Durée 1 h 30 Documents non autorisés

Sujet d'examen 1

Pratique du C

Mars 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. À quoi ressemblera le programme suivant après traitement par le préprocesseur :

```
/* #include<stdio.h> */
   #define N 100
   int main(void){
   #ifdef N
      printf("N vaut %d\n", N);
   #else
      printf ("N n'est pas defini\n");
   #endif
   return 0 ;
   }
```

   Pour répondre, donnez le code C résultant de ce traitement.

   Correction. Ce code affiche `N vaut 100` et ressemble a

```
  int
   main
   (void)
   {
      printf("N vaut %d\n", 100);
   return 0 ;
   }
```

   Dans la correction il faudrait faire attention :
   — à la disparition du commentaire ;
   — à la substitution de N comme constante et pas comme partie de la chaîne de caractères ;
   — au mécanisme de la conditionnelle.

2. Parmi les lignes de codes suivantes :

```
/* 1 */  enum VENT {BOREE, NOTOS, EUROS, ZEPHYR};
/* 2 */  enum Vent {BOREE, NOTOS, EUROS, ZEPHYR} VENT;
/* 3 */  typedef VENT enum {BOREE, NOTOS, EUROS, ZEPHYR};
```

```
/* 4 */  typedef enum VENT {BOREE, NOTOS, EUROS, ZEPHYR};
/* 5 */  typedef enum {BOREE, NOTOS, EUROS, ZEPHYR} VENT;
```

quelle est la bonne déclaration permettant la définition suivante :

```
VENT d;
```

Correction. C'est bien sur :

```
typedef enum {BOREE, NOTOS, EUROS, ZEPHYR} VENT;
```

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 ? Justifier.
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 Validité d'un code ISBN

La plupart des livres sont publiés avec un code les identifiant : il s'agit du code ISBN pour International Standard Book Number. Ce code est composé d'entiers compris entre 0 et 9. On utilise la lettre X pour représenter l'entier 10. De plus, des tirets sont introduit dans le code afin d'en faciliter la lecture sans pour autant avoir d'autre signification.

Seul les 9 premiers chiffres d'un code ISBN sont utilisés pour identifier le livre. Le 10ième caractère sert à contrôler la validité du code (comme la clef d'un RIB ou les deux derniers chiffres de votre numéro de sécurité sociale).

L'algorithme pour tester la validité du codeISBN est simple. On calcule à partir de ce dernier deux sommes s_1 et s_2. Le code ISBN est correct si la valeur finale de s_2 est divisible par 11.

On expose l'algorithme au travers de l'exemple du code ISBN 0-13-162959-X. Considérons tout d'abord le calcul de s_1.

chiffres du code ISBN	0	1	3	1	6	2	9	5	9	10(X)
s_1	0	1	4	5	11	13	22	27	36	46

Le calcul de s_2 est fait en sommant les sommes partielles de s_1

chiffres du code ISBN	0	1	3	1	6	2	9	5	9	10(X)
s_1	0	1	4	5	11	13	22	27	36	46
s_2	0	1	5	10	21	34	56	83	119	165

Pour finir, on constate que 165 est le produit de 15 par 11. Notre code ISBN est donc valide.

Question. Construisez une fonction C d'identificateur `IsISBNValid` qui prend en argument un tableau de caractères représentant le code ISBN (pouvant contenir des tirets) et retourne 0 si, et seulement si, ce code est correct.

```
int
IsISBNValid
(char *isbn)
{
int s1,s2,cnt_digit ;
cnt_digit=s1=s2=0 ;

while(*isbn)
{
if (*isbn >= '0' && *isbn <= '9')
{
  cnt_digit ++ ;
  s1 += *isbn -'0' ;
  s2 += s1 ;
/* incr\'ementation du pointeur isbn */
isbn ++ ;
continue ;
}
if (*isbn=='X')
{
          cnt_digit ++ ;
  s1 += 10 ;
                  s2 += s1 ;
/* incr\'ementation du pointeur isbn */
isbn ++ ;
continue ;
               }
               if (*isbn=='-')
```

```
                {
/* incr\'ementation du pointeur isbn */
isbn ++ ;
continue ;
}
return 1 ; /* erreur dans le code ISBN */
}
if(cnt_digit == 10)
return (s2 % 11 != 0) ;
return 1 ;
}
int
main
(int argc, char **argv)
{
if(argc!=2)
return 1 ;
return IsISBNValid(argv[1]) ;
}
```

4 Tri bulle générique

Tri Bulle. Le tri bulle consiste à parcourir la table à trier en intervertissant toute paire d'éléments consécutifs non ordonnés afin qu'après un parcours, le plus grand élément se retrouve à la fin de la table. On recommence cette opération avec la table considérée sans le dernier éléments.

Questions.

1. Donnez la définition de la fonction de prototype :

   `void TriBulle(int *,int) ;`

   qui tri — de manière destructive — un tableau d'entiers passé en premiers paramètre et dont la taille est passée en second paramètre.

2. Donnez la définition de la fonction de prototype :

   `void TriBulleGenerique (void *tab,int tabsize, int cellsize, int (*compar)(void*, void *)) ;`

   qui tri — de manière destructive — un tableau dont l'origine est passée en premiers paramètre, la taille en second paramètre, la taille (en octet) de chacune de ces cellules en troisième et une fonction de comparaison de cellule en dernier.

Correction. La première question n'est que la répétition de la seconde. On va traiter les deux dans l'ordre :

⟨ *** ⟩≡
  ⟨ *Tri bulle d'un tableau d'entiers* ⟩
  ⟨ *Tri bulle générique* ⟩

1. Ce tri est un grand classique.

⟨ *Tri bulle d'un tableau d'entiers* ⟩≡
```
void
TriBulle
(int *tab,int tabsize)
{
  int i ;
  int tmp ;
  for(tabsize--;tabsize;tabsize--)
    for(i=0;i<tabsize;i++)
      if(tab[i]>tab[i+1])
         {
           tmp=tab[i] ;
           tab[i] = tab[i+1] ;
           tab[i+1] = tmp ;
         }
  return ;
}
```

2. La version générique est une simple adaptation :

⟨ *Tri bulle générique* ⟩≡
```
#define CHOOSECELL(a,b,c) (((char *) a)+(b)*cellsize+(c))
void
TriBulleGenerique
(void *tab,int tabsize, int cellsize, int (*compar)(void *, void *))
{
  int i,j ;
  char tmp ;
  for(tabsize--;tabsize;tabsize--)
    for(i=0;i<tabsize;i++)
      if( compar((void *) CHOOSECELL(tab,i,0),(void *) CHOOSECELL(tab,i+1,0)))
           for(j=0;j<cellsize;j++)
           {
             tmp= * CHOOSECELL(tab,i,j) ;
             * CHOOSECELL(tab,i,j) = * CHOOSECELL(tab,i+1,j) ;
             * CHOOSECELL(tab,i+1,j) = tmp ;
           }
  return ;
}
```