# Langage procédural

## Le langage C

Nizar OUARTI

Laboratoire ISIR (email: ouarti@isir.upmc.fr)

2012

# Problèmes avec les données

- Problème : On voudrait pouvoir gérer le dossier d'un patient
- Différentes variables composent son dossier
- Son nom, son prénom, son groupe sanguin, s'il est marié ou non, son sexe, son age, le nombre d'enfants qu'il a, le numéro téléphone du conjoint
- On voudrait aussi gérer son taux de globule rouge et blanc dans le sang, ainsi que le taux de plaquettes
- Comment représenteriez-vous ces données dans l'ordinateur ?

# Problèmes avec les données

- Problème : On voudrait pouvoir gérer le dossier d'un patient
- Différentes variables composent son dossier
- Son nom, son prénom, son groupe sanguin, s'il est marié ou non, son sexe, son age, le nombre d'enfants qu'il a, le numéro téléphone du conjoint
- On voudrait aussi gérer son taux de globule rouge et blanc dans le sang, ainsi que le taux de plaquettes
- Comment représenteriez-vous ces données dans l'ordinateur ?
- Ecrivez un algorithme qui détecte si un patient est une femme et si elle a un taux de plaquette en dessous d'un seuil donnée et si son âge est supérieur à 65 ans alors on lance une alerte en précisant son identité et en pensant à joindre le conjoint.

# Problèmes avec les données

- Problème : On voudrait pouvoir gérer le dossier d'un patient
- Différentes variables composent son dossier
- Son nom, son prénom, son groupe sanguin, s'il est marié ou non, son sexe, son age, le nombre d'enfants qu'il a, le numéro téléphone du conjoint
- On voudrait aussi gérer son taux de globule rouge et blanc dans le sang, ainsi que le taux de plaquettes
- Comment représenteriez-vous ces données dans l'ordinateur ?
- Ecrivez un algorithme qui détecte si un patient est une femme et si elle a un taux de plaquette en dessous d'un seuil donnée et si son âge est supérieur à 65 ans alors on lance une alerte en précisant son identité et en pensant à joindre le conjoint.
- Que feriez-vous si vous aviez 200 patients à gérer ?

# Les types composés

- **Types composés** : regroupement de **types élémentaires** (étudiés précédemment)
- Pourquoi regrouper les données ?

# Les types composés

- **Types composés** : regroupement de **types élémentaires** (étudiés précédemment)
- Pourquoi regrouper les données ?
- Pour mieux les structurer

# Les types composés

- **Types composés** : regroupement de **types élémentaires** (étudiés précédemment)
- Pourquoi regrouper les données ?
- Pour mieux les structurer
- Pour obtenir plus de clarté dans le code

# Les types composés

- **Types composés** : regroupement de **types élémentaires** (étudiés précédemment)
- Pourquoi regrouper les données ?
- Pour mieux les structurer
- Pour obtenir plus de clarté dans le code
- Pour obtenir un code plus synthétique (encapsulation)

# Les types composés

- **Types composés** : regroupement de **types élémentaires** (étudiés précédemment)
- Pourquoi regrouper les données ?
- Pour mieux les structurer
- Pour obtenir plus de clarté dans le code
- Pour obtenir un code plus synthétique (encapsulation)
- Faciliter la manipulation

# Les types composés

- **Types composés** : regroupement de **types élémentaires** (étudiés précédemment)
- Pourquoi regrouper les données ?
- Pour mieux les structurer
- Pour obtenir plus de clarté dans le code
- Pour obtenir un code plus synthétique (encapsulation)
- Faciliter la manipulation
- Aller vers un gestion automatisée

# Différents Types Composés

- **Les structures**
- **Les tableaux**
- **Les unions**
- **Les énumérations**

# Les structures

- Mot clé **struct**
- Une structure a un nom
- Une structure possède différents champs
- Une structure réunit des champs de types différents
- Une structure peut contenir un tableau, voire une structure
- Les champs sont contigus en mémoire

# Les structures

- Mot clé **struct**
- Une structure a un nom
- Une structure possède différents champs
- Une structure réunit des champs de types différents
- Une structure peut contenir un tableau, voire une structure
- Les champs sont contigus en mémoire

```
1 struct voiture  // nom de la structure
2 {
3         char marque[20];          //Champs 1 (tableau)
4         bool neuve;               //Champs 2
5         int nbr_kilometre;        //Champs 3
6         struct moteur diesel;     //Champs 4 (structure)
7 };
```

# structure patient

- Ecrivez une structure patient tel que décrite précédemment

# structure patient

- Ecrivez une structure patient tel que décrite précédemment

```
1  struct patient //description d'un patient
2  {
3          char nom[20];
4          char prenom[20];
5          char groupe_sanguin[3];
6          bool mariage;
7          bool homme;
8          int age;
9          int nbr_enfant;
10         float taux_glob_rouge;
11         float taux_glob_blanc;
12         float taux_plaquette;
13
14 };
```

# Définition, initialisation et utilisation de structures

- Accédez aux champs de cette structure pour résoudre le problème qui avait été posé
- Après la définition de **struct patient** on peut écrire :

```
1 //premier type de déclaration
2 struct patient patnum1;// déclaration sans
      initialisation
3 //deuxième type de déclaration
4 struct patient patnum1{"Charles","Dupont","AB+",true,
      true,42,2,0.2,0.1,0.001};// initialisation à la
      déclaration
5 patnum1.age=43; //affectation d'une valeur au champs
      âge appartenant à patnum1
```

# Les tableaux

- Pas de mot clé mais utilisation de [ et ]
- Un tableau a un nom
- Un tableau réunit des objets de types indentiques

# Les tableaux

- Pas de mot clé mais utilisation de [ et ]
- Un tableau a un nom
- Un tableau réunit des objets de types indentiques

```
1 char marque[20];          //tableau de char contenant une
      chaine de caractère: 20 éléments de 0 à 19
2 int results_loto[8];      //tableau de int contenant les
     chiffres du LOTO: 8 éléments de 0 à 7
3 float coordonnes[3];      //tableau de float contenant
     les coordonnées 3D d'un point: 3 éléments de 0 à 2
4 bool succes[5];           //tableau de bool contenant les
      succès et échecs à un pari: 5 éléments de 0 à 4
```

# Affectation

- int tab[4] ;tab[1]=5 ; avec 1 l'indice du tableau (2ième élément)
- Problème fréquent : choisir un indice plus petit que zéros ou supérieur à la taille du tableau
- Comment affecter toutes les valeurs d'un tableau de taille 1000 sans avoir à écrire 1000 lignes ?

# Affectation

- int tab[4] ;tab[1]=5 ; avec 1 l'indice du tableau (2ième élément)
- Problème fréquent : choisir un indice plus petit que zéros ou supérieur à la taille du tableau
- Comment affecter toutes les valeurs d'un tableau de taille 1000 sans avoir à écrire 1000 lignes ?
- Utilisation fréquente : les tableaux sont souvent affectés dans une boucle for

# Usage

- Créer une variable contenant un numéro de téléphone

# Usage

- Créer une variable contenant un numéro de téléphone
- char num[10] ;

# Usage

- Créer une variable contenant un numéro de téléphone
- char num[10] ;
- Nous voudrions créer une variable contenant les coordonnés en 3D de 100 points

# Usage

- Créer une variable contenant un numéro de téléphone
- char num[10] ;
- Nous voudrions créer une variable contenant les coordonnés en 3D de 100 points
- Solution : **tableaux à 2 dimensions** (matrice)
- Syntaxe : float points[3][100] ;

- Nous voudrions créer une liste de 30 patients.

- Nous voudrions créer une liste de 30 patients.
- Solution : **Tableaux de structures**
- Syntaxe : struct patient patn[30] ;

# Les tableaux de caractère

- Les tableaux de caractères sont appelés chaînes de caractères
- Comme on les manipule souvent, des fonctions sont spécialement faite pour les exploiter
- la fonction printf() sert à afficher une chaine de caractère et à formater les sorties texte

```
1 printf("Son nom est %s, il a %d ans et son taux de
      plaquettes est %f \n", patn[2].nom,patn[2].age,patn
      [2].taux_plaquette);
2 // Permet l'affichage /n correspond à retour à la ligne
      , %s correspond à un 'string', chaine de caractère;
       %d Affiche un entier; %f pour afficher un réel.
3
4 scanf("%s %d  %f ", patn[2].nom,&(patn[2].age),&(patn
      [2].taux_plaquette));// Lit les entrées au clavier
```

# Les union

- Mot clé **union**
- Une union a un nom
- Une union possède différents membres
- Une union juxtapose des membres de types différents
- Ils correspondent au même espace mémoire
- La taille d'une union correspond à la taille de l'objet de plus grande taille

# Les union

- Mot clé **union**
- Une union a un nom
- Une union possède différents membres
- Une union juxtapose des membres de types différents
- Ils correspondent au même espace mémoire
- La taille d'une union correspond à la taille de l'objet de plus grande taille

```
1 union data        // nom de l'union
2 {
3         int discret;              //Membre 1
4         float continu;            //Membre 2
5 };
```

# Principe de l'union

- On va stocker dans un même espace mémoire une variable qui peut être de différent type
- union data taille ;
- taille.discret=3 ; et taille.continu=3.55 ; pas d'erreur
- Pourtant en mémoire la place qui a été allouée n'est pas la place pour deux variable contrairement à structure
- La place allouée en mémoire est de la taille d'un float dans notre exemple
- La zone mémoire à un instant précis n'est occupée que par 1 membre
- ex int taille1 ;taille.continu=3.55 ;taille1=taille.continu ; est déconseillé
- Lors d'une affectation l'autre membre disparait ! ici taille1=taille.discret n'existe plus

# Les énumération

- Mot clé **enum**
- Une énumération a un nom
- Une énumération possède différents identificateurs
- Ces identificateurs sont des constantes

# Les énumération

- Mot clé **enum**
- Une énumération a un nom
- Une énumération possède différents identificateurs
- Ces identificateurs sont des constantes

```
1  // affectation numérique explicite
2  enum jour
3  {
4          lundi=0,          //#define lundi 0
5          mardi=1,          //#define mardi 1
6          mercredi=2,       //#define mercredi 2
7          jeudi=3,          //#define jeudi 3
8          vendredi=4,       //#define vendredi 4
9          samedi=5,         //#define samedi 5
10         dimanche=6        //#define dimanche 6
11 };
```

# Les énumération

- Possibilité de définition implicite

# Les énumération

- Possibilité de définition implicite

```
1  // affectation numérique explicite
2  enum jour
3  {
4          lundi,              //#define lundi 0 par défaut
5          mardi=50,           //#define mardi 50
6          mercredi,           //#define mercredi (50+1)
7          jeudi,              //#define jeudi (51+1)
8          vendredi=2,         //#define vendredi 2
9          samedi,             //#define samedi (2+1)
10         dimanche            //#define dimanche (3+1)
11 };
```

# usage des énumérations

- Equivalent à une déclaration de constante avec #define

# usage des énumérations

- Equivalent à une déclaration de constante avec #define

```
1 enum jour j1;
2 enum jour j2;
3
4 int n;
5
6 n = mardi;          //  n sera égal à 1
7 j1=jeudi;          // j1 sera égal à 3
8 j2=j1;             // j2 sera égal à 3
```

# Définition de type

- Mot clé **typedef**
- Sert à raccourcir l'écriture des définitions
- Sert à mieux structurer son code
- Sert à se passer des mots clé struct union enum lors des déclarations

# Définition de type

- Mot clé **typedef**
- Sert à raccourcir l'écriture des définitions
- Sert à mieux structurer son code
- Sert à se passer des mots clé struct union enum lors des déclarations

```
1 typedef struct patient Patient_type;
2 Patient_type patnum1;
3 //Avant on écrivait:
4 struct patient patnum1;
```

# Définition de type

- Une autre méthode est possible :

# Définition de type

- Une autre méthode est possible :

```
1  typedef struct Newtype_
2  {
3      int data1;
4      int data2;
5      char data3;
6  } Newtype;          // nom du nouveau type
7
8  // usage
9
10 Newtype a; // très simple à déclarer
```