TP Noté - Introduction au C

SUJET A

21/02/2025

Instructions :

- Durée : 1h30
- Utilisation d'internet, de vos cours et de vos TP **autorisé**, accès à un LLM **interdit** (ChatGPT, LLama, Gemini, etc ...)
- Codes sources (tous les fichiers source et programmes) à déposer sur le dépôt "TP Noté Intro C - SUJET A" dans le module "FASE" sur Moodle au format ZIP (**Aucun retard autorisé**)
- **Il est fortement déconseillé de faire les exercices dans l'ordre. Réalisez-les du plus simple au plus difficile !**

Code initial classique

Vous considérez ce code comme étant le "code initial classique" pour chaque exercice :

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stddef.h>
#include <stdbool.h>
#include <limits.h>

int main(int argc, char** argv) {
    printf("Hello World!\n");
    return 0;
}
```

Exercice 1

Créez un fichier "exo1.c" (votre programme s'appellera "exo1") contenant le code initial classique.

1. Demandez à l'utilisateur de rentrer un nombre entier positif "N".
2. Créez un tableau dynamique de "int" de taille "N" avec `malloc`

3. Remplissez les N cases du tableau avec les valeurs "1, 2, 3, ..., N".
4. Affichez chaque case du tableau. Verifiez que la dernière case du tableau est "N".
5. Créez une fonction "appliquer" qui ne retourne rien et prend en entrée un tableau de int "array", sa taille N et un pointeur de fonction "function" (qui prend en entrée un "int" et renvoie un "int"). La fonction "appliquer" transforme chaque case de "array" en appliquant la fonction "function".
6. Définissez une fonction "doubler" qui prend en entrée un "int" et renvoie un "int". Cette fonction applique l'opération $x \times 2$.
7. Définissez une fonction "auCarre" qui prend en entrée un "int" et renvoie un "int". Cette fonction applique l'opération x^2.
8. Affichez dans le terminal les valeurs de votre tableau initial (rempli de 1, 2, 3, ..., N) après avoir appliqué "doubler" (avec la fonction "appliquer").
9. Affichez dans le terminal les valeurs de votre nouveau tableau après avoir appliqué "auCarre" (avec la fonction "appliquer").

Exercice 2

Créez un fichier "exo2.c" (votre programme s'appellera "exo2") contenant le code initial classique.

1. Créez une fonction "caractereVersMajuscule" qui prend un caractère `char` et renvoie le même caractère en majuscule (si c'est une lettre en minuscule).
   Affichez le résultat pour les caractères :
   - 'a' [doit se transformer en 'A'],
   - 'G' [doit se transformer en 'G'],
   - '!' [doit se transformer en '!']
2. Créez une fonction "texteEnMajuscule" qui prend en entrée une chaine `char* chaine` et transforme chaque caractère pour les passer en majuscule en utilisant la fonction "caractereVersMajuscule". Cette fonction ne renvoie rien.
   Affichez le résultat pour les chaines :
   - "Hello" [doit se transformer en "HELLO"],
   - "World!" [doit se transformer en "WORLD!"],
   - "Hello World!" [doit se transformer en "HELLO WORLD!"]
3. Créez une fonction "couperAuPremierEspace" qui prend en entrée une chaine `char* chaine` et ne renvoie rien. Transformez la chaine de caractère pour qu'elle se termine au premier caractère espace (" ").
   Affichez le résultat pour les chaines :
   - "Hello World!" [doit se transformer en "Hello"],
   - "Polytech" [doit se transformer en "Polytech"]
   - "4-3*2 = 5" (un espace avant et après le "=") [doit se transformer en "4-3*2"].

Exercice 3

Créez un fichier "exo3.c" (votre programme s'appellera "exo3") contenant le code initial classique.

1. Créez deux variables "x" et "y" contenant les valeurs 4.5 et 1.3
2. Créez une fonction "echanger" qui échange la valeur des deux variables passées en paramètre.
3. Créez une fonction "normaliser_vecteur" qui prend en entrée les deux composants ("x", "y") d'un vecteur 2D. Le programme "normalise" le vecteur, c'est à dire :

$$x_{\text{en sortie}} = \frac{x}{\sqrt{x^2 + y^2}} \tag{1}$$

$$y_{\text{en sortie}} = \frac{y}{\sqrt{x^2 + y^2}} \tag{2}$$

   Dans votre fonction "main()", verifiez qu'à la sortie de votre fonction $x^2 + y^2 = 1$ en affichant "x", "y" et "$x^2 + y^2$".

Exercice 4

Créez un fichier "exo4.c" (votre programme s'appellera "exo4") contenant le code initial classique.

1. Créez une variable "x" de type "entier" dont la valeur est 42. Affichez cette variable dans le terminal.
2. Créez une variable "y" de type "flottant" dont la valeur est 3.1415. Affichez cette variable dans le terminal.
3. Créez une variable "z" de type "flottant" dont la valeur est le produit de "x" et "y". Affichez cette variable dans le terminal.
4. Étant donné vos variables "x" et "y", citez au moins 1 problème avec le résultat de ce calcul, et expliquez-le (Affichez votre réponse dans le terminal) :

   ```
   char ma_variable = x * y;
   ```

Exercice 5

Créez un fichier "exo5.c" (votre programme s'appellera "exo5") contenant le code initial classique.

1. Avant la définition de votre fonction "main()", ajoutez les lignes suivantes :

```
typedef struct {
    int numero;
    char caractere;
    double decimale;
} MaStructure;

typedef long long int tresLongInt;
```

2. Créez une fonction "afficher_taille_des_données()" qui affiche la taille (en octets) des types de données suivantes :
   - `int`
   - `float`
   - `double`
   - `double`
   - `char`
   - `tresLongInt`
   - `MaStructure`
   - `MaStructure*`
3. Y a-t-il une différence de taille entre `MaStructure` et `MaStructure*` ? Si oui, pourquoi ? (Affichez votre réponse dans le terminal)

Exercice 6

Créez un fichier "exo6.c" (votre programme s'appellera "exo6") contenant le code initial classique.

1. Que signifient les paramètres "argc" et "argv" dans la définition de la fonction "main" ? (Affichez votre réponse dans le terminal)
2. Votre programme doit afficher la valeur de "argc" quand vous appelez votre programme avec :
   - `./exo6`
   - `./exo6 argument1`
   - `./exo6 argument1 --option1`
   - `./exo6 argument1 -f --option1`
   - `./exo6 --help`
3. Votre programme doit maintenant afficher chaque valeur contenue dans le paramètre "argv" (1 valeur par ligne) quand vous appelez votre programme avec :
   - `./exo6`
   - `./exo6 argument1`
   - `./exo6 argument1 --option1`
   - `./exo6 argument1 -f --option1`
   - `./exo6 --help`

Exercice 7

Créez un fichier "exo7.c" (votre programme s'appellera "exo7") contenant le code suivant :

```c
#include <stdio.h>

int main(int argc, char** argv)
{
    for (unsigned int i = 11; i >= 0; i++) {
        printf("%f -- %f \n", i, i/2);
    }
}
```

1. Corrigez le code pour qu'il affiche le résultat suivant sans erreur et sans warning :

```
10 -- 5.00000
9 -- 4.50000
8 -- 4.00000
7 -- 3.50000
6 -- 3.00000
5 -- 2.50000
4 -- 2.00000
3 -- 1.50000
2 -- 1.00000
1 -- 0.50000
0 -- 0.00000
```

Exercice 8

Créez un fichier "exo8.c" (votre programme s'appellera "exo8") contenant le code initial classique.

1. Créez une fonction "doubler" qui prend en entrée une variable décimale "x" et renvoie $2 \times x$.
2. Votre programme doit afficher la valeur doublée de :
   - 100,
   - 89,
   - -25.8,
   - 0
3. Créez une fonction "puissance" qui prend en entrée une variable "x" et une variable "y" et renvoie x^y.
4. Votre programme doit afficher la valeur de :
   - 2^5,
   - 74^2,
   - 2.7182^7

Exercices conseillés

Par ordre de difficulté, je vous conseille :

- Exercice 4,
- Exercice 5,
- Exercice 8,
- Exercice 3,
- Exercice 2,
- Exercice 1,
- Exercice 6,
- Exercice 7