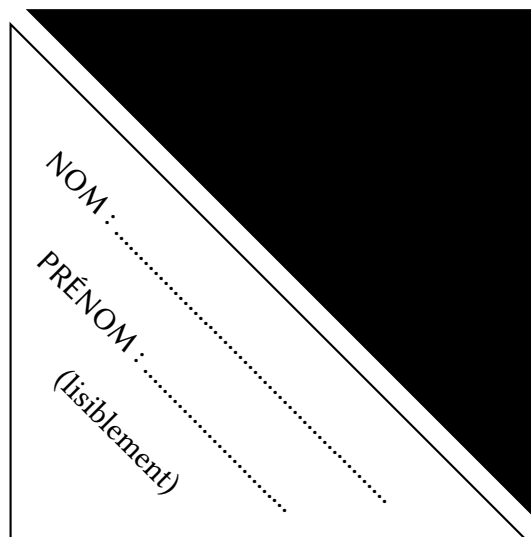


MIDO – L2 MI2E – 2020-2021

Programmation C

Examen du 14 janvier 2021
(durée 2h)



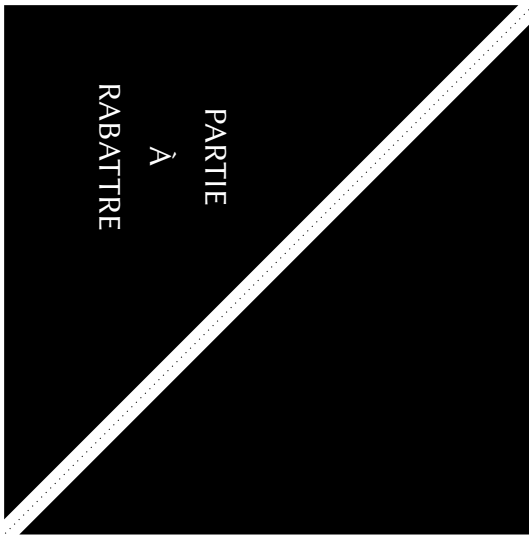
NOM : ...
PRÉNOM : ...
(lisiblement)

Répondez directement sur le sujet.

Documents autorisés : une feuille A4 manuscrite recto-verso

Calculatrice interdite

Si de la place manque, vous pouvez utiliser la page supplémentaire située à la fin.



Exercice 1

Cocher la ou les bonne(s) réponse(s)

1. $a=b$ est équivalent à $a==b$

- ☐ toujours;
- ☐ jamais;
- ☐ si a et b ont la même valeur;
- ☒ **si a et b valent 1.**

2. Quelle(s) instruction(s) permet(tent) d'arrêter définitivement une boucle `while()` ?

- ☐ `printf(...);`
- ☒ **`return;`**
- ☒ **`break;`**
- ☐ `continue;`

3. Deux variables de même nom mais de types différents peuvent être déclarées dans un programme :

- ☐ toujours;
- ☐ jamais;
- ☒ **si l'une est globale et l'autre locale à une fonction;**
- ☒ **si les deux sont locales mais déclarées dans des fonctions différentes.**

4. Que vaut i après la boucle suivante :

```
int i = 11;
while (1) {
    if (i%2 == i%3)
        break;
    if (i-- > 9)
        continue;
    i -= 2;
}
```

☐ Il y a en fait une erreur de syntaxe ;

☐ C'est une boucle infinie ;

☒ 6

☐ 7

☐ 11

5. Pour allouer dynamiquement un espace de stockage pour une chaîne de 9 caractères, il faut écrire :

☐ malloc(9 * sizeof(char))

☒ malloc(10 * sizeof(char))

☐ malloc(sizeof(char *))

☐ malloc(sizeof(char[]))

6. Un prototype de fonction :

☒ sert au compilateur à vérifier la conformité des appels à cette fonction ;

☐ est toujours obligatoire dans le code ;

☐ sert à définir pour le compilateur dans quel fichier se trouve la fonction ;

☒ n'est pas obligatoire mais évite certains bugs.

Exercice 2

Soit le programme suivant :

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

int main() {
    char str[] = "qcpqcp";
    char *p1 = str;
    char *p2 = str + strlen(str)-1;
    do {
        *(p2--) = ++(*p1);
    } while (*(++p1));
    printf("%s\n", str);
    return 0;
}
```

L'exécution du programme affichera :

Solution :

serres

Exercice 3

Soit le programme suivant :

```
#include <stdio.h>

int a = 0;

int func(int *X, int *Y, int Z) {
    a++;
    Z = (*X)++;
    *Y = ++(*(++X)) - 2;
    return Z + *(++X) - 1;
}

int main() {
    int b = -10;
    int c[3] = {2, 3, 5};
    int d = func(c, &b, b);
    printf("%d %d %d %d %d %d\n", a, b, c[0], c[1], c[2], d);
    return 0;
}
```

L'exécution du programme affichera :

Solution :

1 2 3 4 5 6

Exercice 4

On souhaite réaliser un programme permettant d'approcher la valeur de la constante e (base des logarithmes naturels), qui vaut environ 2.71828, en utilisant la formule :

$$e \approx \sum_{i=0}^n \frac{1}{i!}$$

Pour cela, on a écrit en C le programme suivant :

```
1  #include <stdio>

2  long fact(int n)

3      int r=1, i=1;

4      while (i < n)

5          r *= i++;

6      printf(r);

7  }

8

9  int main() {

10     int i = 0, n;

11     float exp = 0,0;

12     scanf("%d", n);

13     while (i <= n) {

14         exp = exp + 1.0%fact(i);

15     }

16     printf("Approximation de 'e' : %f\n",exp);

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> // bon nom d'entête
long fact(int n) { // oubli de l'accolade
    int r=1, i=1;
    while (i < n+1) // il faut aller jusqu'à n+1
        r *= i++;
    return r; // return et pas printf
}

int main() {
    int i = 0, n;
    float exp = 0.0; // 0.0 ou 0 mais pas 0,0
    scanf("%d", &n); // &n et pas n
    while (i <= n) {
        exp = exp + 1.0/fact(i); // / et pas le modulo %
        i = i + 1; // Ne pas oublier l'incrément
    }
    printf("Approximation de 'e' : %f\n",exp);
    return 0;
}
```

Exercice 5

On souhaite supprimer les lettres dupliquées qui se suivent dans une chaîne. Par exemple, avec la chaîne "aabbcccddeef" en entrée, on veut avoir "abcdef" en sortie; avec "aabaabbabab" en entrée, on veut "abababab" en sortie; une chaîne vide rendra une chaîne vide...

Dans cet exercice, vous avez le droit d'utiliser n'importe quelle fonction liée aux chaînes, par exemple `strlen()`.

1. Écrire une fonction

```
char *suppDoublons(char *str);
```

qui crée une nouvelle chaîne, copie de celle passée en paramètre dans laquelle les lettres identiques successives sont supprimées pour n'en laisser qu'une. (On ne demande pas à ce que l'espace réservé pour la nouvelle chaîne soit minimal.)

Solution :

Listing 1. `suppDoublons()`

```
char *suppDoublons(char *str) {
    char *p = malloc(strlen(str)+1);
    int iStr = 0, iP = 0;

    p[0] = str[0];
    while (str[iStr]) {
        if (str[iStr] != p[iP])
            p[++iP] = str[iStr];
        iStr++;
    }
    p[iP] = '\0';
    return p;
}
```

Autre version :

Listing 2. `suppDoublons()`

```
char *suppDoublons(char *str) {
    char *p = malloc(strlen(str)+1);
    char *save = p; /* sauvegarde du debut */
    char c;
    do {
        c = *p++ = *str++;
        while (c && c == *str)
            str++;
    } while (c);
    return save;
}
```

2. Écrire la fonction principale `int main()` qui
- demande une chaîne à l'utilisateur (qu'on supposera toujours de taille strictement inférieure à 256 caractères);
 - affiche la chaîne renvoyée par la fonction précédente à partir de celle de l'utilisateur.

Solution :

Listing 3. programme principal

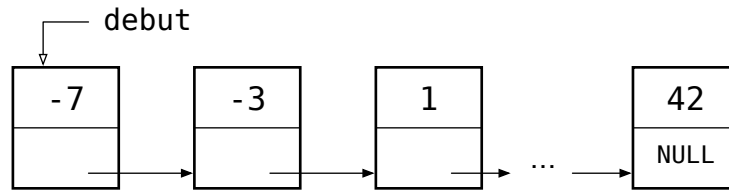
```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

char *suppDoublons(char *str);
int main() {
    char buff[256];
    fgets(buff, 256, stdin);
    printf("%s\n", suppDoublons(buff));
    return 0;
}

// gets(buff) est moins bien car non-sécurisé.
// scanf("%s",buff) est moins bien car ne peut pas lire n'importe
// quelle chaîne (elle ne peut pas contenir d'espace).
```

Exercice 6

On souhaite gérer une liste simplement chaînée d'entiers (de type `int`) de telle sorte que la liste soit **toujours triée** par ordre croissant. On gèrera la liste à partir d'un pointeur sur sa première cellule (pointeur valant `NULL` si la liste est vide); ce pointeur sera passé en argument à chaque fonction et si sa valeur doit être modifiée, la fonction renverra la nouvelle adresse du début de la liste.



1. Définir la structure correspondante `struct cellule` ainsi qu'un `typedef Cellule` équivalent.

Solution :

Listing 4. structure

```
struct cellule {
    int valeur;
    struct cellule *next;
};

typedef struct cellule Cellule;
```

2. Écrire une fonction

```
int total(Cellule *debut);
```

qui renvoie la somme des valeurs de la liste.

Solution :

Listing 5. total()

```
int total(Cellule *debut) {
    int total = 0;
    while (debut) {
        total += debut->valeur;
        debut = debut->next;
    }
    return total;
}
```

3. Écrire une fonction

```
void affiche(Cellule *debut);
```

qui affiche sur une ligne toutes les valeurs de la liste.

Solution :

Listing 6. affiche()

```
void affiche(Cellule *debut) {  
    while (debut) {  
        printf("%d ", debut->valeur);  
        debut = debut->next;  
    }  
    printf("\n");  
}
```

4. Écrire une fonction

```
Cellule *suppNeg(Cellule *debut);
```

qui supprime de la liste toutes les valeurs strictement négatives (et libère la mémoire).

Solution :

Listing 7. suppNeg

```
Cellule *suppNeg(Cellule *debut) {  
    Cellule *p;  
    while (debut && debut->valeur < 0) {  
        p = debut->next;  
        free(debut);  
        debut = p;  
    }  
    return debut;  
}
```

5. Écrire une fonction

```
Cellule *insere(Cellule *debut, int x);
```

qui insère une valeur x dans une nouvelle cellule au bon endroit de la liste.

Solution :

Listing 8. insere

```
Cellule *insere(Cellule *debut, int x) {  
    Cellule *courant = debut;  
    Cellule *q = malloc(sizeof(Cellule));  
    q->valeur = x;  
    q->next = NULL;  
    if (debut == NULL) {  
        return q; /* liste vide */  
    }  
    else if (x < debut->valeur) {  
        q->next = debut;  
        return q; /* ajout en tete */  
    } else {  
        while (courant->next && courant->next->valeur < x)  
            courant = courant->next;  
        q->next = courant->next;  
        courant->next = q;  
    }  
    return debut;  
}
```

6. Écrire la fonction principale `main()` qui
- crée un pointeur de début initialisé à `NULL` ;
 - insère dix nombres aléatoires entre `-5` et `5` dans la liste (on pourra utiliser `random()%11-5`);
 - affiche la liste et la somme de ses valeurs ;
 - supprime les éléments négatifs et réaffiche la liste.

Solution :

Listing 9. main()

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int main() {
    srand(time(NULL));
    int i;
    Cellule *debut = NULL;
    for (i = 0; i < 9; i++)
        debut = insere(debut, random()%11 -5);
    affiche(debut);
    printf("\n%d\n\n", total(debut));
    debut = suppNeg(debut);
    affiche(debut);
    return 0;
}
```