Programmation C

Outils syntaxiques non algorithmiques

ING1-GI

CY Tech

2020-2021

Énumerations

Énumerations

Définition

- Variable ne pouvant prendre qu'un nombre fini d'états

Exemple

- Mois : Janvier, Février, ...
- Continents, Trimestre, Sessions, Booléen, ...

- Utilisation des énumérations

Énumerations

Déclaration et utilisation

```c
/* Definition : dans le header */
enum nom_enumeration {
    .../...
};
```

```c
/* Lors de l'utilisation */
enum nom_enumeration nom_variable;
```

- La déclaration de l'énumération se met dans le header
- Elle se termine toujours par un ;
- Les valeurs de l'énumération sont séparées par des **virgules**

Énumerations

Exemple

```c
/* Definition : dans le header */
enum examen {
    Janvier,
    RJanvier,
    Juin,
    RJuin
};
```

```c
/* Lors de l'utilisation */
int main (int argc, char** argv) {
   enum examen enu_periode;
    enu_periode = Juin;
    .../...
    return(0);
}
```

Énumerations

Réalité

- Affectation d'une valeur, à chaque élément de l'énumération
- Premier élément : 0
- Itération pour les autres
- Possibilité de donner une valeur différente
  - de départ
  - pour chaque élément

Énumerations

Exemple

```
enum examen {
    Janvier = 100,
    RJanvier,
    Juin,
    RJuin = 200
};
```

- Janvier vaut 100, RJanvier : 101, Juin : 102
- RJuin : 200
- Possibilité d'utiliser la valeur numérique

Énumerations

Attention

```c
enum examen {
    Janvier = 100,
    RJanvier = 100,
    Juin,
    RJuin = 200
};
```

```c
/* Dans une fonction ... */
toto = 100;
if (toto == Janvier) {
    printf("%d\n", toto);
}
if (toto == RJanvier) {
    printf("%d\n", toto);
}
```

- Affichera deux fois 100

Énumerations

Documentation

```
/*! \enum examen
 *  Enumeration pour les periodes d'examen
 */
enum examen {
   Janvier = 100,   /*!< Examen de janvier */
   RJanvier = 100,  /*!< Rattrapage de janvier */
   Juin,            /*!< Examen de juin */
   RJuin = 200      /*!< Rattrapage de juin */
};
```

- /*!< : permet de spécifier l'utilité du champ

Structures

Structures

Définition

- Permet de regrouper des éléments de natures différentes au sein d'une même variable
  - Exemple : identité (nom, prenom, code, ville, ...)
- Peut aussi servir à regrouper des éléments identiques
  - Exemple : nombre complexe (réelle, imaginaire)
- Notion de champ

Structures

Déclaration et utilisation

```c
/* Definition : dans le header */
struct nom_structure {
   type nom1;
   type nom2;
   ...
};
```

```c
/* Lors de l'utilisation */
struct nom_structure nom_variable;
```

- La déclaration de la structure se met dans le header
- Elle se termine toujours par un ;

Structures

Exemple

```
/*! \struct scomplexe
 * Structure pour la manipulation de nombre complexe
 * \remark Les nombres sont sous la forme \f$a+ib\f$
 */
struct scomplexe {
  double dbl_reel;         /*!< Partie reelle */
  double dbl_imaginaire; /*!< Partie imaginaire */
};
```

```
/* Lors de l'utilisation */
struct scomplexe str_comp_a;
```

Structures

Opération sur une structure

- Accès à un champ : .
  - `stru_comp_a.dbl_reelle=0, stru_comp_a.dbl_imaginaire=0`
- Affectation globale : `stru_comp_a = stru_comp_b`
  - Attention aux pointeurs !
- Comparaison : obligatoire de comparer la structure champ par champ
  $\Leftrightarrow$ `stru_comp_a == stru_comp_b` est interdit !
- Possibilité d'utiliser l'opérateur `sizeof`
- Ne jamais utiliser de *cast*

Structures

Tableau de structures et composition de structure

```c
/* Dans le header, et dans le bon ordre */
struct nom_prenom {
   char str_prenom[30];
   char str_nom[30];
};
struct livre {
   struct nom_prenom stru_auteur;
   int int_annee;
   char str_titre[50];
   char str_editeur[50];
   .../...
};
```

```c
/* Utilisation des structures */
struct livre tstru_biblio[100];
printf("%s\n",tstru_biblio[5].stru_auteur.str_nom);
```

Types équivalents

Types équivalents

Définition

- Renommage d'un type existant
- Utile pour économiser des caractères, rendre le code lisible, portabilité

Types équivalents

Déclaration et utilisation

```
typedef type type_equivalent;
typedef type type_equiv1, type_equiv2, ... ;
```

Types équivalents

Exemple

```c
typedef double reel;
typedef int entier, booleen;

typedef struct livre slivre;
```

```c
// Utilisation
entier ent_a;
booleen bool_condition;
slivre biblio[100];
```

Types équivalents

Cas particulier

- Replacement d'un type tableau, par un autre
  - `typedef char chaine [80];`
  - `typedef char chaine[80];`
- Utilisation à la déclaration d'une structure

```
typedef struct {
   chaine nom;
   chaine prenom;
} auteur ;
```

Directives préprocesseurs

Directives préprocesseurs

Présentation

- `#include` : inclusion de sources
- `#define` : définition de constantes ou macros
- `#if #elif #else #endif` : compilation conditionnelle
- `#ifdef #ifndef` : existence de symbole
- `#error` et `#pragma` : portabilité

Directives préprocesseurs

#include

- Incorpore un fichier dans un autre
- Deux syntaxes :
  - `#include <stdio.h>` : recherche stdio.h dans les répertoires d'include
  - `#include "toto.h"` : recherche toto.h dans le répertoire courant ainsi que dans les répertoires spécifiés par `-I`

Directives préprocesseurs

#if et consort

- Compilation conditionnelle
- Peut servir pour avoir du code différent en fonction des variables d'environnement, de l'environnement, ...
- Possibilité de faire passer une variable via la ligne de compilation (`-DLOGIN=elisabeth`)

```
#if OS == LINUX
  #include <linux.h>
#elif OS == UNIX
  #include <unix.h>
#else
  #include <os.h>
#endif
```

Directives préprocesseurs

#ifndef, #ifdef

- Compilation conditionnelle sur existence de variables
- Souvent utilisé pour le debuggage

```
#ifdef __cplusplus
  #include <cstdio>
  #include <cstdlib>
#else
  #include <stdio.h>
  #include <stdlib.h>
#endif
```

Directives préprocesseurs

#error, #pragma

- `#error "msg"` : stoppe la compilation et affiche le message d'erreur
- `#pragma ...` : donne un ordre spécifique à un compilateur. Instruction non normalisée.