Remise à niveau en C - 1

Ce module a pour but de vous familiariser avec quelques notions du language C et de préparer les TP du module C++.

Durée du module : $4 * 3h$

I Mise en place

1 Programme minimal

Créez un fichier main.c dans le répertoire de votre choix et recopier ce programme :

```c
#include <stdio.h>

int main(){

  printf("hello\n");
  return 0;
}
```

2 Compilation en ligne de commande

Pour compiler le programme, il suffit de se placer dans le dossier contenant le fichier `main.c` et de rentrer la commande suivante dans le terminal : `gcc main.c -o main`

(l'argument `-o` permet de choisir le nom de l'exécutable).

Lorsque la compilation est terminée, un fichier `main` apparait dans le répertoire. La commande `./main` permet de le lancer.

II Mise en pratique des notions de base

1 Compilation séparée

En plus du fichier `main.c`, il est pratique de diviser les programmes en plusieurs fichiers : les fichiers `.c` sont en général (sauf pour `main.c`) accompagné d'un fichier `.h` (fichier *header*, ou fichier d'en-tête) qui contient les prototypes des fonctions et les définitions des structures. Dans la suite, créez un fichier `exercice.c` et `exercice.h`.

On implémentera les fonctions dans ces fichiers et on testera leur bon fonctionnement dans le fichier `main.c`.

Pour ajouter le fichier `exercice` au fichier principal, il suffit d'ajouter au début de `main.c` :

```
#include "exercice.h"
```

La commande pour compiler devient :

```
gcc main.c exercice.c -o main
```

Quand le nombre de fichiers deviendra important, on utilisera un outil de compilation plus sophistiqué (écriture d'un *makefile*, ou utilisation de *Cmake* par exemple).

2 Exercices

i Nombre aléatoire

Écrire une fonction `int random(int n)` qui renvoie un entiers aléatoires entre 0 et n.

(on utilisera l'instruction *rand()* et l'opérateur %)

ii Affichage d'un tableau

Écrire une fonction `void print_array(int T[], int n)` qui prend en argument un tableau de taille n et qui affiche chacun de ses éléments.

iii Recherche du maximum

Écrire une fonction `int max_array(int T[], int n)` qui prend en argument un tableau de taille n et qui renvoie son maximum.

III Présentation des pointeurs

Cette section aborde une notions phare du language C : les pointeurs.

Les pointeurs étant un outil indispensable pour la suite, je vous encourage à lire un cours sur le sujet si le besoins s'en fait sentir.

1 Exemple simple d'un pointeur sur une variable

```c
#include <stdio.h>

int main() {

  int a = 4;

  int* p; // création d'un pointeur nul:
          // (p) ne pointe sur rien car il n'est pas initialisé

  p = &a; // on fait pointer (p) sur l'adresse de (a)

  // quelques tests:
  printf("valeur de a: %d\n", a);
  printf("adresse mémoire de a: %p\n", (void *)&a); // affichage de l'adresse
  ↪  en hexadécimal
  printf("\n");

  printf("valeur du pointeur p: %p\n", (void *)p); // affichage de l'adresse
  ↪  sauvegardée dans (p) en hexadécimal
  printf("valeur de a en passant par le pointeur p: %d\n", *p);


  *p = 0; // modification de a en utilisant le pointeur
  printf("valeur de a: %d\n", a);

  return 0;
}
```

2 Exercices

i Échange

Écrire une fonction `void swap(int *a, int *b)` qui échange les valeurs de deux variables.

ii Pointeur sur un tableau

Les tableaux étant stockés en mémoire de façon contigüe, il est possible de parcourir un tableau en incrémentant (++) un pointeur sur son premier élément. L'incrémentation va faire bouger le pointeur, c'est à dire qu'il pointera sur la case suivante du tableau.

```c
#include <stdio.h>

int main() {
  int T[5] = {1, 2, 3, 4, 5};

  int* p = &T[0];

  for (int i = 0; i < 5; i++) {
    printf("&T[%i] = %p\n", i, (void *)&T[i]); // adresse de la case (i)
    printf("p = %p\n", (void *)p); // adresse pointée
    printf("*p = %d\n", *p); // valeur pointée
    printf("-----------------\n");

    p++; // le pointeur bouge sur la case suivante
  }
  return 0;
}
```

Executer ce programme.

iii Modification d'un tableau passé par pointeur

Écrire une fonction `void product(int* p, int b, int n)` qui prend en argument un tableau de n entiers et qui multiplie chacun de ses éléments par b.

Essayez de provoquer un crash en faisant pointer p à l'extérieur du tableau.

IV Tableaux - Allocation dynamique

L'allocation dynamique permet de créer des tableaux de taille non connue à l'écriture du programme (par exemple d'une taille définie par un utilisateur). Pour cela, il faut utiliser le mot clé `malloc` qui va allouer une certaine quantité de mémoire au programme en cours d'exécution.

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

int main() {

  int n = 0;
  printf("taille du tableau ?");
  scanf("%d", &n); // entrée utilisateur sauvegardé dans (n)

  int *T = malloc(n * sizeof(int)); // allocation de mémoire (non
  ↪  initialisé)
  //   int *T = calloc(n, sizeof(int)); // allocation de mémoire (initialisé
  ↪  à 0)

  // affichage de chaque éléments
  for (int i = 0; i < n; ++i) {

    printf("T[%d] = %d\n", i, *(T + i)); // on peut aussi remplacer "*(T+i)"
    ↪  par "T[i]" pour alléger l'écriture
  }

  free(T); // libère l'espace mémoire alloué
  return 0;
}
```

1 Exercice

i Tableaux d'entiers aléatoires

Écrire une fonction `int* random_array(int n)` qui renvoie un pointeur vers un tableau de n entiers aléatoires.

ii Tri croissant

Écrire une fonction `void sort(int* p, int n)` qui prend en argument un tableau de n entiers et le trie dans l'ordre croissant.

iii Tableaux 2D d'entiers aléatoires

Écrire une fonction `int** random_array_2D(int n)` qui renvoie un pointeur vers un tableau de $n \times n$ entiers aléatoires.

(On utilisera un tableau de pointeurs vers int : `int** T`)

V Structures

Les structures sont des types complexes regroupant plusieurs variables de différents types.

1 Exercice : un type Vecteur

Définir une structure représentant un vecteur à n dimensions.

i Initialisation

Écrire une fonction d'initialisation qui remplie tout les champs de la structure pointée.

```
void Vector_init(struct Vector* v, float* data)
```

ii Addition

Écrire une fonction d'addition.

```
Vector Vector_add(struct Vector* u, struct Vector* v)
```

iii Produit scalaire

Écrire une fonction calculant le produit scalaire de deux vecteurs.

```
float Vector_dot(struct Vector* u, struct Vector* v)
```