IUT de Montpellier, Dép. Informatique
M2101 - Archi 2 V. Poupet

Examen (2 heures)

Les documents, calculatrices, téléphones etc. ne sont pas autorisés.
Le sujet comporte 4 pages.
Le barème est donné à titre indicatif et pourra être légèrement modifié.

Remarque : Lorsqu'il vous est demandé d'écrire un programme, vous pouvez ignorer les inclusions de bibliothèques. Vous pouvez aussi négliger les cas d'erreurs, sauf s'il est explicitement demandé d'en tenir compte.

Exercice 1.

Fonctions, tableaux et pointeurs (4 points)

1. Écrivez une fonction `int somme(int *t, int n)` qui prend en argument un tableau d'entiers t et le nombre de cases n de ce tableau et renvoie la somme des éléments contenus dans le tableau.
2. Pourquoi est-il nécessaire de passer en argument le nombre de cases du tableau ?
3. Écrivez une fonction `int *range(int n)` qui renvoie un tableau de n cases contenant les entiers compris entre 0 et $(n-1)$ (inclus).
   Attention : Le tableau renvoyé par la fonction `range` doit être alloué dans le tas lors de l'exécution de la fonction.

Exercice 2.

Débuggage (4 points)

On veut écrire une fonction `int minuscule(char *s)` qui prend en argument une chaîne de caractères (dont la fin est indiquée par le caractère `'\0'`) et renvoie le nombre de lettres minuscules que contient cette chaîne. On écrit le programme suivant :

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int minuscules(char *s) {
5    int nbMin = 0;
6    int i;
7    for (i = 0; s[i] != '\0'; i++) {
8      if (s[i] >= a && s[i] <= z) {
9        nbMin = nbMin + 1;
10     }
11   }
12   return *nbMin;
13 }
14
15 int main() {
16   char texte = "Ceci est un TEXTE.";
17   printf("Le texte contient %d minuscules.\n", minuscules(texte));
18 }
```

Lors de la compilation, le compilateur nous indique entre autres les erreurs et avertissements suivants :

```
prog.c: In function ‘minuscules’:
prog.c:8:15: error: ‘a’ undeclared (first use in this function)
prog.c:8:28: error: ‘z’ undeclared (first use in this function)
prog.c:12:9: error: invalid type argument of unary ‘*’ (have ‘int’)
prog.c: In function ‘main’:
prog.c:16:15: warning: initialization makes integer from pointer without a cast
prog.c:17:2: warning: passing argument 1 of ‘minuscules’ makes pointer from integer without a cast
```

1 Pour chacune des erreurs et avertissements indiqués par le compilateur, expliquez brièvement quel est le problème et indiquez les lignes à modifier pour résoudre le problème.

Exercice 3.

Mémoire (4 points)

On s'intéresse maintenant à un programme qui calcule les valeurs de la suite de *Fibonacci* définie par la relation de récurrence suivante :

$$u_0 = 1, \quad u_1 = 1$$
$$\forall n \geq 0, \quad u_{n+2} = u_n + u_{n+1}$$

Le programme va enregistrer toutes les valeurs calculées dans un tableau `fibo_tab` et compléter le tableau jusqu'à la valeur désirée.

On considère les trois variantes obtenues à l'aide du code suivant, selon que l'on décommente l'une des lignes 5, 9 ou 10 (il faut décommenter exactement l'une de ces trois lignes pour déclarer correctement la variable `fibo_tab`) :

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #define TABSIZE 3000000
4
5  // int fibo_tab[TABSIZE]; // variante 1
6
7  int fibo(int n) {
8    int i;
9    // int fibo_tab[TABSIZE]; // variante 2
10   // int *fibo_tab = malloc(TABSIZE * sizeof(int)); // variante 3
11   fibo_tab[0] = 1;
12   fibo_tab[1] = 1;
13   for (i = 0; i < n-2; i++) {
14     fibo_tab[i+2] = fibo_tab[i] + fibo_tab[i+1];
15   }
16   return fibo_tab[n-1];
17 }
18
19 int main() {
20   printf("fibo(3000000) = %d\n", fibo(3000000));
21 }
```

1 Pour chacune des trois variantes, indiquez dans quel espace de la mémoire du processus sera placé le tableau `fibo_tab`.

Chacune des trois variantes peut être compilée sans erreurs. Cependant lors de l'exécution lorsque l'on essaie de calculer trois millions de valeurs, comme c'est le cas dans le code présenté, l'une des variantes produit une erreur.

2 Quelle est la variante qui ne s'exécute pas correctement ? Quel est le problème ?

Lors de l'exécution des deux variantes qui ne génèrent pas d'erreurs, le message qui s'affiche est

```
fibo(3000000) = 768372992
```

Cette valeur est très inférieure à la valeur réelle du 30000000-ème élément de la suite de Fibonacci (qui est de l'ordre de $6,03 \times 10^{626962}$).

3 Expliquez pourquoi le résultat n'est pas correct.

Exercice 4.

Manipulation de fichiers (4 points)

1 Écrivez un programme qui ouvre un fichier dont le nom est passé en argument lors de l'appel en ligne de commande et affiche à l'écran le nombre de lignes (délimitées par le caractère `'\n'`), ainsi que le nombre de caractères qu'il contient.
Par exemple, si le fichier `toto.txt` contient 3 lignes et 250 caractères, la commande « `./a.out toto.txt` » doit afficher le message « 3 lignes et 250 caractères. »
Remarque : Vous pouvez utiliser selon votre préférence les fonctions de la bibliothèque `stdio.h` telles que `fopen`, `fgetc`, etc. ou les appels système tels que `open`, `read`, etc.

2 Ajoutez à votre programme un test pour vérifier que l'ouverture de fichier s'est bien passée, et affiche un message d'erreur adapté en cas d'erreur.

Exercice 5.

Processus (4 points)
On considère le programme suivant :

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <unistd.h>
4
5  int main() {
6    printf ("pid 0 = %d\n", getpid());
7    int pid;
8    pid = fork();
9    printf ("pid 1 = %d\n", pid);
10   if (fork() == 0) {
11     pid = fork();
12     printf("pid 2 = %d\n", pid);
13   } else {
14     sleep(10);
15   }
16 }
```

1 Représentez à l'aide d'un schéma l'ensemble des processus créés lors de l'exécution du programme, en précisant pour chacun quel est son PID et quel est son parent. On pourra supposer que le premier processus a comme PID 1000 (donc la commande `getpid` de la ligne 6 renvoie 1000) et que les différents processus créés ont des PID successifs 1001, 1002, etc.

2 Donnez la suite de lignes qu'affiche le programme lorsqu'il est exécuté (on suppose toujours que le premier PID est 1000).
Remarque : L'ordre dans lequel les lignes sont affichées n'est pas totalement déterminé, il vous suffit donc de donner une sortie possible du programme.

3 Qu'est-ce qu'un processus *zombie* ? Lors de l'exécution du programme, quels processus deviennent des zombies pendant plus d'une seconde (on rappelle que l'instruction `sleep(10)` provoque un délai de 10 secondes) ?

Annexe

Voici les signatures de quelques fonctions de manipulations de fichier dans la bibliothèque `stdio.h` ainsi que des fonctions d'appels système d'entrée-sortie. Si vous ne vous souvenez plus des arguments exacts pour certaines fonctions (en particulier `open` et `fopen`) mettez des arguments approximatifs.

```
FILE *fopen(const char *restrict filename, const char *restrict mode);
char *fgets(char * restrict str, int size, FILE * restrict stream);
int fgetc(FILE *stream);
int fputc(int c, FILE *stream);
char *fgets(char * restrict str, int size, FILE * restrict stream);
```

```
int fputs(const char *restrict s, FILE *restrict stream);
int fseek(FILE *stream, long offset, int whence);
int fclose(FILE *stream);

int open(const char *path, int oflag, ...);
ssize_t read(int fildes, void *buf, size_t nbyte);
ssize_t write(int fildes, const void *buf, size_t nbyte);
off_t lseek(int fildes, off_t offset, int whence);
int close(int fildes);
```