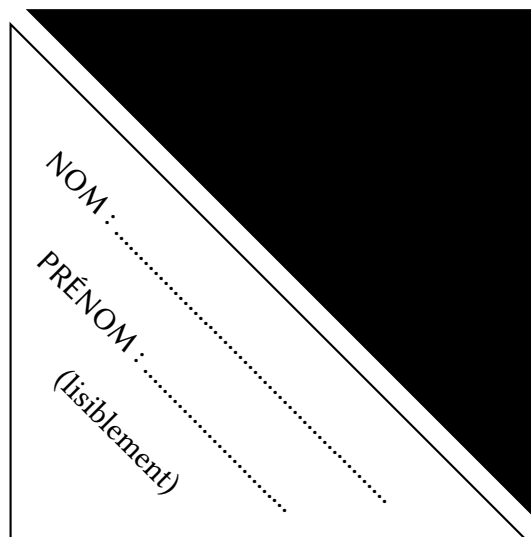


MIDO – L2 MI2E – 2021-2022

Programmation C

Examen du 14 janvier 2022
(durée 2h)



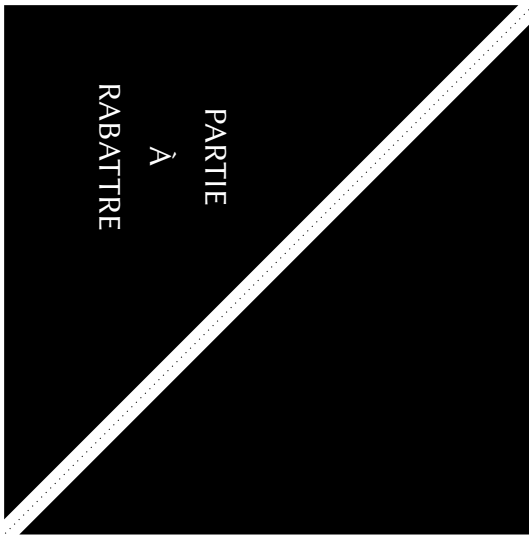
NOM : ...
PRÉNOM : ...
(lisiblement)

Répondez directement sur le sujet.

Documents autorisés : une feuille A4 manuscrite recto-verso

Calculatrice autorisée

Si de la place manque, vous pouvez utiliser les pages supplémentaires situées à la fin.



Exercice 1

Cocher la ou les bonne(s) réponse(s)

1. L'expression :

`a ? 0 : 1`

- ☐ renvoie 0 si a est nul et 1 sinon;
- ☐ renvoie 1 si a vaut 0 ou 1;
- ☒ **renvoie 0 si a est non-nul et 1 sinon;**
- ☐ renvoie 1 si a vaut 1 et 0 sinon;
- ☐ n'a pas de sens en C.

2. Les valeurs associées aux cas d'un **`switch()`** :

- ☒ **peuvent être des caractères (case 'A':);**
- ☐ peuvent être des réels (case 1.5:);
- ☐ peuvent être des variables (case i:);
- ☐ sont forcément des nombres entiers positifs (case 1:).

3. L'instruction :

`long *p = malloc(sizeof(long));`

- ☐ définit un pointeur p toujours initialisé à NULL;
- ☐ définit un long p;
- ☐ définit un pointeur p non initialisé;
- ☒ **définit un pointeur p et un espace de stockage pour un long.**

4. Le test :

`if (x=y) {...} /* x & y sont de type char */`

- ☐ compare les valeurs de x et y; le test est vrai en cas d'égalité;
- ☐ affecte la valeur de y à x; le test est toujours vrai;
- ☒ **affecte la valeur de y à x; le test est vrai si y est différent de 0;**
- ☐ n'a pas de sens en C.

Exercice 2

Soit le programme suivant :

```
#include <stdio.h>
#include <string.h>

int main() {
    char str[] = "skegs";
    char *p1 = str;
    char *p2 = str + strlen(str)-1;
    while (p2 > p1) {
        *p2 = *p1;
        *(p1++) = *(--p2);
    };
    printf("%s\n", str);
}
```

L'exécution du programme affichera :

Solution :

geeks

Exercice 3

Soit le programme suivant :

```
#include <stdio.h>

int a = 4, b = 2;
int func(int i, int *pi, int **ppi) {
    a++;
    b = ++i;
    ++(*pi);
    ++** (ppi++);
    ++**ppi;
    return (**(--ppi))++;
}

int main() {
    int c = 7;
    int d = 6;
    int * T[2] = {&c, &d};
    int e = func(a, &b, &T[0]);
    printf("%d %d %d %d %d\n", a, b, c, d, e);
}
```

L'exécution du programme affichera :

Solution :

5 6 9 7 8

Exercice 4

(Les deux questions sont indépendantes)

1. On désire implémenter un algorithme de tri par sélection **récuratif** sur un tableau d'entiers de taille N, le tableau et sa taille étant passés en argument. L'algorithme de tri est le suivant :
 - on cherche le plus grand des N éléments;
 - on l'échange avec celui situé en dernière place dans le tableau; le plus grand élément est maintenant à sa place finale;
 - on recommence en rappelant la fonction récursivement pour trier le reste du tableau.

Écrire une fonction

```
int max(int T[], int taille);
```

qui renvoie l'indice, dans le tableau, de l'élément le plus grand.

Solution :

Listing 1. indice de l'élément max

```
int max(int T[], int taille) {  
    int i, indice = 0, m = T[0];  
    for (i = 1; i < taille; i++)  
        if (T[i] > m) {  
            indice = i;  
            m = T[i];  
        }  
    return indice;  
}
```

Écrire une fonction

```
void triRec(int T[], int taille);
```

qui trie par l'algorithme récursif précédent le tableau T dont la taille est passée en paramètre.

Solution :

Listing 2. tri récursif d'un tableau d'entiers

```
void triRec(int T[], int taille) {  
    int indice, tmp;  
    if (taille == 0)  
        return;  
    indice = max(T, taille);  
    tmp = T[taille-1];  
    T[taille-1] = T[indice];  
    T[indice] = tmp;  
    triRec(T, taille-1);  
}
```

2. On désire implémenter une fonction de tri par insertion sur un tableau d'entiers passé en argument. Chaque élément du tableau est non-nul et le tableau se termine par la valeur 0.

L'algorithme de tri est le suivant.

Pour chaque élément d'indice i :

- cet élément devient la clef;
- on le compare avec l'élément d'indice $i - 1$;
- si la clef est plus petite, on les échange et on recommence la comparaison avec l'élément précédent (d'indice $i - 2$) et ainsi de suite tant que la clef est plus petite que l'élément qui lui précède;
- quand la clef est à sa place (c'est-à-dire qu'elle est plus grande que ou égale à l'élément qui lui précède), la boucle intérieure est finie et on passe à l'élément d'indice $i + 1$.

Écrivez une fonction

```
void tri(int T[]);
```

qui tri par l'algorithme précédent le tableau T d'éléments non-nuls, terminé par un zéro.

Solution :

Listing 3. tri par insertion

```
void tri(int T[]) {  
    int clef, i = 1, j;  
    while (T[i]) {  
        clef = T[i];  
        j = i - 1;  
        while ((j >= 0) && (clef < T[j])) {  
            T[j+1] = T[j];  
            T[j--] = clef;  
        }  
        i++;  
    }  
}
```

Exercice 5

On souhaite réaliser un programme testant si un nombre donné par l'utilisateur est premier, en essayant de le diviser par tous les nombres plus petits que lui. Pour cela, on a écrit en C le programme suivant :

```
1  #include <stdio.h>

2

3  char premier(int n)

4      int d = 1;

5      while (n%d != 0 & (d < n))

6          d++;

7      printf(d == n);

8  }

9

10 int main() {

11     int i;

12     char *p;

13     printf("Nombre ?");

14     scanf("%f", i);

15     p = premier(i) ? "est" ? "n'est pas";

16     printf("%d %s premier\n", i, p);

17     return 0;

18 }
```

Ce programme comporte **8 erreurs** de syntaxe et/ou d'algorithmique. Corriger ces erreurs avec un stylo de couleur directement sur le programme ci-dessus : utiliser l'espacement entre les lignes pour mettre une version corrigée d'une ligne comportant une (ou des) erreur(s) et/ou ajouter d'éventuelles instructions manquantes.

Solution :

```
#include <stdio.h>

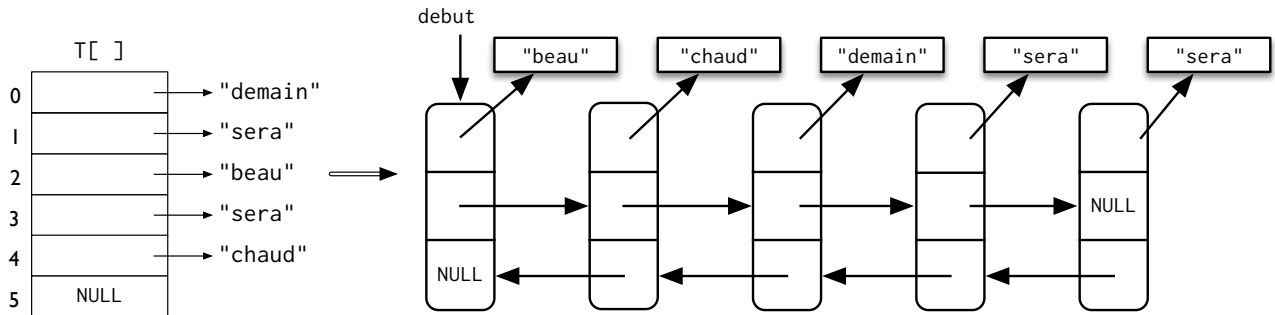
char premier(int n) { // manque l'accolade
    int d = 2; // On commence à d = 2 sinon c'est divisible
    while (n%d != 0 && (d < n)) // && est le ET logique
        d++;
    return(d == n); // return et pas printf
}

int main() {
    int i;
    char *p;
    printf("Nombre ?");
    scanf("%d", &i); // %d pour un int et &i
    p = premier(i) ? "est" : "n'est pas"; // : au lieu de ?
    printf("%d %s premier\n", i, p); // " et pas ' pour la chaîne
    return 0;
}
```

Exercice 6

On souhaite transformer un tableau de chaînes de caractères, terminé par la valeur `NULL`, en liste doublement chaînée **triée** où chaque cellule indique la chaîne. La figure ci-dessous donne un exemple de ce qu'il faut faire.

On pourra utiliser la fonction `int strcmp(char *s1, char *s2);` qui permet de comparer des chaînes de caractères et renvoie une valeur strictement positive si $s1 > s2$, 0 si $s1 = s2$, et une valeur strictement négative si $s1 < s2$.



1. Définir une structure de cellule de liste doublement chaînée `struct cellule {...}` permettant de stocker la chaîne ainsi que les pointeurs sur les cellules suivante et précédente ainsi qu'un `typedef Cellule` équivalent.

Solution :

Listing 4. Tableau vers liste

```
struct cellule {  
    char *str;  
    struct cellule *next;  
    struct cellule *prec;  
};  
typedef struct cellule Cellule;
```


2. Écrire une fonction

Cellule *tab2liste(char *T[]);

qui crée une liste doublement chaînée triée (et renvoie un pointeur sur son début) à partir du tableau de chaînes passé en paramètre. Chaque cellule devra pointer sur la même chaîne que celle dans le tableau, il n'est pas nécessaire de faire une recopie. Les cellules seront triées dans l'ordre des chaînes.

Solution :

Listing 5. Tableau vers liste

```
Cellule *tab2Liste(char *T[]) {
    Cellule *p, *q, *debut = NULL;
    int i = 0;

    while (T[i] != NULL) { /* Pour tous les éléments du tableau */
        p = malloc(sizeof(Cellule));
        p->str = T[i];
        p->next = p->prec = NULL;
        if (!debut) { /* première cellule */
            debut = p;
        } else {
            if (strcmp(p->str, debut->str) < 0) { /* en tête */
                p->next = debut;
                debut->prec = p;
                debut = p;
            } else {
                q = debut; /* parcours de la liste */
                while (q->next && strcmp(p->str, q->next->str) > 0)
                    q = q->next;
                p->next = q->next;
                p->prec = q;
                q->next = p;
                if (p->next != NULL)
                    p->next->prec = p; /* ajout au milieu */
            }
        }
        i++;
    }
    return debut;
}
```