HTA-FR – Kurs Telekommunikation

Verfasser:
D. Gachet / HTA-FR - Telekommunikation

Embedded systems 1 und 2

Programmiersprache C – Komplexe Strukturen

Klasse T-2 // 2018-2019

Inhalt

- Aufzählungen
- Struktur
- Tabelle
- Union
- Bitfeld
- Typendefinition

Aufzählungen

Die Aufzählung (`enum`) ist ein Datentyp, der die Definition einer Folge von konstanten, benannten Ganzzahlwerten (`constant`) erlaubt, die durch den Entwickler spezifiziert werden. Sobald sie definiert sind, können die benannten Konstanten sehr einfach wie ganze Zahlen (Integer) benutzt werden.

Die Definition einer Aufzählung hat die folgende Form:

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

Dieses Beispiel definiert 3 benannte Ganzzahlkonstanten, die Aufzählungsvariablen genannt werden.
Jeder Aufzählungsvariablen wird ein impliziter, aufsteigender Wert zugewiesen. Das erste Element erhält den Wert 0, das zweite den Wert 1 und so weiter, d. h. im oben stehenden Beispiel: `ELE1==0`, `ELE2==1` und `ELE3==2`.

Die Aufzählungsvariable kann auch einen expliziten Wert erhalten, der mit einem konstanten Ausdruck definiert wird, z. B.:

```
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
                 };
```

Struktur – Deklaration

Eine Struktur (oder ein Datensatz), `struct`, ist ein sehr mächtiges Element der Programmiersprache zur Strukturierung eines Programms und seiner Daten. `struct` erlaubt den Entwicklern, mehrere Daten/Variablen (oder Felder) mit unterschiedlichem Typ unter einem einzigen Namen zusammenzufassen. `struct` vereinfacht die Verarbeitung der Information erheblich, die ein Programm behandeln muss. Die Definition einer Struktur hat die folgende Form:

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

Die Felder einer Struktur (Variablen/Daten) werden auch als Komponenten oder Attribute bezeichnet.

Eine Struktur kann ebenso einfach wie eine Variable eines Basistyps der Sprache instantiiert werden, z. B.:

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

Struktur – Verwendung / Initialisierung

Die Komponenten einer Struktur lassen sich in einem Ausdruck leicht mit der folgenden Konstruktion referenzieren `struct_variable_name"."member`.
Um zum Beispiel den Wert der x-Achse des 1. Punkts eines Rechtecks zu erhalten:

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

Eine Struktur kann bei der Instantiierung der Variablen initialisiert werden, z. B.:

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

oder nach Standard C99:

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

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

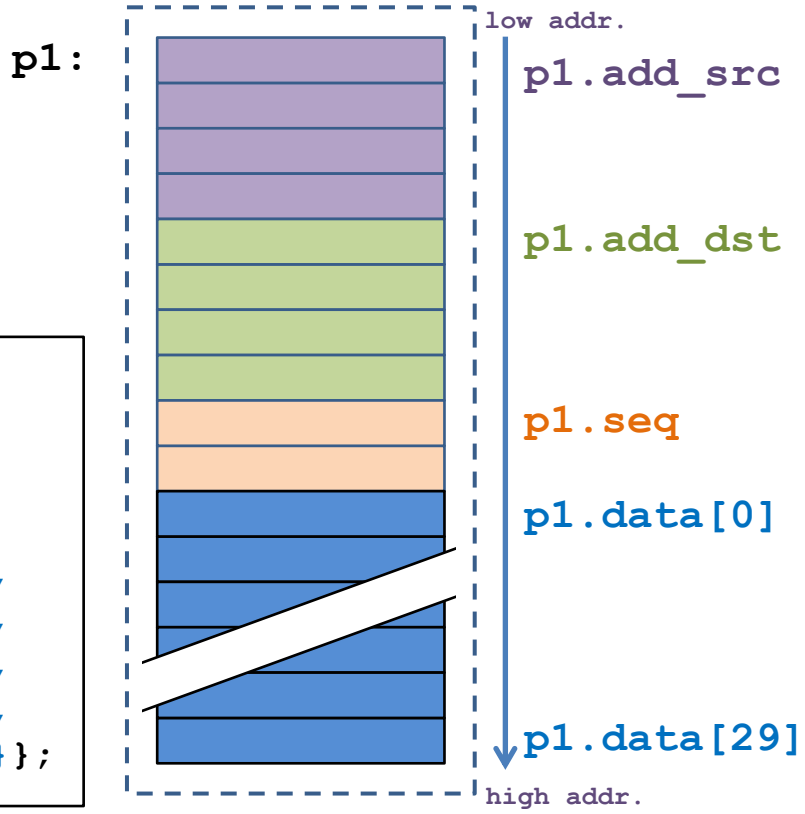
Struktur: Deklarationsbeispiel

Deklaration I:

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

Initialisierung:

```
struct packet 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}};
```



Struktur: weitere äquivalente Deklarationen

Deklaration II:

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

Deklaration III:

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

Deklaration IV:

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

packet_t p1;
```

Array - Deklaration

Arrays (*array*) sind sehr interessante Konstruktionen, die den Entwicklern erlauben, ihre Softwareprogramme gut zu strukturieren. Im Gegensatz zu den Strukturen (`struct`) erlauben Arrays nur die Felder eines gleichen Datentyps zusammenzufassen. C unterstützt das Konzept der mehrdimensionalen Arrays. Die Definition eines Arrays hat die folgende Form:

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

`array1` ist ein Array mit `5` Elementen vom Typ `int`.
`array2` ist ein zweidimensionales Array, das aus `2` Elementen (erste Dimension z. B. die Zeilen) besteht, von denen jedes `3` Elemente (zweite Dimension, z. B. die Spalten) vom Typ `struct point` enthält.

Array – Verwendung / Initialisierung

Die Elemente eines Arrays können in einem Ausdruck leicht mit der folgenden Konstruktion referenziert werden `array_name [index1][index2]`, z. B.:

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

Es ist wichtig, sich zu merken, dass `0` das erste Element eines Arrays indexiert und dass in einem Array mit n Elementen `n-1` das letzte Element adressiert.

Im oben stehenden Beispiel empfängt `x` das 4. Element des Arrays `array1` und `p` erhält das 3. Element der 1. Zeile des Arrays `array2`.

Ein Array kann bei der Instantiierung der Variablen initialisiert werden, z. B.:

```
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} }
};
```

Eine `union` ist eine Konstruktion, die es erlaubt, eine Variable zu deklarieren, die verschiedene Instanzen von Daten eines unterschiedlichen Typs und unterschiedlicher Grösse enthalten soll. Die Definition einer Union hat die folgende Form:

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

Bitfeld

Die Bitfelder (bit-fields) bietet ein anderes Mittel zur Deklaration von Variablen, die Bitfolgen (bitset) verarbeiten müssen, z. B.:

```
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;
};
```

Achtung:

Die Implementierung der Bitfelder ist von der Zielmaschine abhängig.

Typendefinition

Dank des Schlüsselworts `typedef` erlaubt C für einen gegebenen Typ, einen neuen Namen zu deklarieren. Die Definition des neuen Typs hat die folgende Form:

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

z. B.:

```
typedef unsigned char level_t;
```

Anmerkung:

`typedef` ist nur ein Synonym für einen gegebenen Typ und nicht ein bestimmter Typ.

Achten Sie auf die Ausrichtung der Variablen:

1. Gewisse µP unterstützen keinen nicht ausgerichteten Speicherzugriff
2. Benutzen Sie das Pragma `__attribute__((__packed__))` nicht, das vom Compiler abhängt
3. Wir sind gut beraten, die Variablen auf ihre Länge auszurichten:
   1. Variable mit 8 Bit auf 1 Byte → **`0xffff'ffff`**
   2. Variable mit 16 Bit auf 2 Byte → **`0xffff'fffe`**
   3. Variable mit 32 Bit auf 4 Byte → **`0xffff'fffc`**
   4. Variable mit 64 Bit auf 8 Byte → **`0xffff'fff8`**