IFT159
Analyse et programmation
Chapitre 9 — Organisation des données et types

Gabriel Girard

Département d'informatique
UNIVERSITÉ DE SHERBROOKE

22 septembre 2008

Chapitre 9 — Organisation des données et types

Avant-propos

- Nous avons vu l'organisation du traitement.
- Nous devons voir l'organisation des données.
- L'organisation des données permet de regrouper des données afin de simplifier et clarifier un problème.
- Exemple : les tableaux.

Utilisations du préprocesseur

- Macros — Remplacement de *id* par un autre *texte*
  `#define` *id texte_de_remplacement*
- Exemples
  ```
  #define VRAI 1
  #define ET &&
  ```

Représentation interne

- `int` → 32 bits (sur SPARC).
  max = 2147482647, min = -2147482648
- `float` → 32 bits.
  $|max| \simeq 1 * 10^{38}$,
  $|min| \simeq 1 * 10^{-37}$
  précision ≃ 6 nombres décimaux
- `short int` → 16 bits.
  max = 32767, min = -32768
- `long int = int`

Représentation interne

- `signed int = int` (par défaut)
- `unsigned short int` → 16 bits
  max = 65535, min = 0
- `unsigned int` → 32 bits
  max = 4 294 967 295, min = 0
- `unsigned long int` → 32 bits
  max = 4 294 967 295, min = 0

Représentation interne

- `long long int` → 64 bits
  max = $2^{63} - 1$, min = -2^{63}
- `unsigned long long` → 64 bits
  max = 2^{64}, min = 0
- `char` → 8 bits
  max = 127, min = -128
- `unsigned char` → 8 bits
  max = 255, min = 0

Représentation interne

- `double` → réel de 64 bits
  $|max| \simeq 1 * 10^{308}$,
  $|min| \simeq 1 * 10^{-307}$
  précision ≃ 15 nombres décimaux
- `long double` → réel de 128 bits
  $|max| \simeq 1 * 10^{4932}$,
  $|min| \simeq 1 * 10^{-4931}$
  précision ≃ 33 nombres décimaux (sur SPARC)
  précision ≃ 18 nombres décimaux (sur PC)

Important

- Les dimensions sont dépendantes de l'implémentation. Les seules contraintes existantes sont
  16 bits $\leq$ short
  short $\leq$ int $\leq$ long
  long $\geq$ 32 bits
  float $<$ double $<$ long double
- Les `float` sont précis sans l'être.
- `int / int` et `int % int`.

Représentation interne

- L'ordinateur ne travaille pas en décimal (en base 10). Les nombres décimaux doivent donc être convertis
- Cela entraîne une perte de précision car un nombre décimal en base 10 ne possède pas nécessairement un équivalent en base 2.
- Avec les `float`
  - *infinity* signifie division par 0
  - *NaN* est le résultat d'une opération arithmétique illégale.

Représentation interne

- Comment arrondir à la n^{ieme} décimale :
  `floor` $((nbr * 10^n + 0.5))/10.0^n$
- Prudence avec la conversion de type implicite.

Représentation interne

- Chaîne de caractères
  Une chaîne de caractères est une suite de caractères terminée par une sentinelle (NULL).

C =

b	o	n	j	o	u	r	\0	/	/	/	/

Représentation interne

- On peut se servir des tableaux pour emmagasiner les chaînes de caractères (ou le type « string»).
- En entrée, `cin` transforme le « blanc» (espace) ou « retour de chariot» en NULL.
- Pour qu'une suite de caractères soit considérée comme une chaîne par les opérations tel `cout` vous devez explicitement y ajouter un NULL.

Représentation interne

- Manipulation d'une chaîne de caractères
  Pour manipuler une chaîne, on traverse le tableau jusqu'à une valeur NULLE.
- Exemple

```
for(int i=0 ; C[i] != NULL ; i++)
            traiter_car(C[i]) ;
```

- NULL est défini dans <`string`> ou par \0.
- Manipulation de caractères :<`cstring`>
  (islower, isupper, tolower, toupper, isdigit, strcpy, strcat, strcmp, strlen, ...).

Création de nouveau types

- Les types énumérés (`enum`)
- Les types enregistrements (`struct`)

Les types énumérés (enum)

- Servent à rendre le programme plus lisible.
- Syntaxe : `enum` *`nom_type`* {liste de noms} ;
- Exemple :

```
enum type_jour {dimanche, lundi, mardi, mercredi,
                jeudi, vendredi, samedi} ;
type_jour debut_semaine ;
debut_semaine = lundi ;
```

Les types énumérés (enum)

- Ce type associe à chaque valeur un numéro correspondant à sa position (par défaut).
- Ainsi, dimanche = 0, lundi = 1 et samedi = 6. C'est presque l'équivalent de définir sept constantes.
- On peut faire :
  `enum type_longueur_mois {court=28, moyen=30, long};`

Les types enregistrements (struct)

- Un concept important consiste à regrouper des données de types différents sous un même nom.
- On crée donc un agrégat de données appelé de façon générale un tel agrégat un enregistrement.
- En C++, on utilise le mot-clé `struct` ; en Pascal, on utilise le mot-clé `record`.

Les types enregistrements (struct)

Exemple : On veut un enregistrement contenant de l'information sur les étudiants.

Il contiendra donc :

- matricule (entier)
- nom (chaîne de caractères)
- département (type de département)
- téléphone (chaîne de caractères)

Les types enregistrements (struct)

- En C++, on définit des enregistrements (c'est-à-dire un nouveau type) grâce à l'énoncé `struct`.
- Syntaxe :

  `struct` *nom_type*
  {
  type$_1$ liste de noms ;
  type$_2$ liste de noms ;
  ...
  type$_n$ liste de noms ;
  } ;

- Avec cet énoncé, on définit un nouveau type.

Les types enregistrements (struct)

- Exemple de définition de type :

```
struct etudiant
{
   int   matricule;
   char  nom[20];
   int   departement;
   char  tel[10];
};
```


Les types enregistrements (struct)

- Exemple de déclaration de variables

```
etudiant      etudiant1,etudiant2;
```


Les types enregistrements (struct)

- Exemple de manipulation

```
etudiant1.matricule
etudiant2.matricule
etudiant1.nom

etudiant1.matricule = 123454;
strcpy(etudiant1.nom, "prenom nom");  <---------
cout << etudiant1.nom << " : matricule = "
     << etudiant1.matricule;
```


Les types enregistrements (struct)

- Exemple de définition de type :

```
enum type_dept {bio, chi, inf, mat, phy};
struct type_etudiant
{
    int        matricule;
    string     nom;
    type_dept  departement;
    string     tel;
};
```

Les types enregistrements (struct)

- Exemple de déclaration de variables

```
type_etudiant   etudiant1,etudiant2;
```

- Exemple de manipulation

```
etudiant1.matricule
etudiant2.matricule
etudiant1.nom

etudiant1.matricule = 123454;
etudiant1.nom = "prenom nom";
cout << etudiant1.nom << " : matricule = "
     << etudiant1.matricule;
```


Les types enregistrements (struct)

- Enregistrements hiérarchiques Il est possible d'utiliser une enregistrement dans un autre enregistrement.
- Exemple :

```
enum type_session hiver, automne, ete;
```


Les types enregistrements (struct)

- Exemple :(suite) :

```
struct type_adresse
{
    int    no;
    string rue;
    string ville;
    string province;
    string pays;
    string code_postal;
    string tel;
};
```


Les types enregistrements (struct)

- Exemple (suite) :

```
struct type_notes
{
    int final;
    int intra;
    int travaux[10];
};
```

Les types enregistrements (struct)

- Exemple (suite) :

```
struct type_etudiant
{
   int               matricule;
   string            nom;
   type_departement  departement;
   type_session      arrivee;
   type_adresse      temp, permanente;
   type_notes        cours[NB_COURS];
};
```


Les types enregistrements (struct)

- Exemple (suite) :

```
type_etudiant  etudiant1,etudiant2;

etudiant1.matricule = 1234;

etudiant1.cours[1].intra = 70;
etudiant2.cours[4].travaux[2]=25;

etudiant1.temp = lire_adresse();
etudiant1.permanente = lire_adresse();

etudiant1.arrivee = hiver;
```


Les tableaux d'enregistrements

- On peut bien entendu faire des tableaux ayant pour type un type enregistrement.

  Exemple :

```
etudiant  groupe[100];

groupe[2].matricule = 1234;

groupe[2].cours[3].intra= 60;
```


Nouveaux types et flots de données

- Une variable d'un type énuméré se comporte comme une variable de type entier. Elle peut être l'objet d'un `switch` par exemple.
- Une variable d'un type enregistrement peut se transmettre en paramètre, en valeur de retour, devenir un paramètre de sortie, être déclarée comme constante, etc.
- Attention un enregistrement contenant un grand tableau peut poser des problèmes lors d'un passage de paramètre par valeur (copie mémoire du tableau).

Interface

- On ne peut utiliser directement << ou >> avec un type énuméré ou enregistrement.
- Pour chaque type, il est conseillé de créer :
  - une fonction de création avec des paramètres convenablement choisis ;
  - une fonction lisant des données au clavier et créant un élément du type (utilisant la fonction précédente) ;
  - une fonction de transformation en une chaîne de caractères.
- Ceci améliore la gestion d'une structure complexe pour le programmeur.