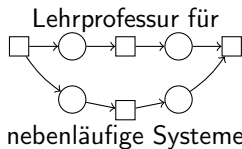


Vorkurs Informatik

Erstes Programm / Variablen / Ganze Zahlen

Robert Lorenz

Tag 2



C-Programm (Wiederholung aus Kapitel 1)

- Ein C-Programm ist eine ungeordnete Sammlung von Funktionsdefinitionen (d.h. die Reihenfolge der Definitionen ist nicht wichtig).
- Es muss immer genau eine Hauptfunktion `main` geben; Wird das Programm gestartet, so werden die Anweisungen dieser Hauptfunktion ausgeführt.
- Meistens werden auch noch Standard-Bibliotheken mit vordefinierten Funktionen eingebunden.

Wir starten mit einfachen Programmen, die nur eine Hauptfunktion und keine weiteren Funktionen haben.

Wir üben dabei einfache Berechnungen und Ausgaben für ganze Zahlen.

Beispielprogramm

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main(void)
5 {
6     int x, y;
7     x = rand();
8     y = 2 * x;
9     printf("Das Ergebnis ist: %i", y);
10    return 0;
11 }
```

- Zeile 1: Bibliothek `stdio.h` einbinden (für Benutzung der Funktion `printf` in Zeile 9)
- Zeile 2: Bibliothek `stdlib.h` einbinden (für Benutzung der Funktion `rand` in Zeile 7)
- Zeile 4: Kopf der Funktion `main`
- Zeilen 5 - 11: Rumpf der Funktion `main` (enthält auszuführende Anweisungen)

Definition

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

Wir besprechen in Vorkurs Datentypen für

- ganze Zahlen (Thema 2): `int`
- Buchstaben / Zeichen (Thema 3): `char`
- Dezimalzahlen (Thema 4): `double`

Definition

Der Datentyp `int` repräsentiert die Menge der ganzen Zahlen zwischen -32767 und 32767 (jeweils einschließlich).

Das Überschreiten der Wertegrenzen in Rechnungen mit dem Datentyp `int` führt zu unerwarteten Ergebnissen (vermeiden!)

Anmerkung: Diese Wertegrenzen sind durch den Standard vorgeschriebene Mindestgrößen. Die tatsächlichen Wertegrenzen sind Compiler-abhängig.

Definition

Eine **Konstante** ist ein Wert in einem Programm, der nach seiner erstmaligen Festlegung nicht mehr verändert werden kann.

- 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

Eine Schreibweise für int-Konstanten

<Vorzeichen><Ziffernfolge>

wobei:

- Das Vorzeichen ist + oder - und ist optional.
- Ziffernfolgen aus mehr als einer Ziffer beginnen nicht mit 0

Beispiele: 2, -5, +314, 0

Eine Schreibweise von String-Konstanten

"<Folge von Zeichen>"

wobei die Folge von Zeichen eine optionale endliche Folge (Aneinanderreihung) von ASCII-Zeichen ist

Beispiele:

- "" : leerer String
- " " : String enthält genau ein Leerzeichen
- "Hallo"

Anmerkung: Der zu Strings gehörende Datentyp wird im Vorkurs in Thema 6 besprochen.

Benötigt man eine Konstante öfter, so kann man ihr einen symbolischen Namen geben. Das verbessert die Übersichtlichkeit und Wartbarkeit des Programms.

Beispiel

Die Standardbibliothek `limits.h` enthält einige ganzzahlige symbolische Konstanten:

- `INT_MAX`: Maximaler Wert für `int`
- `INT_MIN`: Minimaler Wert für `int`

Man kann auch selbst symbolische Konstanten definieren (nicht Teil des Vorkurses)

Definition

Eine Variable ist ein **symbolischer Name für einen Speicherbereich**, in dem ein Wert eines bestimmten Datentyps gespeichert werden kann. Variablen dienen der **Speicherung von veränderlichen Daten**.

Bildung von Variablennamen

Der Name einer Variable in C darf aus lateinischen Buchstaben und dem `_`-Zeichen gebildet werden.

Variablen müssen **deklariert** werden, bevor man sie benutzen kann.

Anweisung: Variablen-Deklaration

```
<Typ> <Name>;
```

<Typ>	Datentyp der Variable
-------	-----------------------

<Name>	Variablenname
--------	---------------

Die Deklaration einer Variable ist eine Programmanweisung:

- Diese legt den Datentyp und den Namen einer Variable fest
- Der Compiler ordnet der Variable einen passenden Speicherbereich zu

Beispiele

- `int x;`

Deklaration einer Variable `x` vom Typ `int`

- `int x, y;`

Deklaration von zwei Variablen `x` und `y` jeweils vom Typ `int`

- Ausdrücke sind Programmbestandteile zur Berechnung von Datenwerten.
- Ausdrücke haben dazu einen Wert - man spricht hier auch von **Auswertung eines Ausdrucks**
- Ausdrücke haben einen Datentyp - das ist der Datentyp ihres Werts.

Definition

- Der Name einer `int`-Variable ist ein **ganzzahliger (elementarer) Ausdruck** (z.B. `x`)
- Eine `int`-Konstante ist ein **ganzzahliger (elementarer) Ausdruck** (z.B. `-5`)

(deren Wert / Auswertung ist klar)

Ganzzahlige Ausdrücke lassen sich mit Rechenoperatoren zu neuen ganzzahligen Ausdrücken kombinieren

Ganzzahlige arithmetische Ausdrücke

Sind A und B Ausdrücke mit ganzzahligem Wert (**ganzzahlige Ausdrücke**), so sind auch

- (A) (Klammerung)
- $A + B$ (Addition)
- $A - B$ (Subtraktion)
- $A * B$ (Multiplikation)
- A / B (**ganzzahlige** Division)
- $A \% B$: (Modulo - Rest bei ganzzahliger Division)

ganzzahlige Ausdrücke. Die Operatoren $+$, $-$, $*$, $/$, $\%$ heißen **arithmetische Rechenoperatoren**.

Beispiele

Durch Klammerung und arithmetische Rechenoperatoren können schrittweise immer komplexere Ausdrücke konstruiert werden:

- 5 (hat Wert 5)
- 4 (hat Wert 4)
- $5 + 4$ (hat Wert 9)
- $(5 + 4)$ (hat Wert 9)
- $(5 + 4) / 2$ (hat Wert 4 - ganzzahlige Division!)

Definition

Die Operatoren in einem arithmetischen Ausdruck werden (wie gewohnt nach der Regel *Klammer vor Punkt vor Strich*) in folgender Reihenfolge ausgewertet:

- 1 $()$
- 2 $*, /, \%$ (gleichrangig)
- 3 $+, -$ (gleichrangig)

Gleichrangige Operatoren werden von links nach rechts ausgewertet.

Die Auswertungsreihenfolge beeinflusst den Wert eines Ausdrucks.

Beispiele

- $(5 + 4) / 2$:
 - Zuerst wird die Addition in der Klammer $(5 + 4)$ ausgewertet: deren Wert ist 9
 - Dann wird die Division $9 / 2$ ausgewertet: deren Wert ist 4
- $5 + 4 / 2$:
 - Zuerst wird die Division $4 / 2$ ausgewertet: deren Wert ist 2
 - Dann wird die Addition $5 + 2$ ausgewertet: deren Wert ist 7
 - $5 + (4 / 2)$ hat dieselbe Auswertungsreihenfolge
- $5 + 4 - 2$:
 - Zuerst wird die Addition $5 + 4$ ausgewertet: deren Wert ist 9
 - Dann wird die Subtraktion $9 - 2$ ausgewertet: deren Wert ist 7
 - $(5 + 4) - 2$ hat dieselbe Auswertungsreihenfolge

- Werte von Ausdrücken können durch **Wertzuweisungen** an eine Variable gespeichert werden.
- Der Wert des Ausdrucks muss zum Typ der Variable passen.

Anweisung: Wertzuweisung

```
<Variable> = <Ausdruck>;
```

<Variable>	Variablenname
------------	---------------

<Ausdruck>	Ausdruck
------------	----------

Eine Wertzuweisung ist eine Programmanweisung:

- **Zuerst** wird der Ausdruck ausgewertet.
- **Dann** wird der Wert im Speicherbereich der Variable gespeichert

Beispiel

`r = 5 / 2;`

- Zuerst wird der Ausdruck `5 / 2` ausgewertet: dessen Wert ist 2
- Dann wird `r` der Wert 2 zugewiesen.

- Im Ausdruck auf der rechten Seite steht ein Variablenname für den momentanen Wert der Variable,
- Auf der linken Seite steht er für das Ziel der Zuweisung, also den zugehörigen Speicherbereich!

Beispiel

$x = 3 * y + z;$

- Bestimme die momentanen Werte von y und z
- Werte damit den Ausdruck $3 * y + z$ aus
- Lege das Ergebnis im Speicher für x ab

Eine Variable kann zugleich **links und rechts** in einer Wertzuweisung vorkommen

Beispiel

```
x = x + 1;
```

- Berechne $x + 1$ mit dem momentanen Wert von x
- Lege das Ergebnis im Speicher als neuen Wert für x ab
- kurz: erhöhe den Wert von x um 1

Also ist $x = x + 1$ hier **keine** mathematische Gleichung (die durch beidseitige Subtraktion von x zum Widerspruch $0 = 1$ führen würde)

Vor der ersten Wertzuweisung hat eine Variable einen "zufälligen" **Wert**: Das ist der Wert, der aktuell im zugewiesenen Speicherbereich, z.B. nach Benutzung durch ein anderes Programm, liegt.

Initialisierung

Bevor man eine Variable in einem Rechenausdruck benutzt, sollte man ihr erstmal einen Wert zuweisen (man sagt *die Variable initialisieren*)

Einer Variable kann gleich in der Deklaration ein sinnvoller Wert zugewiesen werden

Initialisierung in der Deklaration

```
<Typ> <Variable> = <Ausdruck>;
```

Beispiel: `int x = 0;`

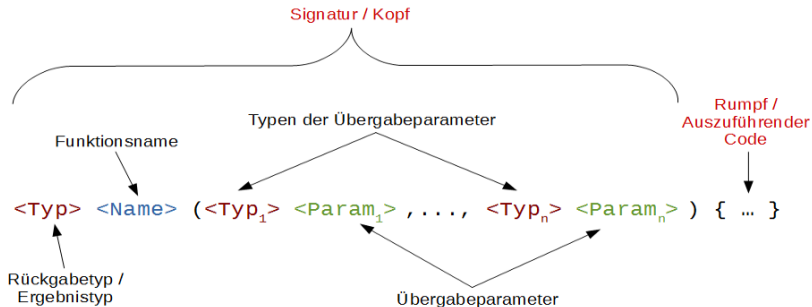
Eine Variable kann (immer) wiederbenutzt werden, d.h. ihr (bisheriger) Wert durch einen neuen Wert überschrieben werden.

Beispiel

```
int x = 5;  
printf("%i", x); /*Wert 5 ausgeben*/  
x = 0; /*Wert überschreiben*/  
printf("%i", x); /*Neuen Wert 0 ausgeben*/
```

Funktionen

Allgemeiner Aufbau



- Es gibt auch Funktionen **ohne Parameter** ($n=0$), dann schreibt man `<Typ> <Name> (void) { ... }`.
- Es gibt auch Funktionen **ohne Rückgabewert**, dann schreibt man `void <Name> (...) { ... }`.

Beispiel (Kopf der main-Funktion)

```
int main(void)
```

int	Typ des Rückgabewerts
main	Funktionsname
keine	Eingabeparameter

Beispiel (Kopf der rand-Funktion)

```
int rand(void)
```

int	Typ des Rückgabewerts
rand	Funktionsname
keine	Eingabeparameter

Beispiel (Kopf der srand-Funktion)

```
void srand(unsigned int seed)
```

void	kein Rückgabewert
srand	Funktionsname
seed	Eingabeparameter vom Typ unsigned int (nicht-negative ganze Zahl)

Der Funktions-Rumpf ist mittels geschweifter Klammern zu einem Block geklammert. Er enthält eine Folge sogenannter **Programmanweisungen**, grundsätzlich jeweils abgeschlossen durch ein Semikolon ;

Programmanweisungen in Kapitel 2

- Variablendeklarationen (bereits bekannt)
- Wertzuweisungen (bereits bekannt)
- Funktionsaufrufe
- return-Anweisung

```
1 int main(void)
2 {
3     int x, y;
4     x = rand();
5     y = 2 * x;
6     printf("Das Ergebnis ist: %i", y);
7     return 0;
8 }
```

- Zeile 3: Deklaration der Variablen x und y.
- Zeile 4: Wertzuweisung
Auf der rechten Seite steht ein Aufruf der Funktion rand (als Ausdruck) - dieser erzeugt eine Zufallszahl (siehe später)
- Zeile 5: Wertzuweisung
- Zeile 6: Aufruf der Funktion printf (als Programmanweisung)
- dieser sorgt für eine Ausgabe (siehe später)
- Zeile 7: return-Anweisung (siehe später)

return-Anweisung von main

`return <Ausdruck>;`

- `<Ausdruck>` muss ein ganzzahliger Ausdruck sein und bestimmt den sog. **Rückgabewert**.
- Rückgabewert 0: Programm erfolgreich beendet
- Anderer Rückgabewert: Programm mit Fehler beendet
- Die `return`-Anweisung beendet die Funktion.

Der das Programm aufrufende Prozess (z.B. das Betriebssystem) kann den Rückgabewert von `main` benutzen (nicht Teil des Vorkurses)

- Die printf-Funktion ist eine vordefinierte und -installierte Bibliotheksfunktion aus der sog. **Header-Datei** `stdio.h`
- Zu deren Benutzung muss man `stdio.h` mit einer `include`-Anweisung ins Programm einbinden

Was leistet printf?

Mit printf können formatierte Folgen von Zeichen auf Kommandozeile ausgegeben werden.

- Erste Benutzung: folgende Folien
- Weiterführende Benutzung: folgende Kapitel und https://de.wikibooks.org/wiki/C-Programmierung:_Einfache_Ein-_und_Ausgabe

Anweisung: printf-Aufruf ohne Umwandlungsangaben

Falls ein String `<String>` keine Umwandlungsangaben (siehe nächste Folie) enthält, so gibt der Funktionsaufruf

```
printf(<String>);
```

`<String>` auf Kommandozeile aus.

Ein Funktionsaufruf von `printf` ist eine Programmanweisung.

Anweisung: printf-Aufruf mit Umwandlungsangaben für ganze Zahlen

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

- %i ist eine Umwandlungsangabe für ganze Zahlen
- Für jede Umwandlungsangabe in <String> muss bei Aufruf ein ganzzahliger Ausdruck als zusätzlicher Eingabeparameter übergeben werden.
- Die Werte der zusätzlichen Ausdrücke werden für die Ausgabe von links nach rechts für die Umwandlungsangaben in <String> eingesetzt.

Das Sonderzeichen % wird mit %% ausgegeben

Beispiele

- Der Funktionsaufruf
`printf("Das Quadrat von %i ist %i", 2, 2 * 2)`
bewirkt die Ausgabe:
Das Quadrat von 2 ist 4
- Hat x den Wert 1, so bewirkt der Funktionsaufruf
`printf("Der Zinssatz ist %i%%", x)`
die Ausgabe:
Der Zinssatz ist 1%

Es gibt noch andere Umwandlungsangaben für ganze Zahlen, die ein anderes Format der Ausgabe bewirken (Eigenrecherche, Übungsaufgaben)

- Die rand-Funktion ist eine vordefinierte und -installierte Bibliotheksfunktion aus der sog. **Header-Datei** `stdlib.h`
- Zu deren Benutzung muss man `stdlib.h` mit einer `include`-Anweisung ins Programm einbinden

Was leistet rand?

Ein Aufruf von `rand` gibt eine ganze Pseudo-Zufallszahl von 0 bis `RAND_MAX` zurück (jeweils einschließlich, `RAND_MAX` ist eine symbolische Konstante aus `stdlib.h`).

Der Funktionsaufruf sieht so aus:

```
rand()
```

Doch wie lässt sich dieser benutzen?

Funktionsausdrücke

Hat eine Funktion einen Rückgabewert, so ist ein Aufruf dieser Funktion ein Ausdruck, der mit seinem Rückgabewert ausgewertet wird.

Benutzung

Der Rückgabewert einer Funktion kann mit Wertzuweisung in einer Variable gespeichert werden:

```
<Variable> = <Funktionsaufruf>;
```

Beispiel:

```
int x = rand();
```

Speichert eine nicht-negative ganze Pseudo-Zufallszahl in x.

- Die `srand`-Funktion ist eine vordefinierte und -installierte Bibliotheksfunktion aus der sog. **Header-Datei** `stdlib.h`

Was leistet `srand`?

Der Funktionsaufruf sieht so aus:

```
srand(<Wert>);
```

Er setzt den Anfangswert für die Berechnung von Zufallszahlen durch `rand` auf `<Wert>`.

- Die `srand`-Funktion ist eine vordefinierte und -installierte Bibliotheksfunktion aus der sog. **Header-Datei** `stdlib.h`

Anwendung

Einmal bei Programmstart aufrufen

```
srand(time(NULL));
```

Setzt bei jedem Programmstart einen neuen Anfangswert (mehr in Informatik 1):

- `time(NULL)` = Anzahl der Sekunden seit 1.1.1970 Mitternacht
- Zur Benutzung der `time`-Funktion die Bibliothek `time.h` einbinden)

Anweisungsblock

Ein Anweisungsblock in C ist von der Form

```
{  
    <Anweisungen>  
}
```

Beispiel: Funktionsrümpfe

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;
```

Ausblick

Es gibt noch zwei weitere Arten von Variablen

- **Globale Variablen:** Werden außerhalb aller Funktionsrumpfe einer Programmdatei deklariert und sind in der gesamten Programmdatei (in allen Funktionsrumpfen) gültig (benutzbar). Benutzen wir im Vorkurs nicht (haben nur spezielle sinnvolle Anwendungsfälle)!
- **Statische Variablen:** Benutzen wir im Vorkurs nicht (haben nur spezielle sinnvolle Anwendungsfälle)!

Die include-Anweisung

Macht vordefinierte Funktionen und Datentypen aus sog. **Bibliotheken** benutzbar

Einige Bibliotheken

- `stdio.h` (Standard Input Output): Sammlung von Funktionen zur Eingabe und Ausgabe (z.B. `printf`)
- `stdlib.h`: Sammlung von Hilfsfunktionen-Funktionen verschiedener Art (z.B. `rand`)

Anweisung: Einbindung von Bibliotheken

```
#include <Bibliotheksname>
```

Benötigte Bibliotheken müssen zu Beginn des Quellcodes eingebunden werden.

Variante 1: In `/* */` eingeschlossener, auch mehrzeiliger, Text

Variante 2: Einzeiliger Text nach `//`

- optional
- Text zur Erläuterung für den menschlichen Leser
- erhöht Verständlichkeit
- wird bei der Übersetzung in Maschinencode und Programmausführung ignoriert

Konventionen für Variablennamen

- Variablennamen sollten sprechend sein (die Bedeutung des gespeicherten Werts beschreiben)
- Mehrere Worte sollen durch das `_`-Zeichen getrennt werden
- Gute Variablennamen: `alter_hund`
- Schlechte Variablennamen: `ah`, `alterhund`

Konventionen für Funktionen

- Rumpf beginnt in der Zeile unter dem Kopf; Abschließendes `}`-Zeichen in separate Zeile setzen
- Anweisungen im Rumpf um 1 Tab (= 8 Leerzeichen) einrücken

Weitere Konventionen

- Nach jedem Komma ein Leerzeichen setzen:
`int x, y;`
- Kein Leerzeichen nach (und vor):
`rand()`
- Leerzeichen vor und nach zweistelligen Operatoren setzen:
`x + y`
- Nur komplexe, aber keine offensichtlichen Sachverhalte kommentieren
- Variablen nach Möglichkeit lokal deklarieren

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

- `int`: Datentyp für ganze Zahlen
- `main`: Name der Hauptfunktion
- `printf`, `rand`, `srand`: Bibliotheksfunktionen
- `return`: Festlegung eines Rückgabewerts
- `void`: Für Funktionen ohne Eingabeparameter
- `include`: Einbindung von Bibliotheksfunktionen

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

Beispielprogramm

```
1 int main(void)
2 {
3     int x, y;
4     x = rand();
5     y = 2 * x;
6     printf("Das Ergebnis ist: %i", y);
7     return 0;
8 }
```

- Zeile 3: Ganzzahl-Variablen x und y deklarieren
- Zeile 4: x einen Zufallswert zwischen 0 und 32767 zuweisen
- Zeile 5: y den doppelten Wert von x zuweisen
- Zeile 6: Den Wert von y mit Umwandlungsangabe %i ausgeben
- Zeile 7: main beenden mit Rückgabe 0 (Erfolgsfall)