Le langage C - Nouveaux types de données

16 février 2010

Enumération

Principe

- Une variable qui ne prend qu'un nombre fini de valeurs. Exemples :
  - booléen : vrai, faux
  - feu tricolore : rouge, jaune, vert
  - jours : lundi, mardi, mercredi...
- L'utilisation d'une énumération permet une écriture plus claire du code

Enumération

Enumération

```
enum nom_enum {
    ....
};

enum nom_enum variable;
```

Enumération

Exemple

```
enum feuTriclore {
  ROUGE,
  JAUNE,
  VERT
};

enum feuTriclore feu1, feu2;
feu1 = VERT;
feu2 = rand()%3;
if (feu1==VERT && feu2==VERT) {
    ...
}
```

Enumération

Enumération = liste de constantes

- Affectation d'une valeur à chaque constante de l'énumération
- Par défaut, premier = 0 puis incrémentation de 1 pour les autres
- Possibilité d'imposer d'autres valeurs

Enumération

Exemple

```
enum feuTriclore {
    ROUGE = 12,
    JAUNE,
    VERT
};
// ROUGE=12, JAUNE=13, VERT=14
```

Exemple

```
enum feuTriclore {
    ROUGE = 120,
    JAUNE = 130,
    VERT = 140
};
```

Structure

Principe

- Regrouper au sein d'une seule variable plusieurs variables de types différents.
  Exemples :
  - Personne : nom, prenom, age, tel, cp, ville
  - Complexe : réel, imaginaire
  - **Noeuds d'une liste chaînée : valeur, suivant**

Structure

Structure

```
struct nom_struct {
   type champ1;
   type champ2;
   ....
};

struct nom_struct variable;
```

Structure

Exemple

```
struct cmplx {
   float re;
   float im;
   };

struct cmplx a, b;
```

Structure

Opérations

- Accès à un champ :

```
a.re = 3.0;
a.im = -2.5;
```

- Affectation directe possible (ou copie champ par champ)

```
b = a; // b.re = a.re, b.im = a.im
```

- Comparaison directe impossible

```
if (a == b) ... // ne marche pas !
if (a.re==b.re && a.im==b.im) ...
```

- `sizeof(a)` renvoit la taille en octets de la structure (= la somme des tailles des champs impliqués)

Structure

Composition de structures

```
struct personne {
  char nom[30];
  char prenom[20];
  int age;
};

struct livre {
  struct personne auteur;
  char titre[50];
  char editeur[20];
  int annee;
};

struct livre L;
L.annee = 1994;
L.auteur.age = 45;
```

Question

Liste chaînée

Comment représenter, grâce au concept de structure, un noeud de liste chaînée ?

Réponse

Liste chaînée

```
struct Noeud {
  int valeur;
  struct Noeud* suivant;
};
```

Structure

Pointeurs de structure

- Déclaration identique aux autres types :
  struct cmplx* p ;
- Accès aux champs :
  (*p).re = 1.0 ;
  (*p).im = 2.0 ;
  ou bien
  p->re = 1.0 ;
  p->im = 2.0 ;
- Attention : -> n'est valide que pour les pointeurs !

Union

Principe

- Il est parfois nécessaire de manipuler des variables auxquelles on désire affecter des valeurs de type différent.
- Une union est constituée de champs qui peuvent être de types différents (élémentaires ou définis par l'utilisateur, y compris énumération, structure ou union...)
- Une union permet de réunir plusieurs éléments de taille différente dans un même espace, et de n'utiliser que la taille du plus grand de tous ces éléments.

Union

Union

```
union nom_union {
   type champ1;
   type champ2;
   ....
};

union nom_union variable;
```

Union

Exemple

```
union entierOuFlottant {
   int   n;
   float x;
};

union entierOuFlottant a, b;
```

Union

Opérations

- Accès à un champ :

```
a.n = 3;
a.x = -2.5; // ECRASE a.n!!
```

- Affectation directe possible

```
b = a;
```

- `sizeof(a)` renvoit la taille en octets de l'union (= la plus grande des tailles des champs impliqués)

Union

Problème

On n'a malheureusement aucun moyen de savoir à un instant donné, quel est le membre de l'union qui possède une valeur.

Solution

Une union doit donc toujours être associée à une variable dont le but sera d'indiquer le membre de l'union qui est valide. En pratique, une union et son indicateur sont généralement englobés à l'intérieur d'une structure.

Union

Exemple

```
enum typeNombre { ENTIER, FLOTTANT };
union entierOuFlottant {
  int   n;
  float x;
};
struct entierOuFlottant
{
  enum typeNombre type;
  union entierOuFlottant nombre;
};
struct entierOuFlottant a1,a2;
a1.type = ENTIER;
a1.nombre.n = 10;
a2.type = FLOTTANT;
a2.nombre.x = 3.14159;
if (a1.type==ENTIER) ... // consulter a1.nombre.n
else ... // consulter a1.nombre.x
```

Types équivalents

Principe

Le mot clé `typedef` permet de définir des types synonymes dans le but de rendre les programmes plus clairs.

Déclaration

typedef type type_equiv ;

Types équivalents

Exemple

```
typedef float flottant;
typedef int[3] vecteur;

flottant x = 4.67;
vecteur v;
v[0] = 1;
v[1] = 5;
v[2] = 4;
```

Types équivalents

typedef sur enum, struct, union

`typedef` permet en particulier d'alléger la syntaxe d'une énumération, d'une structure et d'une union.

Exemple

```
struct cmplx {
  float re;
  float im;
};
typedef struct cmplx cmplx;

cmplx a, b;
```

Types équivalents

typedef sur enum, struct, union

Il existe la possibilité d'utiliser `typedef` dès la définition d'une énumération, d'une structure et d'une union.

Exemple

```
typedef struct {
  float re;
  float im;
} cmplx;

cmplx a, b;
```

Types équivalents

Exemple

```
typedef enum {
  FALSE,
  TRUE
} boolean;

boolean b = TRUE;
```

Types équivalents

Utilisation pour les listes chaînées

Maintenant nous connaissons tous les éléments nécessaires pour définir et utiliser une liste chaînée en C.

Notre définition du type noeud

```
typedef struct Noeud {
  int valeur;
  struct Noeud* suivant;
} Noeud;

// Exemple de création d'un noeud
Noeud* p = (Noeud*)malloc(sizeof(Noeud));
p->valeur = 6;
p->suivant = NULL;
```

Longueur d'une liste

Pseudocode

```
Fonction longueur (teteListe : Pointeur) : Entier
Variables l : Pointeur, n : Entier
Début
  l ← teteListe
  n ← 0
  Tantque l ≠ null
    n ← n + 1
    l ← suivant(l)
  FinTantQue
  Retourner n
Fin
```

Longueur d'une liste

Code C

```
int longueur (Noeud* teteliste) {
  Noeud* l = teteListe;
  int n = 0;
  while (l != NULL) {
    n++;
    l = l->suivant;
  }
  return n;
}
```

Ajouter en tête de liste

Pseudocode

```
Fonction ajouterTete (teteListe : Pointeur, x : Entier) :
Pointeur
Variables p : Pointeur
Début
  p ← creernoeud()
  valeur(p) ← x
  suivant(p) ← teteListe
  Retourner p
Fin
```

Ajouter en tête de liste

Code C

```
Noeud* ajouterTete (Noeud* teteListe, int x) {
  Noeud* p = (Noeud*)malloc(sizeof(Noeud));
  p->valeur = x;
  p->suivant = teteListe;
  return p;
}
```

Exercices

Exercices

Ecrire en C les fonctions

- ajouterFin (ajoute un noeud en fin de la liste)
- supprimerTete (supprime le premier noeud de la liste)