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

HTA-FR – Kurs Telekommunikation

Verfasser:
D. Gachet / HTA-FR - Telekommunikation

Embedded systems 1 und 2

Programmiersprache C – Die Daten

Klasse T-2 // 2018-2019

Inhalt

- Deklaration der Variablen
- Ganze Zahlen
- Boolsche Zahlen
- Reelle Zahlen
- Zeichenketten
- Konstanten
- Speicherungsklassen

Deklaration der Variablen

In C hat die Deklaration der Variablen folgende Form:

```
TYPE var_name [= init_value] {, var_name [= init_value]};
```

Zum Beispiel:

```
int   var1, var2;          → identisch mit  int var1; int var2;
float var4;
int   var5, var6 = 10;     → identisch mit  int var5; int var6 = 10;
float var7 = 1.0;
char *var8 = "hello world!";
int  *var9, var10; → identisch mit  int *var9; int var10;
```

Anmerkung:

- Vorzugsweise sollte pro Zeile nur eine Variable deklariert werden.

Achtung:

- Die Deklaration von Variablen muss zwingend am Anfang eines Blocks vor dem ersten Befehl / der ersten Operation erfolgen (Einschränkung ab **C99** aufgehoben)

Ganze Zahlen (Integer)

C kann sowohl mit Zahlen mit als auch ohne Vorzeichen und mit Zahlen unterschiedlicher Länge gut umgehen.

Grenzen der Zahlen mit Vorzeichen: -2^{n-1} .. $2^{n-1} - 1$ (vgl. Datei <limit.h>)

Grenzen der Zahlen ohne Vorzeichen: 0 .. $2^{n} - 1$

Typen mit Vorzeichen:	ohne Vorzeichen:	Bits:	stdint.h:
`signed char`	`unsigned char`	`8`	`[u]int8_t`
`signed short int`	`unsigned short int`	`oft 16`	`[u]int16_t`
`signed int`	`unsigned int`	`16 or 32`	
`signed long int`	`unsigned long int`	`32`	`[u]int32_t`
`signed long long int`	`unsigned long long int`	`64 (GNU)`	`[u]int64_t`

Anmerkung:

Für die Typen **`short`**, **`long`** und **`long long`** kann das Präfix **`int`** weggelassen werden. Für die Datentypen mit Vorzeichen kann der Kennzeichner **`signed`** ebenfalls weggelassen werden.

Achtung:

Die Sprache spezifiziert nicht, ob die Daten vom Typ **`char`** Typen mit oder ohne Vorzeichen sind.

Arithmetische Operatoren und Komparatoren

Operatoren:

+	Addition
-	Subtraktion
*	Multiplikation
/	Division
%	Rest/Modulo

Komparatoren:

==	gleich
!=	ungleich
<	kleiner als
>	grösser als
<=	kleiner oder gleich
>=	grösser oder gleich

Achtung:

- C führt keine Tests zur Bereichsüberschreitung durch.
- Eine Division durch null (0) führt zu einem Absturz der Anwendung.

Bitweise Operatoren

Operatoren:

|	bitweise ODER
&	bitweise UND
^	(bitweise) Exklusiv ODER
~	1er-Komplement
<<	Verschiebung nach links
>>	Verschiebung nach rechts

Achtung:

Das Verhalten des bitweise Operators "**>>**" angewendet auf Zahlen mit Vorzeichen ist durch die Sprache nicht definiert und hängt vom Compiler ab.

Boolsche Zahlen und die Operatoren

Die Sprache C definiert keinen spezifischen Typ für Daten vom Typ logisch. Dagegen schlägt sie 3 Operatoren vor, die die Ausführung logischer Operationen auf ganze Zahlen erlauben.

Operatoren:

`&&`	logisch UND
`||`	logisch ODER
`!`	logisch NICHT

Der Typ `bool` wurde in C99 eingeführt. Der Typ ist in der Datei `<stdbool.h>` definiert, die auch die Konstanten `true` (1) und `false` (0) liefert.

Anmerkung:

Alle von 0 (!= 0) verschiedenen Werte stellen wahr "true" dar, während der Wert 0 (== 0) falsch "false" darstellt.

Reelle Zahlen

C ist auch in der Lage, reelle Zahlen (Gleitkommazahlen) zu verarbeiten.

`float`	**einfache Genauigkeit**
`double`	**doppelte Genauigkeit**
`long double`	**erweiterte Genauigkeit**

Operatoren und Komparatoren:

Mit der Ausnahme von Modulo (`%`) werden alle Operatoren und Komparatoren der ganzen Zahlen auch für reelle Zahlen unterstützt.

Anmerkung:

In den meisten embedded systems erfolgt die Verarbeitung der reellen Zahlen durch Bibliotheken zur Emulation mathematischer Coprozessoren.
Dadurch werden die Berechnungszeiten oft sehr lang.

Achtung:

Die Sprache spezifiziert die Genauigkeit dieser Typen nicht. Diese hängt von den eingesetzten Mikroprozessoren und Compilern ab. Der Wertebereich ist in der Datei <float.h> definiert.

Zeichen und Zeichenketten (strings)

C unterstützt wie alle andern Programmiersprachen einen Datentyp "Zeichen" und Zeichenketten (strings), um den Menschen zu erlauben, mit den Maschinen zu interagieren.

`char`	**ein einzelnes Zeichen**
`char*, char[]`	**eine offene Zeichenkette**
`char[n]`	**eine Zeichenkette mit einer festen Anzahl Zeichen**

Operatoren und Komparatoren:

Auf die Daten vom Typ char dürfen alle Operatoren und Komparatoren der ganzen Zahlen angewendet werden. Dagegen bietet C keinerlei Unterstützung für die Verarbeitung von Zeichenketten. Diese ist dem Programmierer überlassen. Die Bibliotheken "string.h" und "ctype.h" stellen indessen Funktionen zur Verfügung, die die Arbeit massiv vereinfachen.

Anmerkung:

Für die Behandlung von strings ist die Beherrschung von Tabellen (Arrays) und Zeigern unabdingbar.
Es ist wichtig, darauf hinzuweisen, dass die strings durch null ('\0') abgeschlossen sind/werden müssen.

Achtung:

Die Sprache spezifiziert nicht, ob die Daten vom Typ `char` Typen mit oder ohne Vorzeichen sind.

Zeichen und Zeichenketten (Beispiele)

```
char  a_character   = 'a';
```

→ ein einzelnes Zeichen

```
char* a_1st_string     = "Hello Guys";
```

→ verweist auf eine "konstante" Kette mit 10 Zeichen + '\0'

```
char  a_2nd_string[]   = "Bonjour tout le monde";
```

→ Tabelle mit 21 Zeichen + '\0'

```
char  a_3rd_string[32] = "Good Morning";
```

→ 12 Zeichen + '\0' in einer Tabelle von 32 Elementen enthalten

C unterstützt das Konzept der Konstanten für alle Basistypen.

Ganze Zahlen (`int`):

- Dezimalzahlen: `1234`
- Oktalzahlen: `0ooo` `(017, 0377)` → $(15_{10}, 255_{10})$
- Hexadezimalzahlen: `0xhh` or `0XHH` `(0xf, 0XFF)` → $(15_{10}, 255_{10})$
- Suffixe:

`l (L)`	für eine `long int`	`(-77l)`
`ll (LL)`	für eine `long long int`	`(77ll)`
`u (U)`	für eine `unsinged int`	`(77u)`
`ul (UL)`	für eine `unsigned long int`	`(77ul)`
`ull (ULL)`	für eine `unsigned long long int`	`(77ull)`

Reelle Zahlen (`double`):

- Dezimalnotation: `123.45`
- Wissenschaftliche Notation: `1e-2`
- Kombinierte Notation: `123.45e10`
- Suffixe:

`f (F)`	für eine `float`	`(123.45f)`
`l (L)`	für eine `long double`	`(198.45e-48l)`

Konstanten (II)

Zeichen (`char`):

- Ein Zeichen: `'x'` → Zeichen x, `'0'` → Zeichen 0 (null), …
- Oktal: `'\ooo'` `('\170', '\60')` → `('x', '0')`
- Hexadezimal: `'\xhh'` `('\x78', '\x30')` → `('x', '0')`
- Escape:

`\n`	newline	`\\`	backslash
`\'`	single quote	`\"`	double quote
`\b`	backspace	`\v`	vertical tab
`\f`	formfeed	`\t`	horizontal tab
`\a`	alert (bell)	`\?`	question mark
`\r`	carriage return		

Strings (`char*`):

- Ein String: `"I am a non empty string"`
- ein Leerstring: `""`
- eine Stringverknüpfung `"A "  "concatenated " "string"`

Konstanten (III)

In C existieren zwei Arten für die Deklaration von Konstanten:

Symbolische Konstanten:

```
#define name  replacement_text
```

```
z. B.   #define GRETTINGS    "Hello World!"
        #define LOWER        100
        #define UPPER        345
        #define ELEMENTS     (UPPER - LOWER + 1)
```

Konstante Variablen:

```
[static] const TYPE name = value;
```

```
z. B.   const char   GRETTINGS[] = "Hello World!";
        const uint32 LOWER       = 100;
        const uint32 UPPER       = 345;
```

aber

```
        const uint32 ELEMENTS    = UPPER - LOWER + 1;
```

wird nicht unterstützt

Speicherungsklassen

Die Sprache C definiert verschiedene Speicherungsklassen von Variablen. Diese nehmen die folgende Form an:

```
<classe> <type> declarator;
```

Variable	Klasse	Gültigkeitsbereich	Lebensdauer	Erklärung
local	auto	Block	Block	Der Wahl des Compilers überlassen (Vorgabe)
	register			Variable wird in einem Register abgelegt
	volatile			Die Variable wird auf dem Stapel (stack) abgelegt
	static		Programm	Die Variable wird mit den globalen Variablen abgelegt
global	extern	Quelldatei mit den Definitionen und nach jeder Deklaration alle Module des Programms	Programm	Referenz auf eine globale Variable
	static	Quelldatei mit Definition		Die Variable wird mit den globalen Variablen in einem zum Programm gehörenden und zu diesem Zweck reservierten globalen Speicherbereich abgelegt