ;
       {
   if (!stringsize)
   return 1 ;

        *s = (char) getchar() ;
        switch(*s)
        {
        case EOF  :  ;
        case '\n' : *s = 0 ;
        case 0    : break ;
        } ;
        s++ ;
        stringsize-- ;
       }
       return 0 ;
   }
   ```
3. ```
   int
   ScanInt
   (int b, unsigned int *res)
   {
   int c ;
   *res = 0;
   while( ((c=getchar)!=EOF || c!='\n') && ( (c=c-'0')>-1 && c <=b))
   {
     *res *=b;
     *res += c ;
   }
      return c+'0' ;
   }
   ```
4. ```
   int
   mscanf
   (const char *str, ...)
   {
      /* on positionne un pointeur sur le premi\‘ere
      param\‘etre */
      void *tmp  = &((int *) str) ;
      int message = 1 ;
      tmp++;
      while(*str)
      {
   ```
   ```

```
      str ++ ;
      /* on cherche le premier format */
      if(*(str-1)!='%')
         continue ;

      switch(*str)
      {
       case 'c' : message *= ScanString((char *) *(tmp++) , 1) ; break ;
       case 's' : message *= ScanString((char *) *(tmp++) , 1 << 32) ; break ;
       case 'b' : message *= ScanInt(2, *(tmp++) ) ; break ;
       case 'd' : message *= ScanInt(10, *(tmp++) ) ; break ;
       default : continue
      }
      if(!message)
         break ;
   }
   return message ;
}
```