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 : 12h

I Mise en place

1 Programme minimal

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

```c
#include <stdio.h>

int main(int argc, char* argv[]){
  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.

3 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` :

```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).

4 Compilation avec un makefile

On peut simplifier le processus de compilation en écrivant un fichier `makefile` :

```
all : main.o exercices.o
	gcc -o exercices main.o exercices.o
	rm -f *.o
	clear
main.o : main.c
	gcc -c -Wall main.c

exercices.o : exercices.c
	gcc -c -Wall exercices.c

clean :
	rm -f exercices *.o
```

La commande de compilation est remplacée par `make`.

On peut chaîner les commandes de compilation / clear / exécution :

`make && clear && ./main`

II Mise en pratique des notions de base

1 Exercices

i Nombre aléatoire

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

(on utilisera l'instruction *rand()*)

Remplir un tableau d'entier aléatoires pour la suite.

ii Affichage d'un tableau

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

iii Recherche du maximum

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

III Rappels sur les pointeurs

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

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'opérateur d'incrément ++ va ajouter 1 à l'adresse du 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;
}
```

Écrire une fonction `void print_debug_array(int* p, int size)` qui prend en argument un tableau de size entiers et qui affiche pour chaque élément sa valeur et son adresse.

iii Modification d'un tableau passé par pointeur

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

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 size)` qui renvoie un pointeur vers un tableau de size entiers aléatoires.

ii Fonction argmax

Écrire une fonction `int* argmax_array(int* p, int size)` qui renvoie l'indice de l'élément maximum du tableau pointé par p ;

Exemple : $max(1, 3, 9, 5, 3) = 9$ et $argmax(1, 3, 9, 5, 3) = 2$

iii 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 décroissant.

On pourra réaliser cette fonction avec argmax() et swap() (tri à bulle).

iv 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 Libérer la mémoire

Pensez à libérer la mémoire allouée !

On testera la présence de fuites mémoires avec l'outil Valgrind. `valgrind ./exercices`.

V Structures

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

1 Exercice : un type Pixel

Définir une structure représentant un Pixel à 3 canaux RGB. Réaliser une fonction d'affichage `void print_pixel(struct Pixel* p)`

i Addition

Écrire une fonction d'addition et de multiplication canaux à canaux.

```
Pixel Pixel_add(struct Pixel* u, struct Pixel* v)
```

```
Pixel Pixel_mult(struct Pixel* u, struct Pixel* v)
```

ii Une image

Écrire une fonction renvoyant un pointeur sur un tableaux 2 dimensions de structures Pixel dont les pixels sont initialisés aléatoirement ;