Mise à niveau en C

Structures et préprocesseur

Enseignant: P. Bertin-Johannet

Les structures

- Les structures permettent de créer de nouveaux types
- Le nouveau type sera une combinaison de plusieurs champs

```c
struct User{
  int age;
  char* name;
}
int main(){
  struct User alain = {.age = 24, .name = "alain"};
  printf("%d\n", alain.name);
}
```

Les structures

- Une structure permet par exemple de regrouper des variables

```
int main(){
  int x1, y1, z1, x2, y2, z2;
  read_point(&x1, &y1, &z1);
  read_point(&x2, &y2, &z2);
  calc_dist(x1, x2, y1,
            y2, z1, z2);
}
```

```
struct Point{
  int x;
  int y;
}
int main(){
  struct Point p1, p2;
  read_point(&p1);
  read_point(&p2);
  calc_dist(p1, p2);
}
```

Définition d'une structure

- Pour définir une structure on utilise le mot clé `struct`
- On écrit ensuite le nom de la structure puis la déclaration des variables qu'elle contiendra.

```
struct User{
  int age;
  char* name;
  float height;
  char* password;
}
```

Utilisation d'une structure

- On peut définir des variables du type de la structure en écrivant :
  `struct NomStructure variable;`
- Pour accéder à une variable contenue, on utilise un point

```c
int main(){
  struct User alice;
  alice.age = 25;
  alice.password = "Bonjour";
  printf("%d\n", alice.age);
}
```

Utilisation d'une structure

- Depuis C99 , on peut instancier une structure en lui donnant des valeurs ainsi :

```
variable = {.champ1 = valeur1, .champ2 = valeur1, ...};


int main(){
  struct User* alice = {.age = 25, .password = "Bonjour"};
  printf("%d %s\n", alice.age, alice.password);
}
```

Allocation de mémoire d'une structure

- On peut aussi reserver de la mémoire pour une structure en utilisant `malloc`

```c
int main(){
  struct User* alice = malloc(sizeof(struct User));
  (*alice).age = 25;
  (*alice).password = "Bonjour";
  printf("%d %s\n", (*alice).age, (*alice).password);
}
```

Pointeurs vers une structure

- Afin d'éviter d'écrire `(*nom).var` à chaque utilisation d'un pointeur vers une structure on peut utiliser l'opérateur flèche

```c
int main(){
  struct User alice = malloc(sizeof(struct User));
  alice->age = 25;
  alice->password = "Bonjour";
  printf("%d %s\n", alice->age, alice->password);
}
```

Tableau de structures

- On peut aussi créer un tableau de structures

```
int main(){
  struct User utilisateurs[5]; // un tableau de 5
utilisateurs dans la pile
  struct User *administrateurs = malloc(2*sizeof(struct
User)); // un tableau de 2 admins dans le tas
}
```

Alias de type

- Afin d'éviter d'écrire `struct` à chaque utilisation, on peut utiliser le mot clé `typedef` pour créer un **alias de type**

```c
// ici on permet d'écrire User au lieu de struct User
typedef struct User User
int main(){
  User ann = {.age = 28, .name = "ann"};
}
```

Structure et mémoire

- Les champs d'une structure sont enregistrés dans la mémoire dans l'ordre ou ils ont été déclarés

```
struct vecteur{
  int taille; // taille apparaitra en premier dans la
mémoire
  int* elements; // elements en second
}
```

Alignement et taille d'une structure

- A chaque type est associé une valeur alignement.
- Pour des raisons matérielles, une variable d'un type doit être enregistrée à une addresse multiple de son alignement
- Par exemple si l'alignement d'un `int` est quatre, son addresse mémoire devra être un multiple de quatre.
- Pour satisfaire cette condition tout en respectant l'ordre de présence des champs d'une structure, le compilateur peut ajouter des octets de compensation

Alignement et taille d'une structure : Exemple

- Ici on considère un alignement de 4 pour `int` et 1 pour `char`

```
struct IntChar{
  int a;
  char b;
}
int main(){
  struct IntChar a = {.a = 5, .b = 6};
}
```

Valeurs	Addresses
5	0xff80
	0xff81
	0xff82
	0xff83
6	0xff84

Alignement et taille d'une structure : Exemple

- Ici on considère un alignement de 4 pour `int` et 1 pour `char`

```
struct CharInt{
  char a;
  int b;
}
int main(){
  struct CharInt a = {.a = 5, .b = 6};
}
```

Valeurs	Addresses
5	0xff80
??	0xff81
	0xff82
	0xff83
6	0xff84
	0xff85
	0xff86
	0xff87

Le préprocesseur

- On peut utiliser des instructions appelées de **préprocesseur** pour manipuler le texte de nos fichiers
- Le préprocesseur est la première étape de la compilation
- Pour cela il faut utiliser des instructions commenceant par un dièze
- Toutes ces instructions manipulent du texte sans se soucier de son interprétation

L'instruction define

- L'instruction `#define` permet de créer une variable préprocesseur
- Une variable préprocesseur contient uniquement du texte
- Le nom de la variable sera remplacé par le texte dans la suite du programme

```
#define N 1
#define PLUS +
#define A_EQ int a =
int main(){
  A_EQ N PLUS N;
}
```

L'instruction ifdef

- L'instruction `#ifdef` permet d'include du code uniquement si une variable préprocesseur est définie

```c
#define N 1
#ifdef N
int f(){return 1;}
#else
int f(){return 4;}
#endif

int main(){
  printf("%d\n", f());
} // affiche 1
```

L'instruction ifndef

- L'instruction `#ifndef` permet d'include du code uniquement si une variable préprocesseur n'est pas définie

```c
#define N 1
#ifndef N
int f(){return 1;}
#else
int f(){return 4;}
#endif

int main(){
  printf("%d\n", f());
} // affiche 4
```

L'instruction include

- L'instruction préprocésseur `#include` permet de copier le texte écrit dans un fichier

```
// copie colle le texte du fichier bonjour.txt
#include "bonjour.txt"
int a;
// copie colle encore le texte du fichier bonjour.txt
#include "bonjour.txt"
```

Problème de include

- Si on inclut deux fichiers qui incluent un même fichier, ce dernier sera collé deux fois.

```
#include "user.h" // user.h copie le contenu de string.h
#include "file.h" // file.h copie le contenu de string.h
// string.h est copié deux fois
```

Solution

- Quand on crée un fichier qui sera inclus, on s'assure que son contenu ne sera utilisé qu'une seule fois

```
// fichier user.h
// le code est inclus seulement si la variable USER_H
n'existe pas
#ifndef USER_H
#define USER_H // on défini la variable au premier passage
...  // on insère le code du fichier ici
#endif
```

Mise en pratique

```
printf("
          _________
        < TP Final >
          --------
                 \   ^__^
                  \  (oo)\_______
                     (__)\       )\/\
                         ||----w |
                         ||     ||
        ")
```