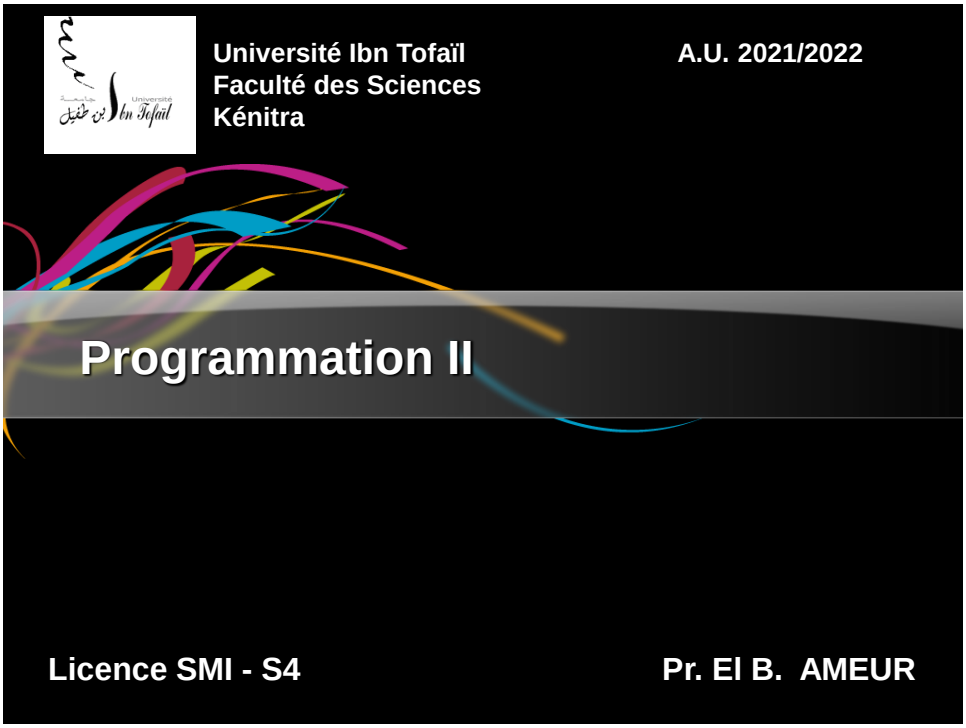

Plan du cours

Partie 1: Rappels et compléments du langage C

1. Les types composés
2. Les pointeurs
3. Les fonctions et la récursivité
4. Les fichiers

Partie 2: Implémentation des Types de Données Abstraits en C

5. Les listes chainées
6. Les piles
7. Les files
8. Les arbres

Support du cours en ligne

Google Classroom:

Programmation II

Utilisez votre émail institutionnel

Code d'accès:

s6yy2mw

3

Partie 1 : Rappels et compléments du langage C

1. LES TYPES COMPOSÉS

5

1.1 Introduction

- A partir des types prédéfinis du langage **C** (caractères, entiers, flottants), on peut créer de nouveaux types, appelés types composés :
  - Tableau
  - Structure
  - Union
  - Enumération, ...
- Les types composés permettent de représenter des ensembles de données organisées.

6

1.2 Les tableaux

- **Tableau :** ensemble fini d'éléments de même type, stockés en mémoire à des adresses contiguës.
- **Déclaration** : tableau à une dimension :

```
Type nomTableau[nombreElements];
```

- **Exemple :**

```
float tabR[5];   /* tableau de 5 flottants (réels). */
int tabE[8];     /* tableau de 8 entiers.   */
```

7

1.2 Les tableaux

- **Recommandation :** Donner un nom à la constante **nombreElements** par une directive au préprocesseur.

- **Exemple :**

```
#define nombreElements 10
```

- Les éléments d'un tableau sont toujours numérotés de **0** à **nombreElements-1**.
- L'opérateur **[ ]** permet l'accès à un élément du tableau.

8

1.2 Les tableaux

- **Exemple :**

```
#define N 10
main(){
  int tab[N];
  int i;
  ...
  for (i = 0; i < N; i++)
    printf("tab[%d] = %d\n", i, tab[i]);
}
```

9

1.2 Les tableaux

- Un **tableau** correspond à un **pointeur constant** (*voir les pointeurs*) vers le premier élément du tableau.
  - ➔ aucune opération globale n'est autorisée sur un tableau.

```
tab1 = tab2;
```

  - ➔ Effectuer l'affectation pour chacun des éléments du tableau :

```
#define N 10
main(){
  int tab1[N], tab2[N];
  int i;
  ...
  for (i = 0; i < N; i++)
    tab1[i] = tab2[i];
}
```

10

1.2 Les tableaux

- On peut initialiser un tableau lors de sa déclaration par une liste d'éléments :
- **Exemple :**

```
#define N 4
int tab[N] = {11, 22, 33, 44};
main(){
  int i;
  for (i = 0; i < N; i++)
    printf("tab[%d] = %d\n", i, tab[i]);
}
```

```
tab[0] = 11
tab[1] = 22
tab[2] = 33
tab[3] = 44
```

11

1.2 Les tableaux

- **Une Chaîne de caractères = Un tableau de caractères avec le caractère '\0' à la fin.**
- **Initialisation :**

```
char p1[10] = {'B','o','n','j','o','u','r','\0'};
char p2[10] = "Bonjour";   /* init. par une chaîne litterale */
char p3[] = "Bonjour";     /* p3 aura alors 8 éléments       */
```

Le compilateur rajoute toujours un caractère '\0' à la fin d'une chaîne de caractères.
➔Il faut donc que le tableau ait au moins un élément de plus.

12

1.3 Les tableau à n-dimensions

- On peut déclarer un tableau à plusieurs dimensions.
- **Exemple :** Tableau à 2 dimensions

```
Type  nomTableau [nombreLignes][nombreColonnes];
```

- Tableau à 2D = Tableau 1D de tableau 1D.
- L'accès à un élément : **`nomTableau[i][j]`**

13

1.3 Les tableau à n-dimensions

- On peut initialiser un tableau à plusieurs dimensions à la compilation par une liste dont chaque élément est une liste de constantes.

```
#define M 2
#define N 3
int tab[M][N] = {{1, 2, 3}, {4, 5, 6}};
main(){
  int i, j;
  for (i = 0 ; i < M; i++){
      for (j = 0; j < N; j++)
        printf("%d\t", tab[i][j]);
    printf("\n");
  }
}
```

```
1       2       3
4       5       6
```

14

1.4 Les types énumérés

- Permettent de définir un type par une liste de valeurs qu'il peut prendre.
- Défini par le mot clé **enum** et un identificateur de modèle, suivis d'une liste de valeurs.

```
enum modele {constante1, constante2, ..., constanten};
```

- Les valeurs possibles **constante1**, **constante2**, **...**, **constanten** sont codées par des entiers de **0** à **n-1**.
- **Exemple :** (Type **booleen**)

```
enum booleen {faux, vrai};
main(){
  enum booleen b;
  b = vrai;
  printf("b = %d", b);
}
```

```
b = 1
```

15

1.4 Les types énumérés

- On peut modifier le codage par défaut des valeurs de la liste d'une énumération lors de la déclaration.
- **Exemple :**

```
enum booleen {faux = 12, vrai = 23};
```

16

1.5 Les structures

- **Structure :** agrégat de plusieurs objets de types différents regroupés dans une même variable.
- Une donnée composant une structure = un champ ou attribut
- **Un champ** peut être de types quelconques
  - Type simple,
  - Tableaux,
  - Autres structures,
  - …
- Les champs possèdent un identificateur qui permet d'accéder directement à la donnée qu'ils contiennent.

17

1.5 Les structures

- **Exemple :**

  Pour un **étudiant**, on peut regrouper dans une seule variable :
  - son **nom** (chaîne de caractères),
  - son **prénom** (chaîne de caractères),
  - son **âge** (entier),
  - son **sexe** (masculin ou féminin),
  - son **numéro CNE** (tableau de 10 chiffres), ...

18

1.5 Les structures

- Déclaration d'un **modèle** de structure

```
struct modele{
  type1 membre1;
  type2 membre2;
    ...
  typen membren;
};
```

- Déclaration d'une **variable** de type structure :

```
struct modele objet;
```

- Ou bien :

```
struct modele{
  type1 membre1;
  type2 membre2;
    ...
  typen membren;
}objet;
```

19

1.5 Les structures

- **Exemples:**
  - Une structure de nom **Etudiant** correspondant à un étudiant :

```
enum sexes {Feminin, Masculin};

struct Etudiant {
    char nom[20], prenom[20];
    int age;
    enum sexes sexe;
    int numero[5];
} element = {"Aissaoui", "Ali", 22, Masculin, {02,49,11,39,42}};
```

20

1.5 Les structures

- **Exemples:**
  - Une structure **Point** correspondant à un point de coordonnées **x**, **y** dans un plan.

```
struct Point {
      float x;
      float y;
};
```

  - Une structure de nom **Adresse** possible pour coder une adresse postale :

```
 struct Adresse {
      int num;
      char rue[40];
      long code;
      char ville[20], pays[20];
};
```

21

1.5 Les structures

- **Exemples:**
  - Une structure de nom **Individu** dont:
    - l'un des champs a la structure **Adresse** précédente (dont la définition est supposée connue)
    - et contenant la définition d'une **structure interne anonyme** pour le champ de nom **Identite** :

```
struct Individu{
       struct {
         char nom[20];
         char prenom[20];
       } Identite;
       int age;
       struct Adresse domicile;
};
```

22

1.5 Les structures

- Le C permet de créer des alias sur une structure avec **typedef** :

```
typedef struct {
    int a, b;    // a et b dans la même déclaration
} couple;
```

- On accède aux différents membres (champs) d'une structure grâce à l'opérateur membre de structure, noté "."

```
nomObjet.nomChamp
```

23

1.5 Les structures

- **Remarque :**
  - L'affectation globale au moyen de l'opérateur **=** entre deux objets de même structure est maintenant autorisée par la plupart des compilateurs.

```
struct Complexe z1, z2;
...
z2 = z1;
```

  - La comparaison (**==**) n'est par contre pas admise généralement. Il est nécessaire de tester chacun des champs l'un après l'autre.

- Les règles d'initialisation d'une structure lors de sa déclaration sont les mêmes que pour les tableaux:

```
struct Complexe z = {2. , 2.};
```

24

1.5 Les structures

- **Exemple 1 :**

```c
#include <math.h>
struct Complexe{
    double reelle;
    double imaginaire;
};
main(){
    struct Complexe z = {3.5, 2.9};
    double norme;
    norme = sqrt(z.reelle * z.reelle +
                 z.imaginaire * z.imaginaire);

    printf("La norme de (%.2f + i %.2f) = %.2f\n",
                               z.reelle, z.imaginaire, norme);
}
```

```
La norme de (3.50 + i 2.90) = 4.55
```

25

1.5 Les structures

- **Exemple 2 :**

```c
struct Personne {
                char CIN[8];
                char nom[10];
                int age;
};
int main(){
       struct Personne P1;
       strcpy (P1.nom, "Hamid");
       strcpy (P1.CIN, "A123");
       P1.age = 21;
       printf ("%s %s %d\n",P1.CIN, P1.nom, P1.age);
       return 0;
   }
```

```
A123 Hamid 21
```

1.5 Les structures

- **Exemple 3 :**

```
enum propulsion {pedales, moteur, reacteur};
struct vehicule {
    char nom[20];
    int longueur, poids;
    enum propulsion mode;
} velo = {"Euroteam", 2, 5, pedales};
main () {
    struct vehicule voiture ={"Toyota", 5, 1500, moteur };

    struct vehicule avion;
    strcpy(avion.nom, "Jumbo");
    avion.longueur = 60;
    avion.poids = 450000;
    avion.mode = reacteur;

    printf ("Nom: %s\n", velo.nom);
    printf ("Longueur: %d, poids: %d\n", velo.longueur, velo.poids);
    printf("Mode de propulsion: %d", velo.mode);printf ("\n");
    printf ("Nom: %s\n", avion.nom);
    printf ("Longueur: %d, poids: %d\n",avion.longueur, avion.poids);
    printf("Mode de propulsion:%d\n", avion.mode);
}
```

1.5 Les structures

- **Exemple 3 :**

```
Nom: Euroteam
Longueur: 2, poids: 5
Mode de propulsion: 0
Nom: Jumbo
Longueur: 60, poids: 450000
Mode de propulsion:2
```

1.6 Les champs de bits

- Possible de spécifier la longueur des champs d'une structure au bit près si ce champ est de type entier.
- **Exemple :**
  - La structure Registre possède deux membres, **actif** codé sur un seul bit, et **valeur** codé sur 31 bits:

```
struct Registre{
  unsigned int actif : 1;
  unsigned int valeur : 31;
};
```

- Tout objet de type **struct Registre** est donc codé sur 32 bits.

29

1.7 Les unions

- **union** désigne un ensemble de variables de types différents susceptibles d'occuper **alternativement** une même zone mémoire.

- Si les membres d'une **union** sont de longueurs différentes, la place réservée en mémoire pour la représenter correspond à la taille du membre le plus grand.

- Déclarations et opérations sur **union** sont les mêmes que **struct**.

30

1.7 Les unions

- **Exemple :**
  - La variable **hier** de type **union Jour** peut être soit un entier, soit un caractère.

```
union Jour{
    char lettre;
    int numero;
};
main(){
    union Jour hier, demain;
    hier.lettre = 'J';
    printf("Hier = %c\n", hier.lettre);
    hier.numero = 4;
    demain.numero = (hier.numero + 2) % 7;
    printf("Demain = %d\n", demain.numero);
}
```

```
Hier = J
Demain = 6
```

31

1.8 Définition de types composés avec typedef

- Pour alléger l'écriture, on peut affecter un nouvel identificateur à un type composé à l'aide de **typedef** :
- **Exemple 1 :**

```
struct Eleve{
    char nom[20];
    char prenom[20];
    float note;
};
typedef struct Eleve Eleve;
main(){
  Eleve a = {"Alaoui", "Hamza", 17}, b;
  strcpy(b.nom, "Hatimi");
  strcpy(b.prenom, "Bouchra");
  b.note = 13;
  printf("Eleve 1 : %s %s %.2f\n", a.nom, a.prenom, a.note);
  printf("Eleve 2 : %s %s %.2f\n", b.nom, b.prenom, b.note);
}
```

```
Eleve 1 : Alaoui Hamza 17.00
Eleve 2 : Hatimi Bouchra 13.00
```

32

1.8 Définition de types composés avec typedef

- **Exemple 2 :**

```
struct Eleve{
    char nom[20];
    char prenom[20];
    float note;
};
typedef struct Eleve Eleve;
typedef Eleve TabEleve[100];
main(){
    TabEleve T;    int i;
    printf("------ Saisie ------\n");
    for(i=0;i<3;i++){
        printf("Eleve %d : ", i+1);
        scanf("%s %s %f", T[i].nom, T[i].prenom, &T[i].note);
    }
    printf("------ Affichage ------\n");
    for(i=0;i<3;i++)
        printf("Eleve %d : %s %s %.2f\n",
                                    i+1, T[i].nom, T[i].prenom, T[i].note);
}
```

```
------ Saisie ------
Eleve 1 : Hachimi Souad 16
Eleve 2 : Alami Brahim 13
Eleve 3 : Hamdi Merieme 14
------ Affichage ------
Eleve 1 : Hachimi Souad 16.00
Eleve 2 : Alami Brahim 13.00
Eleve 3 : Hamdi Merieme 14.00
```

Exercices

Exercice 1

Déclarer **un type enregistrement Produit** contenant les informations suivantes:

- Num(entier)
- Nom(chaîne de caractères)
- Prix(réel)
- Quantite(entier)

Exercices

Exercice 2

Déclarer **une variable P de type Personne** ayant les informations suivantes:

- Nom (chaîne de caractères)
- Prénom (chaîne de caractères)
- Date de naissance:
  1. Jour (entier)
  2. Mois (entier)
  3. Année (entier)
- Code(chaîne de caractères)

- Remplir P avec les informations suivantes: Nom:"Amir" Prénom:"Salem" Date de naissance: 03/04/2005 Code: "A32"

35

2 LES POINTEURS

36