Durée 1 h 30 Documents autorisés

Sujet d'examen 1

Pratique du C

Novembre 2009

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. On considère les déclarations suivantes :

   ```
   int a[8] = {5, 15, 34, 54, 14, 2, 52, 72};
   int *p = &a[1], *q = &a[5];
   ```

   (a) Quelle est la valeur de `*(p+3)` ?

   (b) Quelle est la valeur de `q-p` ?

   (c) La condition `p<q` est-elle vraie ou fausse ?

   (d) La condition `*p<*q` est-elle vraie ou fausse ?

   (e) Réecrivez le code `int *p = &a[1], *q = &a[5];` sans utiliser de crochets.

   Correction.

   (a) 14

   (b) 4

   (c) c'est vrai

   (d) c'est faux

   (e)
   ```
   int *p = a+1;
   int *q = a+5;
   ```

2. Qu'affiche le programme suivant :

   ```
   #include<stdio.h>
   #define PRECEDENTE 1
   #define ACTUELLE 2
   #define PROCHAINE 3
   int
   main
   (void)
   {
     int annee = ACTUELLE;
     switch (annee)
   ```

```
  {
   case PRECEDENTE: annee = 2007;
   case ACTUELLE: annee = 2008;
   case PROCHAINE: annee = 2009;
   default:annee = 2010;
  }
  printf("Bonne annee %d\n", annee);
  return 0;
}
```

Correction. Ce programme affiche

```
% ./a.out
Bonne annee 2010
%
```

En effet, aucun `break` n'est utilisé dans le `switch`. Donc, toutes les instructions après `ACTUELLE` sont exécutées.

3. Considérons les fonctions :

```
int foo(int a) { | int bar(int *b) { | int foobar(int *c) {     | int barfoo(int d) {
    a=a*2 ;      |     *b=(*b)*3 ;   |     return foo(bar(c)) ; |    int t ;
    return a+3 ; |     return *b+7 ; | }                        |    t = foo(d);
}                | }                 |                          |    return bar(&t) ;
                 |                   |                          |  }
```

Donnez la valeur des différentes variables définies ci-dessous après exécution du code suivant :

```
int a,b,c,d,ra,rb,rc,rd ;
a = 10; b = 20; c = 50; d = 70;
ra = foo(a); rb = bar(&b); rc = foobar(&c); rd = barfoo(d);
```

Correction. a = 10, b = 60, c = 150, d = 70, ra = 23, rb = 67, rc = 317, rd = 436

2 Recherche par fonction de hachage

On se propose de coder un carnet d'adresse par une table (d'au plus 4×131 entrées) et pour ce faire on désire construire un tableau dont chaque élément (aussi appelé cellule) donne accès à trois champs :

- un nom (d'au plus 30 caractères) que l'on considère être la clef de la table ;
- un numéro de téléphone (codé par un entier non signé) ;
- un entier initialisé à -1 et dont l'usage sera explicité par la suite.

Le hachage est une méthode de recherche dans une table. On utilise une fonction h de l'ensemble des clefs dans un intervalle d'indice. Pour une clef x, l'entier $h(x)$ est l'indice de la cellule qui contient x dans la table.

Pour construire h, on choisit un nombre premier n (diviseur de la taille de la table) et un entier positif b (puissance de 2 pour des raisons arithmétiques) ; dans cet exercice $n = 131$ et $b = 128$. Ces entiers permettent de construire une fonction h qui à partir d'une chaîne x de longueur ℓ retourne un entier :

$$h \ : \quad x \to \big(x[0] + x[1] \cdot b + \cdots + x[\ell - 1] \cdot b^{\ell-1}\big) \bmod n. \tag{1}$$

Pour toute clef x, la fonction h donne une entrée possible dans les n premières cellules la table. Deux cas se présentent alors :

- soit la clef contenu dans la cellule d'indice $h(x)$ est bien x ;

– soit ce n'est pas le cas. Cette situation est une *collision* : il existe une autre clef x' telle que $h(x') = h(x)$. Dans ce cas, on utilise le troisième champs pour indiquer l'indice i_1 de la cellule contenant la clef x'. Si on ne trouve pas la valeur cherchée à cet indice, alors une nouvelle collision est rencontrée et on peut utiliser le troisième champs de la cellule d'indice i_1 pour pointer sur une autre cellule, etc.

La figure suivante donne un exemple de cette structure de donnée :

indice	nom	tél	collision
0	Sandrine	2345	-1
1			-2
2	Marianne	3556	-1
3			-2
4	Béatrice	4234	6
5			-2
6	Anne	4333	430
	...		
430	Laëtitia	3205	-1

La valeur -1 indique qu'il n'y a pas de collision et la valeur -2 que la cellule est vide.

Questions.

1. Donnez la déclaration d'un type permettant de représenter une cellule et la définition du tableau global codant la table.
2. Donnez le code d'une fonction de hachage h qui prend en entrée une chaîne de caractères et qui retourne un entier donné par la formule (1).
3. Donnez le code d'une fonction de recherche qui prend en argument une chaîne de caractères et qui retourne un numéro de téléphone.
4. Donnez le code d'une fonction d'insertion qui prend en argument une chaîne de caractères et un numéro de téléphone et qui insère ces informations dans la table.

Correction.

1. ```
   #define N 131
   #define B 128
   #define TABSIZE 4*N
   struct cell_m
   {
   char nom[30] ;
   unsigned int num_tel ;
   int collision ;
   } ;

   struct cell_m table[TABSIZE] ;
   ```
2. ```
   int
   h
   (char *nom)
   {
   int res = 0 ;
   int pow = 1 ;
   int ind = 0 ;
   while(nom[ind])
   {
     res +=nom[ind]* pow ;
     pow *= b ;
   ```

```
  ind++ ;
}
return res % N
}
```

3. ```
   /* on se donne une fonction auxiliare */
   int
   strcmp
   (char *s1,char *s2)
   {
       char c1, c2;
       char v;

       do
       {
           c1 = *s1++;
           c2 = *s2++;
          v = c1 - c2;
       } while ((v == 0) && (c1 != '\0'));

       return v;

   }

   #define TELINCONNU -3

   int
   recherche
   (char * nom)
   {
     int indice ;
     indice = h(nom) ;

     while(1)
     {
       if (!strcmp(nom,table[indice].nom)
         return table[indice].num_tel ;
       if (table[indice].collision=-2 ||
           table[indice].collision=-1)
         return TELINCONNU
       indice = table[indice].collision ;
     }

     return TELINCONNU ;

   }
   ```

4. ```
   void
   insertion
   (char *nom, unsigned int tel)
   {
      static int nextcollisionplace ;
      int indice,nextindice,i ;
      indice = h(nom) ;
      nextindice = indice ;
   ```

```
        while(1)
        {
          if (indice>TABSIZE)
             return ;

          if(table[indice].collision==-2)
          {
            for(i=0;nom[i];i++)
               table[indice].nom[i]=nom[i] ;

            table[indice].num_tel = tel ;
            table[indice].collision = - 1 ;
            return ;
          }

         nextindice = N + nextcollisionplace++ ;

         if(table[indice].collision==-1)
            table[indice].collision=nextindice ;

         indice = nextindice ;
        }
        return ;
    }
```

3 Exercice sur les classes d'allocations

Considérons le fichier source `aux.c` suivant :

```
#include <stdio.h>
int x;
static int y;
void affectation(void) { x = 2; y = 3; }
void choix(int a) {
    int x = 1;
    static int y;
    if (a == 0)
        y = 0;
    else y += x;
    printf("choix: %d %d\n", x, y);
}

void proc1(void) { printf("proc1: %d %d\n", x, y); }
```

et le fichier source `main.c`

```
#include <stdio.h>
extern int x;
static int y;

void proc2(void)
{ printf("proc2: %d %d\n", x, y); }
```

```
int main(void) {
    choix(0);
    affectation();
    choix(1);
    proc1();
    proc2();
    return 0 ;
}
```

Questions :

1. Que manque-t'il au fichier source `main.c` en vue de la production d'un code objet ?
2. Donnez les commandes shell permettant d'obtenir un fichier exécutable à partir du fichier source `main.c`.
3. Donnez la sortie produite par l'exécution du fichier ainsi obtenu.

Correction.

1. il manque les déclarations des fonctions définies dans `aux.c`

   ```
   extern void affectation(void) ;
   extern void choix(int) ;
   extern void proc1(void) ;
   ```

2. ```
   $ gcc -c aux.c
   $ gcc -c main.c
   $ gcc -o executable aux.o main.o
   ```

3. ```
   choix: 1 0
   choix: 1 1
   proc1: 2 3
   proc2: 2 0
   ```

4 Exercice sur le passage de paramètres

Dans cet exercice, vous allez écrire trois programmes en C réalisant la même opération — une addition de deux entiers par un appel de fonction — qui diffèrent par la méthode de passage de paramètres.

1. Définissez trois variables globales `i`, `j` et `k` de type entier. Écrire une fonction `globadd()` qui fait l'addition de `i` et `j` et stocke le résultat dans `k`. Écrire la fonction `int main()` qui réalise la saisie des variables `i` et `j`, fait appel à la fonction d'addition et retourne le résultat contenu dans `k`.
2. Même exercice que le précédent mais en utilisant le passage de paramètres et le retour de fonction. Les trois variables `i`, `j` et `k` de type entier sont déclarées localement dans la fonction `main()`. La fonction d'addition `add` est une fonction retournant un entier. Ele accepte deux paramètres entiers (`p1` et `p2`) et retourne la somme de ces deux paramètres. La fonction `main()` permet la saisie des deux variables locales `i` et `j` et utilise la fonction d'addition en récupérant le résultat de cette fonction dans la variable locale `k`. Elle retourne le contenu de `k`.
3. Même exercice que le précédent mais en utilisant le passage de paramètres et un pointeur pour modifier une variable dans la fonction appelante. Les trois variables `i`, `j` et `k` de type entier sont declarées localement dans la fonction `main()`. La fonction d'addition `ptadd()` est une fonction qui ne retourne rien. Elle accepte trois paramètres : deux entiers (`p1` et `p2`) et un paramètre de type pointeur vers un entier qui sert à affecter avec la somme de ces deux premiers paramètres, la variable dont la fonction appelante passe l'adresse. La fonction `main()` saisit les deux variables locales `i` et `j` et appelle la fonction d'addition passant l'adresse de la variable locale `k`. Elle retourne le contenu de `k` après l'appel de fonction.

Correction.

1. ```
   int i,j,k ;
   void
   globaladd
   (void)
   {
     k=i+j ;
   }
   ```
2. ```
   #include <stdio.h>

   int
   add
   (int p1,int p2)
   {
     return p1+p2 ;
   }

   int
   main
   (void)
   {
     int i,j,k ;
     scanf("%d",&i) ;
     scanf("%d",&j) ;
     k=add(i,j) ;
     return k ;
   }
   ```
3. ```
   #include <stdio.h>

   void
   ptadd
   (int p1,int p2, int *res)
   {
     *res = p1+p2 ;
     return  ;
   }

   int
   main
   (void)
   {
     int i,j,k ;
     scanf("%d",&i) ;
     scanf("%d",&j) ;
     add(i,j, &k) ;
     return k ;
   }
   ```
   ```