Durée 3 h Documents autorisés

Sujet d'examen 1

Pratique du C

Janvier 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. Donnez l'affichage produit par l'exécution du programme suivant :

```
#include<stdio.h>
char *f0 (char *a[][3]){ return (**a); }
char *f1 (char *a[][3]){ return (*(*(a+1)+2)); }
int main (void){
  char *(*fcts[2]) (char *tableau[][3]);
  static char *tableau[][3] =
          {{"0","1","2"},
           {"3","4","5"},
           {"6","7","8"}};
  fcts[0] = &f0;
  fcts[1] = &f1;
  printf("%s\n", (**fcts)(tableau));
  printf("%s\n", (**(fcts+1))(tableau+1));
  return 0 ;
}
```

2. 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 ;
}
```

3. (a) Que retourne au shell l'exécutable produit par le code source suivant :

```
void CalculAireRectangle (int longueur, int largeur, int AireRectangle){
AireRectangle = longueur * largeur;
}

int main (void){
     int longueur = 5;
     int largeur = 4;
     int AireRectangle = 0;
     CalculAireRectangle(longueur, largeur, AireRectangle);
     return AireRectangle ;
}
```

(b) Modifier ce programme pour que sa valeur de retour corresponde bien à l'aire du rectangle définit par les longueur et largeur spécifiées.

4. 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.

2 Chiffrement de César

2.1 Échauffement

Donnez la définition des fonctions

```
int maxi_tableau(int *tab,int taille) ;
int min_tableau(float *tab,int taille) ;
```

qui retournent l'indice de la cellule contenant le maximum d'un tableau d'entiers et le minimum d'un tableau de réels.

Le principe du chiffrement de César.

Étant donné un entier positif n, ce codage consiste à remplacer chaque lettre d'un texte par la lettre qui se situe n places plus loin dans l'alphabet. Par exemple avec un décalage de 7, on obtient :

Clair	T	A	Q	U	E
Chiffré	[	H	X	\	L

Ainsi, le texte clair T (code ASCII 84) est chiffré par le texte [(code ASCII 91).

2.2 Chiffrement.

Donnez la définition de la fonction de prototype :

```
char *chiffrer_cesar(const char *clair, int decalage) ;
```

qui prend en paramètre une chaîne de caractères non codée et retourne son chiffré (sans détruire le clair).

2.3 Déchiffrement.

Utiliser la fonction de chiffrement pour implanter l'opération inverse dans la fonction de prototype :

```
char *dechiffrer_cesar(char *chiffre, int decalage) ;
```

2.4 Premier pas vers le cassage du code.

Donnez la définition de la fonction de prototype :

```
char calcul_frequence(char *texte, float frequences_calculees[]) ;
```

qui calcule les fréquences de chaque lettre de la table des caractères ASCII (de 0 à 255) dans le texte et les stocke dans la table `frequences_calculees`. (On pourra déjà calculer l'occurrence de chaques lettre pour en déduire les fréquences d'apparitions). De plus, cette fonction doit retourner la lettre la plus fréquente.

2.5 Calcul du décalage moyen

On suppose disposer d'une table codée par le tableau `float french[255]` qui contient la probabilité d'occurrences des lettres de la table ASCII en français. Par exemple, `french['e']` contient $0,1486$, `french[' ']` contient $0,1835$ et `french['u']` contient $0,0506$. Ainsi, pour un texte de 1000 caractères, on s'attend à ce qu'il y ait 148 occurrences de la lettre e.

Écrire une fonction de prototype

```
int decalage(char *texte, float *langue) ;
```

qui analyse le contenu du fichier crypté et retourne une proposition de décalage employé par le chiffre de César. (On cherche la lettre la plus courante dans le texte, on suppose que c'est la lettre ayant le plus de probabilités d'apparaître ; on obtient ainsi un décalage probable).

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 convertion 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 entier 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 fini 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 places 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 sortie standard 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.