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. Quel est le contenu du tableau `tab` défini ci-dessous après l'exécution des instructions suivantes :

```
#define N 10
int tab[N] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
int *p = &tab[0], *q = &tab[N-1], temp;
while (p < q) {
   temp = *p;
   *(p++) = *q;
   *(q--) = temp;
}
```

2. Considérons le code suivant :

```
#include<stdio.h>
int
main
(void)
{
   int s=0;
   while(s++<10)
   {
      if(s<4 && s<9)
         continue;
      printf(" %d ",s);
   }
   return s ;
}
```

   (a) Quel est l'affichage produit ?

   (b) Quelle est la valeur de retour ?

Correction. Ce code affiche

```
% ./a.out
 4  5  6  7  8  9  10
```

et retourne 11.

3. Considérons le code C suivant :

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
char* remplace(char c, int i) {
    /* C'est ici que l'on veut ins\'erer la ligne */
    str[i] = c;
    return str;
}
int main(void){
    char* chaine = remplace(' ', 2);
    chaine = remplace('n', 0); chaine = remplace('o', 1);
    printf("%s\n", chaine);
    return 0;
}
```

On souhaite que ce code affiche la chaîne de caractères

```
cassandre % gcc Quizz_13.c && ./a.out
no fun
```

Pour ce faire, on considère 4 possibilités pour chacune des lignes suivantes que l'on suppose écrites à la place du commentaire `/* C'est ici que l'on veut ins\'erer la ligne */`.

(a) `char str[] = { 'C',' ',' ','f','u','n' } ;`

(b) `char str[] = "C  fun" ;`

(c) `char *str = (char *) malloc(7*sizeof(char)); strncpy(str, "C  fun");`

(d) `static char str[] = { 'C',' ',' ','f','u','n' } ;`.

Dans chaque cas, indiquez

(a) si le programme compile,

(b) s'il provoque une erreur ou un résultat indéterminé en cours d'exécution et

(c) le résultat de l'affichage.

Justifier vos réponses.

Remarques : La fonction

```
#include <string.h>
char *strncpy (char *dest, const char *src, int n);
```

copie la chaîne pointée par `src` (y compris le caractère `'\0'` final) dans la chaîne pointée par dest. Les deux chaînes ne doivent pas se chevaucher. La chaîne `dest` doit être assez grande pour accueillir la copie.

Dans le cas où la longueur `src` est inférieure à n, la fin de dest sera remplie avec des caractères nuls.

La fonction strncpy() renvoient un pointeur sur la chaîne destination dest.

Le programme compile dans les quatres cas.

(a) `char str[] = { 'C',' ',' ','f','u','n' } ;` str étant une variable automatique i.e. définie sur la pile, le compilateur le signale (warning) et le résultat de l'affichage est plus que loufoque : le tableau str, bien qu'associé à une adresse, n'est pas un pointeur et l'emplacement mémoire de ce tableau peut être occupé par autre chose (les variables et paramètres de l'appel à printf).

(b) C'est un peu mieux car cette fois, c'est l'adresse que l'on stocke dans `str`. Mais l'affichage produit sera `Co fun`.

(c) `char *str = (char *) malloc(7*sizeof(char)); strncpy(str, "C  fun");` Cette fois, on n'utilise plus une variable automatique (on utilise le tas) mais un autre espace mémoire est utilisé à chaque appel de la fonction `remplace`. Le résultat de l'affichage est donc verb+Co fun+.

(d) `static char str[] = "C fun";`. C'est le bon affichage. Le code compile sans provoquer de message (str est ici une variable static donc de portée locale à la procédure remplace mais n'est pas automatique i.e. définie sur la pile)

2 Makefile

On dispose d'un répertoire dont le contenu est :

```
% ls
Makefile foo.c foo.o main.c main.o
module.c module.h module.o prog
```

Le fichier `Makefile` contient le code suivant :

```
% cat Makefile
CC=gcc
CFLAGS = -Wall -ansi -pedantic
OBJS = main.o module.o foo.o

.PHONY: doit clean

prog: $(OBJS)
     $(CC) -o prog $(OBJS) -lm

main.o: main.c module.h
     $(CC) -c $(CFLAGS) main.c

clean:
    -rm *.o

doit:
     make prog
     rm -f prog
     make clean
```

Questions.

1. En supposant qu'il n'y a pas d'erreur de compilation, d'édition de liens ou d'accès aux fichiers, donner l'ensemble des commandes que le shell exécute si l'utilisateur tape l'instruction :

   `make doit`

2. Que se passe-t-il si on recommence cette instruction ?
3. Modifiez ce Makefile pour corriger le problème (`foo.c` est le seul source n'incluant pas l'entête `module.h`).

Corrections.

1. la compilation de tous les objets, la suppression de l'exécutable produit et de tous les objets.
2. Comme tous les objets ont été détruits et que `foo.o` et `module.o` ne sont pas reconstruit, la compilation de l'exécutable `prog` ne peut qu'échouer.
3. il faut imposer la production des objets `foo.o` et `module.o` sur le même modèle que celle de `main.o`.

3 Matrice creuse

La taille d'une matrice carrée creuse n'est pas connue à la compilation mais seulement lors de sa création. Les coefficients de cette matrice sont des entiers machines signés.

On représente une matrice carrée creuse sous la forme d'une liste chaînée dont chaque élément correspond à une ligne de la matrice et contient comme information l'indice de ligne et la représentation d'une ligne. Cette liste est triée suivant la valeur de l'indice de ligne.

Une ligne est représentée par une liste chaînée stockant les éléments non nuls de cette ligne. Un élément d'une ligne contient comme information le couple (indice de colonne, coefficient). Chaque liste est triée suivant la valeur de l'indice de colonne.

Questions.

1. Donnez la déclaration des types `ligne_t` et `element_t` correspondant à une ligne et à un élément de la liste chaînée représentant une ligne.
2. Donnez la déclaration du type `MatriceCreuse_t` représentant une matrice carrée creuse comme décrit ci-dessus.
3. Donnez la définition d'une fonction de prototype `void freemat(MatriceCreuse_t)` qui libère l'espace mémoire associé à la matrice creuse passée en paramètre.
4. Donnez la définition d'une fonction `void multscal(int a, MatriceCreuse_t M)` qui réalise la multiplication de `M` par le scalaire `a`. Les éléments de `M` sont directement modifiés par cette procédure. La multiplication d'une matrice par un scalaire revient à multiplier chaque élément de la matrice par ce scalaire.
5. Donnez la définition d'une fonction `MatriceCreuse_t convert(int **M, int n)` qui convertit une matrice carrée pleine `M` de taille `n` et codée par un tableau bidimentionnel en une matrice creuse et renvoie le résultat. Cette fonction réalisera toutes les allocations dynamiques nécessaires.

Indication : Vous pouvez utiliser les fonctions classiques d'allocation et de désallocation :

```
void * malloc(int);
void free(void *) ;
```

Corrections.

1. ```
   struct listecouple_m{
       unsigned int           indice ;
       int                    coeff  ;
       struct listecouple_m * next   ;
   } ;

   typedef struct couple_m couple_t

   typedef couple_t * ligne_t ;
   ```
   ```

```
struct MatriceCreuse_m{
   unsigned int taille ; /* pour savoir combien il y a de ligne */
   ligne_t ligne ;
} ;
typedef struct MatriceCreuse_m MatriceCreuse_t
```

2.
```
void freecoefflist(couple_t * cell)
{
    if(cell) return ;
    freecoefflist(cell->next) ;
    free(cell) ;
    return ;
}
void freemat(MatCreuse_t M)
{
   unsigned int lignecourante ;
   for(lignecourante=0;lignecourante<M.taille;lignecourante++)
         freecoefflist(M.ligne[lignecourante]) ;
   free(M.ligne) ;
   return ;
}
```

3.
```
void multscal(int a, MatriceCreuse_t M)
{
   unsigned int lignecourante ;
   couple_t coeffcourant ;
   for(lignecourante=0;lignecourante<M.taille;lignecourante++)
   {
     coeffcourant = M.ligne[lignecourante] ;
     while(coeffcourant) ;
     {
       coeffcourant->coeff *= a             ;
       coeffcourant         = M.ligne->next ;
     }
   }
   return ;
}
```

4.
```
couple_t *creercellule(unsigned int indice,int coeff)
{
  couple_t *res ;
  res = (couple_t *) malloc(sizeof(couple_t)) ;
  res.indice = indice ;
  res.coeff  =  coeff ;
  res.next   =   NULL ;
  return res ;
}

MatriceCreuse_t convert(int **M, int n)
{
   MatriceCreuse_t mat ;
   unsigned int i,j ;
   couple_t *head, *tail ;

   mat.taille = n ;
   mat.ligne  = (ligne_t *) malloc(sizeof(couple_t)*n) ;
```

```
        for(i=0;i<mat.taille;i++)
        {
          head = NULL ;
          for(j=0;j<mat.taille;j++)
             if(M[i][j])
             {
               if(head==NULL)
                  head = tail = creercellule(i,M[i][j]) ;
               else
               {
                  tail->next = creercellule(i,M[i][j]) ;
                  tail = ptr->next ;
               }
             }
          mat.ligne=ptr.head ;
        }
    }
```

4 Que fait ce programme ?

Donnez la représentation schématique de i et p en mémoire avant l'exécution de la boucle. Qu'affiche le programme ? Quel est le contenu du tableau i à la fin de l'exécution ?

```
void f(int **i){
        *((*i)++) += 1 ;
}
int main(void){
        int i[5] = {0,2,4,6,8}, *p = i ;
        for(;*p!=8;)
        f(&p) ;
        printf("%d %d\n",i[0],*p);
        return 0 ;
}
```