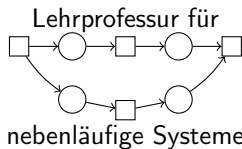


Vorkurs Informatik

ASCII-Zeichen / Fallunterscheidungen

Robert Lorenz

Tag 3



Wiederholung

- Alle in einem C-Programm verwendeten Daten haben einen festgelegten sog. **Datentyp**.
- Jeder Datentyp definiert eine Wertemenge, aus der die zugehörigen Daten kommen müssen.

Definition char

- Der Datentyp char repräsentiert die Menge der **ASCII-Zeichen** (siehe ASCII-Tabelle aus Kapitel 1).

Zeichen-Konstanten sind vom Typ `char`

Wiederholung

- Alle in einem Programm verwendeten Konstanten haben einen festgelegten Datentyp; dieser ergibt sich aus deren Schreibweise
- Der Compiler interpretiert eine im Quellcode vorkommende Folge von Zeichen als Konstante, wenn die Folge von Zeichen die zu einem Datentyp passende Schreibweise hat

Allgemeine Schreibweise von Zeichen-Konstanten

`'<Zeichen>'`

wobei `<Zeichen>` ein Zeichen der ASCII-Tabelle ist.

Allgemeine Schreibweise von Zeichen-Konstanten

'<Zeichen>'

wobei <Zeichen> ein Zeichen der ASCII-Tabelle ist:

- druckbare Zeichen: werden genauso ausgegeben wie im Quellcode angegeben (z.B. A, 4, =)
- diverse nicht druckbare Zeichen und Sonderzeichen: werden mit sog. **Escapesequenzen** angegeben

Escapesequenzen

Werden als mit \ beginnende Folge von Zeichen angegeben:

- \n Ausgabe neue Zeile
- weitere: \\, \t, \a, ... selbst recherchieren

Wert von Zeichen-Konstanten

- Zeichen in C-Programmen werden als **ganze Zahlen** zwischen 0 und 127 gespeichert, char ist also ein ganzzahliger Typ.
- Die Zuordnung von Zeichen zu Zahlen gibt die ASCII-Tabelle an; hier werden die ASCII-Zeichen durchnummeriert
- Die zu einem Zeichen gehörende Zahl (Nummer) aus der ASCII-Tabelle heißt deren **ASCII-Code**

Deshalb kann man mit Zeichen rechnen (also die arithmetischen Rechenoperationen anwenden), z.B.:

- 'z' - 1 ist die Darstellung für das Zeichen **vor** z in der ASCII-Tabelle

In der ASCII-Tabelle haben einige Gruppen zusammengehöriger Zeichen **aufeinanderfolgende ASCII-Codes**.

Ziffern

Die Ziffern '0' bis '9' haben die ASCII-Codes 48 - 57.

Lateinische Kleinbuchstaben

Die lateinischen Kleinbuchstaben 'a' bis 'z' haben die ASCII-Codes 97 - 122.

Lateinische Großbuchstaben

Die lateinischen Großbuchstaben 'A' bis 'Z' haben die ASCII-Codes 65 - 90.

Wiederholung

Funktionsaufruf von printf:

```
printf(<Zeichenkette>,<Ausdruck>,...);
```

Umwandlungsangaben für Zeichen

- %c ist eine Umwandlungsangabe für Zeichen
- Für jede solche Umwandlungsangabe in <Zeichenkette> muss bei Aufruf ein char-Ausdruck als zusätzlicher Eingabeparameter übergeben werden.

Man kann ein Zeichen auch für die Umwandlungsangabe %i einsetzen: in diesem Fall gibt man den ASCII-Code des Zeichens aus.

Beispiel

Der Funktionsaufruf

```
printf("%c ist eine Ziffer", '3')
```

bewirkt die Ausgabe:

3 ist eine Ziffer

Typumwandlungen in Wertzuweisungen

Haben Variable und Ausdruck in einer Wertzuweisung

`<Variable> = <Ausdruck>;`

unterschiedlichen Datentyp, so wird der Wert des Ausdrucks (automatisch) in den Datentyp der Variable umgewandelt.

Beispiele

- `char c = 65;`
c erhält als Wert das Zeichen mit dem ASCII-Code 65 - also den Wert 'A' (ACHTUNG: Unerwartete Ergebnisse, wenn der Wert des Ausdrucks nicht zwischen 0 und 127 liegt!)
- `int n = '0';`
n erhält als Wert den ASCII-Code des Zeichens 0 (also den Wert 48)

Typumwandlungen in Ausdrücken

`char`-Werte werden in Ausdrücken immer in den Datentyp `int` umgewandelt.

Die Bibliothek `ctype.h` enthält einige nützliche vordefinierte und -installierte Funktionen für Zeichen.

is-Funktionen

Mit dem Aufruf einer is-Funktion kann man testen, ob ein Zeichen zu einer bestimmten Gruppe von Zeichen gehört, z.B.

- Der Aufruf `isdigit(c)` gibt 0 zurück, wenn das Zeichen `c` keine Ziffer ist, sonst einen Wert ungleich 0
- Der Aufruf `islower(c)` gibt 0 zurück, wenn das Zeichen `c` kein lateinischer Kleinbuchstabe ist, sonst einen Wert ungleich 0

to-Funktionen

Mit dem Aufruf einer to-Funktion kann man Zeichen umwandeln

- Der Aufruf `toupper(c)` gibt `c` zurück, wenn `c` kein lateinischer Kleinbuchstabe ist, sonst den zu `c` gehörenden Großbuchstaben
- `tolower(c)`: gibt `c` zurück, wenn `c` kein lateinischer Großbuchstabe ist, sonst den zu `c` gehörenden Kleinbuchstaben

Beispiel

Der Funktionsaufruf

```
printf("Der Grossbuchstabe zu %c ist %c", 'a',  
toupper('a'))
```

bewirkt die Ausgabe:

Der Grossbuchstabe zu a ist A

Eine **Bedingung** ist ein **logischer Ausdruck** und repräsentiert eine Aussage, die wahr sein kann oder falsch.

Einfache Bedingungen

Sind A und B vom Typ char oder int, so sind folgende Ausdrücke **Bedingungen**:

- $A > B$ ('größer als' - Operator)
- $A \geq B$ ('größer als oder gleich' - Operator)
- $A < B$ ('kleiner als' - Operator)
- $A \leq B$ ('kleiner als oder gleich' - Operator)
- $A == B$ ('ist gleich' - Operator)
- $A != B$ ('ist ungleich' - Operator)

$<$, $\leq$, $>$, $\geq$, $==$, $!=$ heißen **Vergleichsoperatoren**

Auswertung

- **Wahre** Bedingungen haben in C den Wert 1
- **Unwahre** Bedingungen haben in C den Wert 0

Beispiele

- $10 > 10$ hat den Wert 0
- $10 \geq 10$ hat den Wert 1
- $10 == 10$ hat den Wert 1
- $10 != 10$ hat den Wert 0

Komplexe Bedingungen

Sind A und B Bedingungen, so sind auch folgende Ausdrücke Bedingungen:

- $A \ \&\& \ B$ ('und' - Operator)

Hat genau dann den Wert 1, wenn A und B beide den Wert 1 haben

- $A \ || \ B$ ('oder' - Operator)

Hat genau dann den Wert 0, wenn A und B beide den Wert 0 haben

- $!A$ ('nicht' - Operator)

Hat genau dann den Wert 1, wenn A den Wert 0 hat

Die Operatoren $\&\&$, $||$, $!$ heißen **logische Operatoren**.

Zahlwertige Bedingungen

- In C kann man jeden zahlwertigen Ausdruck A als Bedingung benutzen, z.B. indem man einen der logischen Operatoren anwendet
- Dieser hat den (Wahrheits-)Wert $A \neq 0$ (wird also genau dann als wahr interpretiert, falls er einen Wert ungleich 0 hat)
- Beispiel: $5 \ || \ 0$ hat den Wert 1

Verwendung von bool

- Bindet man die Bibliothek `stdbool.h` in das Programm ein, so steht der Datentyp `bool` zur Verfügung.
- Der Datentyp `bool` repräsentiert die Wahrheitswerte `true` (wahr) und `false` (falsch).
- Wahrheitswerte in C-Programmen werden als **ganze Zahlen** ausgewertet (kompatibel mit der Auswertung von Bedingungen):
 - `true` hat den Wert 1.
 - `false` hat den Wert 0.
- Wahrheitswerte lassen sich mit `printf` (nur) über die Umwandlungsangabe `%i` als ganze Zahl ausgeben

Auswertungsreihenfolge

Vergleichs- und logische Operatoren werden in folgender Reihenfolge ausgewertet:

- 1 `()`
- 2 `!`
- 3 `<, <=, >, >=` (gleichrangig)
- 4 `==, !=` (gleichrangig)
- 5 `&&`
- 6 `||`

Gleichrangige Operatoren werden von links nach rechts ausgewertet

Beispiele

- Die Bedingung `('0' <= c) && (c <= '9')` überprüft, ob der Wert der Variable `c` zwischen den ASCII-Codes der Ziffern `'0'` und `'9'` liegt
- `1 > 0 || 1 < 0` ist eine wahre Bedingung

Achtung

Der Ausdruck `'0' <= c <= '9'` hat **nicht** den gewünschten Effekt!
(Wieso?)

Definition

```
1 <A> /* Vorherige Anweisung */  
2 if (<Bedingung>) { /* Beginn des if-Blocks */  
3   <Optionale_Anweisungen>  
4 } /* Ende des if-Blocks */  
5 <B> /* Nachfolgende Anweisung */
```

- Nach der Ausführung von <A> wird die <Bedingung> überprüft (ausgewertet)
- Ist <Bedingung> **wahr**, so werden <Optionale_Anweisungen> ausgeführt und danach
- Ist <Bedingung> **nicht wahr**, so wird sofort mit fortgesetzt (und <Optionale_Anweisungen> ausgelassen / übersprungen)

Definition

```
1 <A> /* Vorherige Anweisung */  
2 if (<Bedingung>) { /* Beginn des if-Blocks */  
3   <Optionale_Anweisungen>  
4 } /* Ende des if-Blocks */  
5 <B> /* Nachfolgende Anweisung */
```

- Für <Bedingung> kann auch ein zahlwertiger Ausdruck eingesetzt werden. Dieser wird als wahr interpretiert, wenn er einen Wert ungleich Null hat.
- Beispiel: `if(isdigit('3'))` wird als wahr interpretiert.
- Beispiel: `if(isdigit('a'))` wird als falsch interpretiert.

Definition für genau eine Alternative

```
1 <A> /* Vorherige Anweisung */  
2 if (<Bedingung>) {  
3   <if_Anweisungen>  
4 } else { /* Beginn des else-Blocks */  
5   <else_Anweisungen>  
6 } /* Ende des else-Blocks */  
7 <B> /* Nachfolgende Anweisung */
```

- Nach der Ausführung von <A> wird die <Bedingung> überprüft (ausgewertet)
- Ist <Bedingung> **wahr**, so werden <if_Anweisungen> ausgeführt und danach
- Ist <Bedingung> **nicht wahr**, so werden <else_Anweisungen> ausgeführt und danach

Definition für mehrere Alternativen

```
1 <A>
2 if (<B_1>) { /* Gecheckt nach <A> */
3   <A_1> /* Falls <B_1> wahr */
4 } else if (<B_2>) { /* Gecheckt falls <B_1> falsch */
5   <A_2> /* Falls (!<B_1> && <B_2>) wahr */
6 } else {
7   <A_3> /* Falls (!<B_1> && !<B_2>) wahr */
8 }
9 <C>
```

- Auswahl zwischen drei alternativen Anweisungsblöcken
- Zur Auswahl zwischen mehr als drei alternativen Anweisungsblöcken lassen sich in der Mitte mehrere `else if` - Blöcke hintereinander benutzen

Anweisungsblock

Ein Anweisungsblock in C ist von der Form

```
{  
    <Anweisungen>  
}
```

Beispiele:

- Funktionsrümpfe
- if-, else- und else if-Blöcke (NEU!)

Gültigkeitsbereich

Jeder Anweisungsblock erzeugt einen sog. **Gültigkeitsbereich für Variablen**

- In einem Gültigkeitsbereich deklarierte Variablen heißen für diesen Bereich **lokal**
- Lokale Variablen existieren **nur während der Ausführung** des Anweisungsblocks, also:
 - Reservieren von Speicherplatz durch Deklaration
 - Freigeben von Speicherplatz am Ende des BlocksSie können also nur in ihrem Block verwendet werden

Fehlerhafte Benutzung einer lokalen Variable

```
1 {  
2   int x;  
3 }  
4 x = 0;
```

Konventionen für Anweisungsblöcke

- `if`-, `else`- und `else if`-Blöcke beginnen in der ersten Zeile (nicht in der Zeile darunter wie Funktionsrumpfe); Abschließendes `}`-Zeichen in separate Zeile setzen
- Anweisungen in einem Anweisungsblock um 1 Tab (= 8 Leerzeichen) einrücken (Anweisungsblöcke können ineinander verschachtelt sein!)
- Leerzeichen einfügen vor und nach Bedingungen (eines `if`- und `else if`-Blocks)
- `else`- und `else if`-Block in der letzten Zeile des vorherigen Blocks beginnen; Leerzeichen einfügen vor und hinter `else` und `else if`

In diesem Kapitel wurden u.a. folgende reservierte Schlüsselwörter besprochen:

- `if, else, else if`: Fallunterscheidungen
- `char`: Datentyp für Zeichen
- `bool`: Datentyp für Wahrheitswerte
- `true, false`: Wahrheitswerte
- `isdigit, islower, tolower, toupper`: Bibliotheksfunktionen

ACHTUNG: In C-Programmen wird Klein- und Großschreibung unterschieden!

Beispielprogramm

```
1  int main(void)
2  {
3      int x = rand();
4      char c = x % 128, d;
5      if (c >= '0' && c <= '9') {
6          printf("Das Zeichen %c ist eine Ziffer\n", c);
7      } else if (c >= 'a' && c <= 'z') {
8          d = toupper(c);
9          printf("Als Zeichen: %c\nAls ganze Zahl: %i\n", d, d);
10     } else {
11         printf("Fehler");
12     }
13     return 0;
14 }
```

- Zeile 4: Zeichen-Variablen c, d deklarieren; c Zufallswert zwischen 0 und 127
- Zeile 5: Ist c Ziffer? (ASCII-Code liegt zwischen denen der Ziffern '0' und '9')
- Zeile 6: Ausgabe von c mit Umwandlungsangabe %c
- Zeilen 7 + 10: Alternativer Fälle
- Zeile 8: c in Großbuchstaben umwandeln