Haute école d'ingénierie et d'architecture Fribourg
Hochschule für Technik und Architektur Freiburg

Systèmes Embarqués 1 & 2

a.11 – C – Les structures complexes

Classes T-2/I-2 // 2017-2018

Daniel Gachet | HEIA-FR/TIC
a.11 | 23.10.2017

Hes·so
Haute Ecole Spécialisée de Suisse occidentale
Fachhochschule Westschweiz

Contenu

- Enumération
- Structure
- Tableau
- Union
- Champ de bits
- Définition de type

Enumération

L'énumération (`enum`) est un type de données permettant de définir une suite de valeurs constantes entières nommées (`constant`) spécifiées part le développeur. Une fois définies, les constantes nommées peuvent être très simplement utilisées comme des nombres entiers.

La définition d'une énumération prend la forme suivante:

```
enum type_name {ELE1, ELE2, ELE3};
```

Cet exemple définit 3 constantes entières nommées, appelées énumérateurs.
Une valeur implicite et croissante est assignée à chaque énumérateur.
Le premier élément reçoit la valeur 0, le deuxième 1 et ainsi de suite, soit pour l'exemple ci-dessus: `ELE1==0`, `ELE2==1` et `ELE3==2`.

L'énumérateur peut également recevoir une valeur explicite définie avec une expression constante, p. ex:

```
enum type_name_2 {MIN=-10, MAX=10};
enum type_name_3 {E1,              // E1==0
                  E2=10,           // E2==10
                  E3,              // E3==11
                  E4=10*2+1,       // E4==21
                  E5               // E5==22
                 };
```

Structure – déclaration

Une structure (ou enregistrement), `struct`, est un élément du langage de programmation très puissant pour structurer un programme et ses données. `struct` permet aux développeurs de regrouper plusieurs données/variables (ou champs) de types différents sous un même nom. `struct` simplifie grandement le traitement de l'information qu'un programme doit manipuler. La définition d'une structure prend la forme suivante:

```
struct point {
  int x;
  int y;
};
struct rectangle {
  struct point p1;
  struct point p2;
};
```

Les champs d'une structure (variables / données) sont également appelés membres ou attributs.

Une structure peut être instanciée aussi simplement qu'une variable d'un type de base du langage, p. ex:

```
struct point     p1;
struct rectangle r1;
```

Structure – utilisation / initialisation

Les membres d'une structure peuvent facilement être référencés dans une expression en utilisant la construction `struct_variable_name"."member`. Par exemple pour obtenir la valeur de l'axe x du point 1 d'un rectangle:

```
int x = r1.p1.x;
```

Une structure peut être initialisée lors de l'instanciation de la variable, p. ex:

```
struct point p1 = {
  20,
  -10
};
struct rectangle r1 = {
  {20, -10},   // point 1: x==20 / y==-10
  {40,  10}    // point 2: x==40 / y==10
};
```

ou depuis le standard C99:

```
struct point p1 = {
  .x=20,
  .y=-10,
};
```

```
struct rectangle r1 = {
.p1={.x=20, .y=-10},
.p2={.x=40, .y=10},
};
```

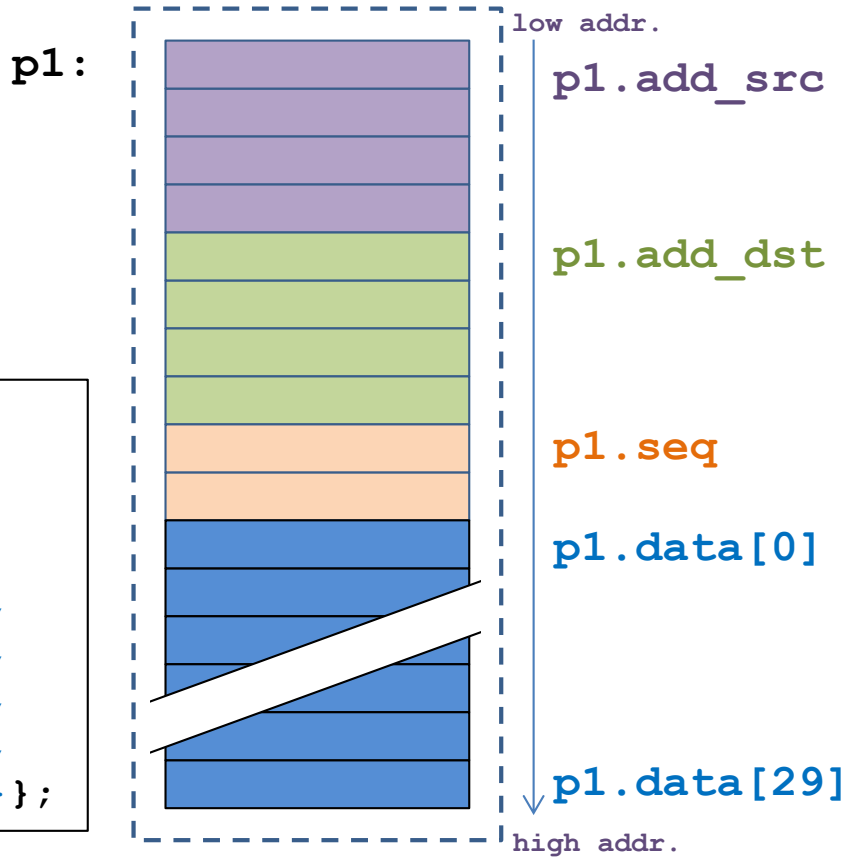
Structure : exemple de déclaration

Déclaration I:

```
struct paquet {
        long    add_src;
        long    add_dst;
        short   seq;
        char    data[30];
};
struct paquet p1;
```

Initialisation:

```
struct paquet p1 = {
        0x03acb294,
        0xabd97553,
        25,
        {0xaa,0xbb,0xcc,0xdd,0xee,0xff,
         0xaa,0xbb,0xcc,0xdd,0xee,0xff,
         0xaa,0xbb,0xcc,0xdd,0xee,0xff,
         0xaa,0xbb,0xcc,0xdd,0xee,0xff,
         0xaa,0xbb,0xcc,0xdd,0xee,0xff}};
```



Structure : autres déclarations équivalentes

Déclaration II:

```
struct {
        long    add_src;
        long    add_dst;
        short   seq;
        char    data[30];
} p1;
```

Déclaration III:

```
struct paquet {
        long    add_src;
        long    add_dst;
        short   seq;
        char    data[30];
} p1;
```

Déclaration IV:

```
typedef struct {
        long    add_src;
        long    add_dst;
        short   seq;
        char    data[30];
} paquet_t;
paquet_t p1;
```

Tableau - déclaration

Les tableaux (*array*) sont des constructions très intéressantes permettant aux développeurs de bien structurer leurs programmes logiciels. Par opposition aux structures (`struct`), les tableaux ne permettent de regrouper que des champs d'un même type de données. C supporte le concept de tableaux à plusieurs dimensions. La définition d'un tableau prend la forme suivante:

```
int array1[5];
struct point array2[2][3];
```

`array1` est un tableau de `5` éléments de type `int`.
`array2` est un tableau à deux dimensions composé de `2` éléments (première dimension, p.ex. les lignes) contenant chacun `3` éléments (deuxième dimension, p.ex. les colonnes) de type `struct point`.

Tableau – utilisation / initialisation

Les membres d'un tableau peuvent facilement être référencés dans une expression en utilisant la construction suivante `array_name [index1][index2]`, p. ex.:

```
int x = array1[3];
struct point p = array2[0][2];
```

Il est important de noter que `0` indexe le premier élément d'un tableau et que pour un tableau de n-éléments, `n-1` adresse le dernier.

Dans l'exemple ci-dessus, `x` reçoit le 4ème élément du tableau `array1` et `p` obtient le 3ème élément de la 1ère ligne du tableau `array2`.

Un tableau peut être initialisé lors de l'instanciation de la variable, p.ex.:

```
int array1[5] = {0, 1, 2, 3, 4};
struct point array2[2][3] = {
  { {10, 20}, {11, 21}, {12, 22} },
  { {13, 23}, {14, 24}, {15, 25} }
};
```

Union

Une `union` est une construction permettant de déclarer à une variable devant contenir à des instants différents des données d'un type différent et d'une taille différente. La définition d'une union prend la forme suivante:

```
union name {
  int   ival;
  float fval;
  char* sval;
  struct point *pval;
};
```

Les champs de bits (bit-fields) fournissent un autre moyen de déclarer des variables ayant à traiter des ensembles de bits (bitset), p.ex.:

```
enum enum1 {E1, E2, E3, E4};

struct bit_field {
  enum enum1 e1 : 2;
  enum enum1 e2 : 2;
  bool       b1 : 1;
  bool       b2 : 1;
  unsigned   u1 : 2;
};
```

Attention:

L'implémentation des champs de bits et dépendante de la machine cible.

Définition de type

Grâce au mot clef `typedef`, C permet de déclarer un nouveau nom pour un type donné. La définition du nouveau type prend la forme suivante:

```
typedef <c_type> <type_name_t>;
```

p.ex.:

```
typedef unsigned char level_t;
```

Note:

`typedef` est seulement un synonyme pour un type donné et non pas un type distinct.

Soyez attentif à l'alignement des variables:

1. Certains µP ne supportent pas des accès mémoires non alignés
2. Ne pas utiliser le pragma `__attribute__((__packed__))`, lequel est dépendant du compilateur
3. Il est de bon conseil d'aligner les variables à la limite de leur taille:
   1. Variable à 8 bits sur 1 byte → **`0xffff'ffff`**
   2. Variable à 16 bits sur 2 bytes → **`0xffff'fffe`**
   3. Variable à 32 bits sur 4 bytes → **`0xffff'fffc`**
   4. Variable à 64 bits sur 8 bytes → **`0xffff'fff8`**