1. Einführung in die Programmiersprache C

1.1. Hinweise

Dieses Skriptum umfasst den Unterrichtsstoff für POS (Programmieren und Software Engineering) des ersten Jahrgangs der Abteilung Informatik an der HTBLA Kaindorf.

Falls aus dem Unterricht etwas unklar bleibt oder man sich für Details interessiert, steht ein frei verfügbares Lehrbuch zur Programmiersprache C unter dem Link: http://openbook.rheinwerk-verlag.de/c_von_a_bis_z/ zur Verfügung.

1.2. Weitere Literatur

Sedgewick, R.; Wayne, K.: Algorithms. Fourth Edition. Addison-Wesley Professional 2011: Homepage:
https://algs4.cs.princeton.edu/home/
PDF-Download: https://github.com/Mcdonoughd/CS2223/raw/master/Books/Algorithhms 4th Edition by Robert Sedgewick%2C Kevin Wayne.pdf

Für Rätselfüchse: Projekt Euler:
https://projecteuler.net/

1.3. Inhaltsverzeichnis

1.4. Übersicht – Unterrichtsstoff POS, 1. Jahrgang

- Windows-Command-Shell (cmd): `dir`, `mkdir`, `ren`, `rmdir`, `cd`, `copy`, `del`, `help <command>`, ...
- Development Environment Visual Studio Code
- Programmiersprachenunabhängiger Programmentwurf mittels Struktogrammen (Nassi-Shneiderman-Diagrammen), Structorizer
- Formatierte Ausgabe, `printf()`-Befehl
- Eingabe mit `scanf()`-Befehl und Menüführung, sobald Schleifen bekannt
- Datentypen für Ganzzahlen, Gleitkommazahlen, Characterliterale.
- Arithmetische Operatoren (`+`, `-`, `*`, `/`, `%`, `++`, `--`) inklusive Ganzzahl-Division und Modulo
- Felder mit einfachen Datentypen (auch mit Characterliterale)
- Vergleichs- und logische Operatoren

- Entscheidung (if-else, switch-case)
- Zufallszahlengenerierung, ganzzahlig und mit Gleitkommazahlen
- Kaufmännisch Runden bzw. Abschneiden mit expliziter Typumwandlung (typecast)
- Schleifen:
    - while
    - do-while
    - for (mit inkrementierender und dekrementierender Zählvariable, Inkrement und Dekrement ungleich Schrittweite 1)
    - `continue`, `break` im Schleifenkontext
    - Schreibtischtest
- Zahlensysteme: Binär, Dezimal, Hexadezimal; Umwandlung; 2er-Komplement, Horner-Schema in Form von Beispielen
- Stringliterale
- Programmstrukturierung mittels Funktionen und Vorwärtsdeklarationen, Call-by-Value, Call-by-Reference, Array-Übergabe

Optional

- Rekursion versus Iteration (am Beispiel Fakultätsberechnung und Fibonacci-Zahlen)
- 2-dimensionale Arrays
- Fehlersuche (mit printf()-Ausgaben, Debugger)

AB HIER MIT REVIEW FORTSETZEN

2. Quick Start

2.1. Wichtige Begriffe für Programmierer

Ein **Programm** ist eine Abfolge von Befehlen/Anweisungen und wird meist in einer höheren Programmiersprache geschrieben.

Ein **Compiler** übersetzt ein Programm, geschrieben in einer höheren Programmiersprache (z.B. C, Java, Python) in Maschinensprache (01100010001...).

Ein **Debugger** dient zur Fehlersuche in Programmen.

Eine **Entwicklungsumgebung** (IDE ... Integrated Development Environment) wie z.B. QtCreator unterstützt Programmierer beim

- Schreiben von Programmen
- Debuggen von Programmen
- Compilieren und Ausführen von Programmen

Jede Programmiersprache stellt **Befehle** für:

1. Eingabe
2. Ausgabe
3. Mathematische Operationen
4. Entscheidungen
5. Schleifen

zur Verfügung.

Zudem stellt jede Programmiersprache **Datentypen** für

- Ganze Zahlen
- Gleitkommazahlen
- Einen logischen Datentyp
- Literale (Zeichen und Zeichenketten)

bereit.

2.2. Geschichte von C

C ist eine imperative und prozedurale Programmiersprache. C wurde 1972 „erfunden" und seit damals stets in Varianten weiterentwickelt. Das American National Standards Institute (ANSI) vereinheitlichte diese C-Varianten und veröffentlichte 1989 ein standardisiertes C. Dieser C-Standard wird im Sprachgebrauch als C89 bzw. ANSI-C bezeichnet. Er wurde von der International Organization for Standardization (ISO) übernommen und wird im Sprachgebrauch auch als C90 bezeichnet.

Im Jahr 1999 erschien C99 mit Elementen der Programmiersprache C++.

2.3. Installation von Visual Studio Code

Achtung: Der Open-Source-Code-Editor Microsoft **Visual Studio Code** darf nicht verwechselt werden mit der viel umfangreicheren integrierten Entwicklungsumgebung (IDE) „Visual Studio 2019".

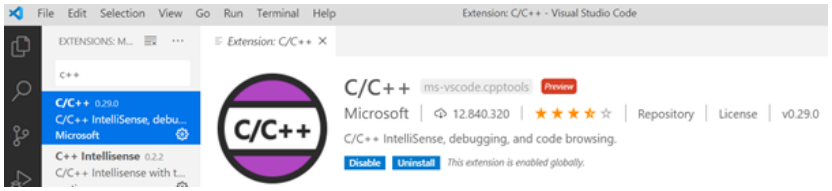
Hier der Download-Link:
https://code.visualstudio.com/

Mingw-w64 (oder **Mingw64**) ist eine Portierung der Linux-Entwicklerwerkzeuge GNU Compiler Collection (GCC) und GNU Debugger (GDB) auf Windows.

Homepage: http://mingw-w64.org/

Hier der Download-Link:
https://sourceforge.net/projects/mingw-w64/files/Toolchains targetting Win32/Personal Builds/mingw-builds/installer/mingw-w64-install.exe

Wie starte ich Visual Studio Code? Einfach in der Windows-Suche "code" eingeben und App öffnen.
Wichtig ist noch: Menü „View" und „Extensions" öffnen. Im Suchfeld „c++" eingeben und die **C/C++ Extension** von Microsoft installieren:



Danach noch **Umgebungsvariable** konfigurieren (https://code.visualstudio.com/docs/cpp/config-mingw): Windows-Suche: "Einstellungen"
Suchfeld: „Umgebungsvariablen für dieses Konto bearbeiten"
Variable `Path` auswählen und „Bearbeiten..." drücken Drücke „Neu" und füge den Pfad zum „\bin"-Verzeichnis deiner mingw-w64-Installation hinzu (z.B. C:\Program Files (x86)\mingw-w64\i686-8.1.0-posix-dwarf-rt_v6-rev0\mingw32\bin)

Installation und Konfiguration prüfen: Visual Studio Code starten: Windows-Suche: „code" eingeben.
Terminal (Konsole) öffnen: Menü „View -> Terminal"
In der Kommando-Zeile „g++ --version" eingeben:

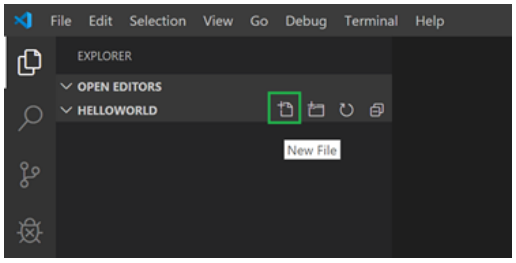
2.4. Erstes Programm mit Visual Studio Code (VSC)

In Windows-Suche „cmd" eingeben und Eingabeaufforderung öffnen.

Projektverzeichnis erstellen und VSC starten:

```
mkdir projects
cd projects
mkdir helloworld
cd helloworld
code .
```

Quelldatei "helloworld.c" erstellen:




```c
// #include bewirkt das Einfügen von Quellcode aus einer anderen
// Datei. Die Header-Datei stdio.h wird in das Programm eingefügt.
// Sie beinhaltet die Funktionen, die wir zur
// (Standard)-Ein/-Ausgabe benötigen
#include <stdio.h>

int main()         // Funktion main() definiert das Hauptprogramm
{                  // definiert den Anfang der Funktion
  printf("Hello World!\n");        // Aufruf der Funktion printf()
  printf("Press a key to continue... ");
  fflush(stdin);
  getchar();
  return 0;                    // Rückgabewert der Funktion main()
}                              // definiert das Ende der Funktion
```

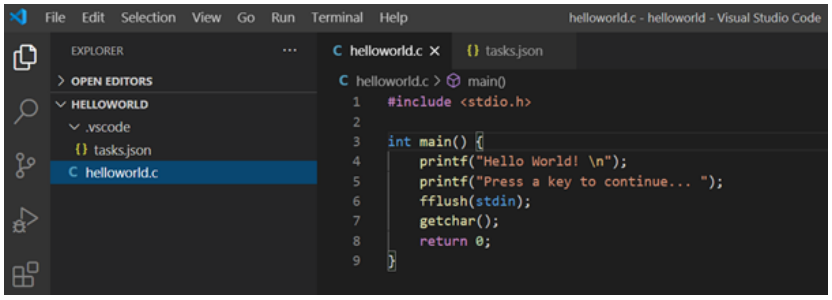
Datei speichern: Menü File -> Save

2.5. `helloworld.c` kompilieren

Um VSC zu sagen, wie die Quelldatei helloworld.c nun zu kompilieren ist, braucht es eine tasks.json Datei:

Menü Terminal -> Configure Default Build Task..., "C/C++:gcc.exe build active file" auswählen.

Quelldatei helloworld.c auswählen



und mit der Tastenkombination Strg+Shift+b kompilieren.

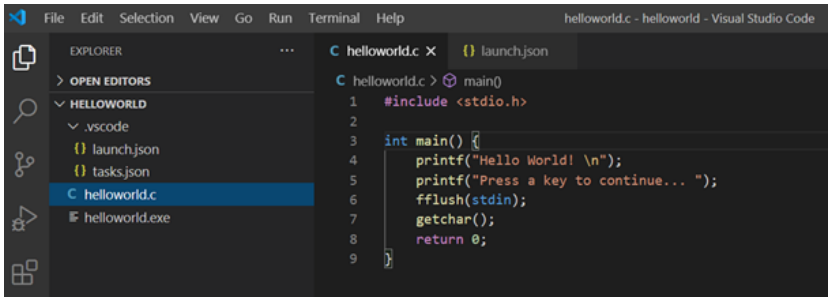
2.6. `helloworld.c` starten/debuggen

Um helloworld.c starten oder debuggen zu können, braucht VSC die Datei launch.json:

Menü Run -> Add Configuration..., „C++ (GDB/LLDB)" und "gcc.exe – Aktive Datei erstellen und debuggen" auswählen.

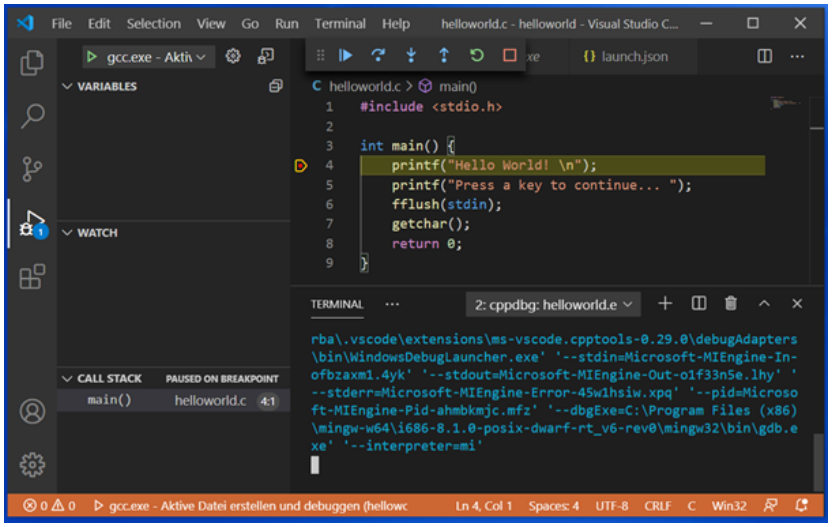
Das Programm wird anschließend gleich ausgeführt was man mit der Tastenkombination Shift+F5 stoppen kann.

Quelldatei helloworld.c auswählen



und mit der Tastenkombination Strg+F5 starten.

Will man das Programm debuggen, an geeigneter Stelle einen Breakpoint setzen und das Programm mit der Taste F5 starten:



2.7. Einführungsbeispiel

Lernt man eine neue Programmiersprache, dann ist meist ein „Hello World" Programm, das erste Programm, das man schreibt. Und das machen wir im folgenden Einführungsbeispiel genauso.

Das folgende „Hello World" Programm zeigt die grundlegende Struktur eines C-Programms:

```c
// #include bewirkt das Einfügen von Quellcode aus einer anderen
// Datei. Die Header-Datei stdio.h wird in das Programm eingefügt.
// Sie beinhaltet die Funktionen, die wir zur
// (Standard)-Ein/-Ausgabe benötigen
#include <stdio.h>

int main()          // Funktion main() definiert das Hauptprogramm
{                   // definiert den Anfang der Funktion
  printf ("Hello World!\n");       // Aufruf der Funktion printf()
  return 0;         // Rückgabewert der Funktion main()
}                   // definiert das Ende der Funktion
```

Erklärungen:

- die Funktion `main()` muss in jedem C-Programm genau einmal vorhanden sein

- die geschweiften Klammern definieren einen Funktionskörper oder Block
- `printf()` ist eine Funktion zur formatierten Ausgabe auf der Konsole
- Parameter der Funktion `printf()` ist hier der String `"Hello World!\n"` Die Datei `stdio.h` muss inkludiert sein.
- die Zeichenkombination `'\n'` ist das Steuerzeichen für „Zeilenende"
- Jede Anweisung wird mit `;` beendet
- Mehrzeilige Kommentare werden in `/*` und `*/` eingeschlossen
- Einzeilige Kommentare beginn mit `//` und erstrecken sich bis zum Zeilenende
- C unterscheidet zwischen Groß- und Kleinschreibung

Das folgende Listing zeigt eine modifizierte Version des „Hello World" Programms:

```c
#include <stdio.h>
int main()
{
  printf ("Hello");
  printf (" World!");
  printf ("\n");
  return 0;
}
```

3. Mein erstes Programm mit Eingabe, Zuweisung und Ausgabe

Im folgenden Listing werden zuerst

- Die Variable a vom Datentyp `int` (Integer ... Ganzzahl)
- Die Variable x vom Datentyp `float` (Float ... Fließkommazahl)
- Die Variable y vom Datentyp `double` (Double float ... Fließkommazahl mit doppelter Genauigkeit)

definiert.

Danach wird dem Anwender mit dem `printf()`-Befehl ausgegeben, dass er bitte entsprechende Zahlen eingibt.
Mit dem `scanf()`-Befehl werden die Eingaben des Anwenders der jeweils passenden Variable (a, x und y) zugewiesen.
Zum Schluss werden die Werte der Variablen (a, x und y) dem Anwender mit dem `printf()`-Befehl an der Konsole ausgegeben.

```c
#include <stdio.h>

int main()
{
    int a;
    float x;
    double y;
    printf("Bitte eine ganze Zahl eingeben: ");
    scanf("%d", &a);
    printf("Bitte eine Kommazahl eingeben : ");
    scanf("%f", &x);
    printf("Bitte eine grosse Kommazahl eingeben: ");
    scanf("%lf", &y);
    printf("------------------------------\n");
    printf("a = %d\n", a);
    printf("x = %f\n", x);
    printf("y = %lf\n", y);
    return 0;
}
```

Erklärungen:

- Beim `scanf()`-Befehl, dienen `%d`, `%f` und `%lf` als Platzhalter, um den vom Anwender eingegebenen Wert einzulesen und der Variable a, x bzw. y zuzuweisen.
- `%d` ist ein Platzhalter für eine Ganzzahl (d ... decimal)
- `%f` ist ein Platzhalter für eine Fließkommazahl (f ... float)
- `%lf` ist ein Platzhalter für eine Fließkommazahl mit doppelter Genauigkeit (lf ... long float)
- Die Platzhalter für die Ausgabe der Variablen a, x und y mit dem `printf()`-Befehl sind gleich wie beim `scanf()`-Befehl.
- Das &-Zeichen im scanf()-Befehl wird zu einem späteren Zeitpunkt erklärt.

3.1. `printf()`-Befehl: Steuerzeichen und Sonderzeichen

Sogenannte Escape-Sequenzen ermöglichen Steuerzeichen und Sonderzeichen auszugeben. Eine Escape-Sequenz wird durch einen Backslash \ eingeleitet, dem ein weiteres Zeichen folgt. Eine Escape-Sequenz gilt trotzdem als einzelnes Zeichen. In der folgenden Tabelle sind die wichtigsten Escape-Sequenzen aufgelistet:

Escape-Sequenz	Bedeutung
`\n`	Zeilenumbruch
`\t`	Tabulator
`\'`	Einfaches Hochkomma

Escape-Sequenz	Bedeutung
\"	Doppeltes Hochkomma
\?	Fragezeichen
\\	Backslash
\0	NULL zum Terminieren von Strings
\b	Backspace - Zurücksetzen um ein Zeichen
\f	Formfeed - Seitenumbruch
\a	Piepston

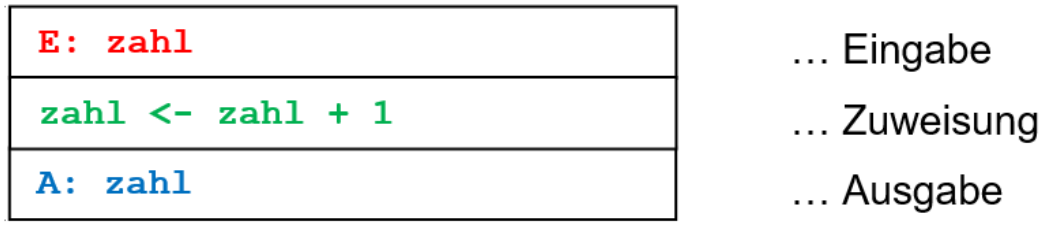
3.2. Struktogramm

Mit Struktogrammen, auch Nassi-Shneiderman-Diagramme genannt, werden Programme unabhängig von der Programmiersprache, mit der sie letztendlich umgesetzt werden, entworfen.
Ein Struktogramm setzt sich aus Strukturblöcken zusammen. Diese können, wie wir noch sehen werden, ineinander verschachtelt sein.

Beispiel eines Struktogramms mit Sequenz-Symbolen:

Aufgabe: Lese eine Ganzzahl ein, erhöhe diese um eins und gib die Zahl aus.

Struktogramm:



Programm:

```c
#include <stdio.h>

int main()
{
    int zahl;                       // ROT
    printf("Ganze Zahl: ");         // ROT
    scanf("%d", &zahl);             // ROT
    zahl = zahl + 1;                // GRÜN
    printf("Ergebnis: %d\n", zahl);  // BLAU
    return 0;
}
```

Vorteil von Struktogrammen:
Man braucht sich beim Entwurf des Programms noch nicht mit Befehls- und Datentypdetails einer Programmiersprache auseinandersetzen.

4. Operatoren – 1. Teil

Es gibt in C drei Arten von Operatoren:

- Unäre Operatoren: benötigen nur einen Operanden, wie z.B. die Vorzeichenoperatoren + und -.
- Binäre Operatoren: benötigen zwei Operanden
- Ternäre Operatoren: benötigen drei Operanden

Im folgenden Abschnitt werden die wichtigsten Operatoren beschrieben:

4.1. Zuweisungsoperator

Der Zuweisungsoperator = weist der Variablen auf der linken Seite den Wert des Ausdrucks der rechten Seite zu.

4.1.1. Variablentausch mit Hilfsvariable

Oft müssen die Inhalte zweier Variablen (z.B. a, b) vertauscht werden. Am einfachsten macht man das mit einer Hilfsvariablen h:

```
int a, b, h;
...
h=a;
a=b;
b=h;
```

4.2. Variablentausch ohne Hilfsvariable

Mathematikfüchse kommen ohne Hilfsvariable aus:

```
int a, b;
...
a=a+b;
b=a-b;
a=a-b;
```

4.3. Arithmetische Operatoren

4.4. Vorzeichenoperatoren

Zur Darstellung der mathematischen Vorzeichen 'plus' und 'minus' werden die unären Vorzeichenoperatoren + und – verwendet.

```
int z1, z2;
z1 = +5;
z2 = -9;
```

4.4.1. Binäre arithmetische Operatoren

Operator	Bedeutung
+	Addition
-	Subtraktion
*	Multiplikation
/	Division
%	Modulo-Division

Regeln für die arithmetischen Operatoren:

Wie in der Mathematik gilt Punkt- vor Strichrechnung, wobei der Modulooperator ein Punktoperator ist.

Gleichrangige Operatoren werden von links nach rechts ausgewertet.

- Sind beide Operanden vom selben Datentyp so ist auch das Ergebnis der Berechnung von diesem Datentyp, mindestens jedoch vom Typ `int`.

Sind die Operanden von unterschiedlichen Datentypen so ist das Ergebnis vom größeren der beiden Datentypen.

- Der Modulooperator ist nur für ganzzahlige Operanden zulässig. Das Ergebnis ist der ganzzahlige Rest der Division: `17 % 5 = 2`. Bei den Divisionsoperatoren (`/` und `%`) führt eine Divison durch `0` zu einem Laufzeitfehler.

5. Datentypen

In C gibt es 4 Grunddatentypen. Deren Größe und Wertebereich sind nicht normiert. Bei den meisten Systemen gelten aber folgende Werte:

Name	Datentyp	Bytes	Minimalwert	Maximalwert
`int`	Ganze Zahl	4	-2147483648 (= -2^{31})	2147483647 (= $+2^{31}-1$)
`char`	Ein Zeichen	1	-128 (= -2^{7})	127 (= $+2^{7}-1$)
`float`	Fließkommazahl, 6-stellige Genauigkeit	4	1.175494E-038	3.402813E+038
`double`	Große Fließkommazahl, 15-stellige Genauigkeit	8	2.225074E-308	1.797693E+308

In C existiert kein logischer Datentyp (Datentyp boolean)! Stattdessen werden Variablen vom Datentyp int für Bedingungen verwendet werden, wobei folgende Regeln gelten:

Variablenwert gleich 0 -> `FALSE`
Variablenwert ungleich 0 -> `TRUE`

Alle Variablen müssen vor ihrer Verwendung deklariert werden. Die Deklaration legt den Namen und den Typ fest. Deklarationen müssen immer am Anfang eines Programms stehen.

6. Zufallszahlen

Zum Erzeugen von ganzzahligen Zufallszahlen stehen in der C Standard Library `<stdlib.h>` folgende Funktionen zu Verfügung:

```
srand(<seed>)       // Anweisung zum Erzeugen eines Zufallszahlen-Seeds
srand(time(NULL))   // Erzeugen eines Zufallszahlen-Seeds,
```

```
                    // initialisiert mit der aktuellen Systemzeit
rand()              // Erzeugen einer ganzzahligen Zufallszahl
```

Für die Funktion `time()` muss zusätzlich die Datei `<time.h>` inkludiert werden. Die Funktion `srand()` definiert den Startwert (Seed) zum Erzeugen der Zufallszahlen. Durch die Übergabe der Funktion `time()` an `srand()` wird sichergestellt, dass bei jedem Programmstart neue Zufallszahlen erzeugt werden. Die Funktion `rand()` gibt eine Zufallszahl im Bereich von `[0,RAND_MAX]` zurück. Wobei die symbolische Konstante `RAND_MAX` dem Wert 32767 entspricht. Durch Verwendung des Modulo-Operators (`%`) kann der Wertebereich eingeschränkt werden.

Benötigt man eine Zufallszahl im Intervall `[Untergrenze, Obergrenze]` gilt folgende Formel:

```
rand()%(Obergrenze + 1 – Untergrenze) + Untergrenze
```

Im folgenden Beispiel wird eine Zufallszahl zwischen `[1, 100]` erzeugt:

```
#include <stdlib.h>
#include <time.h>

int main()
{
  int zz;
  srand(time(NULL));
  zz = rand()%100 + 1;
}
```

Wird eine Zufallszahl vom Datentyp `double` benötigt, so lässt sich folgende Formel anwenden:

```
(double)rand()/RAND_MAX*(Obergrenze – Untergrenze) + Untergrenze
```

Im folgenden Beispiel wird eine Zufallszahl zwischen `[1.0, 5.0]` erzeugt:

```
#include <stdlib.h>
#include <time.h>

int main()
{
  double zz;
  srand(time(NULL));
  zz = (double)rand()/RAND_MAX*(5.0-1.0) + 1.0;
}
```

7. `printf()`-Befehl: Formatierte Ausgabe

Die formatierte Ausgabe auf der Konsole erfolgt mit dem `printf()`-Befehl, der in der Headerdatei `<stdio.h>` definiert ist:

```
int printf("Formatstring" [ , argument1, argument2 ...]);
```

Der erste Parameter dient zur Übergabe eines Formatstrings. Die weiteren Parameter sind optional (eckige Klammern bedeuten optional) und sind die Argumente für den Formatstring. Die Formatierung erfolgt entsprechend den Angaben im Formatstring. Der Rückgabewert des `printf()`-Befehls ist die Anzahl der ausgegebenen Zeichen.

Der Formatstring enthält sowohl normalen Text als auch Platzhalter mit den jeweiligen Formatangaben. Die Syntax für einen Platzhalter sieht wie folgt aus:

```
%[flag][width][.precision]conversion
```

Der wichtigste und einzige nicht optionale Teil ist dabei das Umwandlungszeichen (conversion), das den Datentyp für die Umwandlung bestimmt:

`conversion`	**Datentyp**	**Bedeutung**
`d`, `i`	int	Integer
`f`	float	Gleitkommazahl float - Default von 6 Kommastellen
`lf`	double	Gleitkommazahl double - Default von 6 Kommastellen
`%`		Ausgabe des % Zeichens

Die Flags sind optional und werden unmittelbar nach dem `%`-Zeichen angegeben:

`flag`	**Bedeutung**
-	Ausgabe linksbündig
+	Ausgabe mit Vorzeichen

flag	Bedeutung
0	Leerzeichen werden mit Nullen gefüllt

Die Anzahl der auszugebenden Zeichen ist optional und wird durch die Breite (`width`) festgelegt.

Die Anzahl der Nachkommastellen ist optional und wird durch die Genauigkeit (`precision`) festgelegt.

Beispiele von Formatangaben für ganzzahlige Werte (Datentyp `int`):

```
%d      Ausgabe der Zahl ohne zusätzliche Vorgabe
%5d     rechtsbündige Ausgabe der Zahl auf 5 Stellen
%-5d    linksbündige Ausgabe der Zahl auf 5 Stellen
%05d    Ausgabe der Zahl auf 5 Stellen mit führenden Nullen
%+5d    rechtsbündige Ausgabe der Zahl auf 5 Stellen mit Vorzeichen
%+-5d   linksbündige Ausgabe der Zahl auf 5 Stellen mit Vorzeichen
```

Beispiele von Formatangaben für Fließkommazahlen (Datentyp `float`):

```
%f          Ausgabe der Zahl ohne zusätzliche Vorgabe
%10f        rechtsbündige Ausgabe der Zahl auf 10 Stellen
%.2f        rechtsbündige Ausgabe der Zahl mit 2 Nachkommastellen
%-10.2f     linksbündige Ausgabe auf 10 Stellen, davon 2 Nachkommastellen
%010.2f     Ausgabe der Zahl auf 10 Stellen mit führenden Nullen, davon 2
            Nachkommastellen
%+10.2f     rechtsbündige Ausgabe der Zahl auf 10 Stellen, davon mit Vorzeichen und 2
            Nachkommastellen
%+-.2f      linksbündige Ausgabe der Zahl mit Vorzeichen und 2 Nachkommastellen
```

8. Mathematische Funktionen – `<math.h>`

In der Headerdatei `<math.h>` sind folgende nützliche mathematische Funktionen und Konstanten deklariert:

```
double pow(double x, double y)  // x hoch y
double sqrt(double x)           // Wurzel aus x
double fabs(double z)           // Absolutwert von x
double sin(double x)            // Sinus von x
double cos(double x)            // Cosinus von x
double tan(double x)            // Tangens von x
double exp(double x)            // Exponentialfunktion e hoch x
double log(double x)            // Nat. Logarithmus von x

M_PI    // symbolische Konstante für PI
M_PI_2  // symbolische Konstante für PI/2
M_PI_4  // symbolische Konstante für PI/4
M_E     // symbolische Konstante für e
```

Fürs kaufmännisch Runden, Abschneiden, Abrunden und Aufrunden stehen folgende Funktionen zur Verfügung:

```
double round(double x);  // Kaufmännisch Runden
double trunc(double x);  // Kommazahlen abschneiden
double floor(double x);  // auf nächste Ganzzahl abrunden
double ceil(double x);   // auf nächste Ganzzahl aufrunden
```

Im folgenden Beispiel wird die Verwendung der Funktionen veranschaulicht:

```c
#include <stdlib.h>
#include <math.h>

int main()
{
    double x1, x2, y1, y2;
    x1 = 5.23487392;
    x2 = 5.89437524;
    y1 = round(x1 * 1000)/1000;  // ergibt 5.235000
    y2 = round(x2 * 1000)/1000;  // ergibt 5.894000
    y1 = trunc(x1);              // ergibt 5.000000
    y1 = floor(x1);              // ergibt 5.000000
    y1 = ceil(x1);               // ergibt 6.000000
}
```

Kaufmännisch Runden und Abschneiden ist auch durch explizite Typumwandlung möglich:

```c
#include <stdio.h>

int main()
{
```

```
    double x1, x2;
    int y1, y2;
    double y3, y4;
    x1 = 5.23487392;
    x2 = 5.89437524;
    // Kaufmännisch Runden:
    y1 = (int)(x1 + 0.5);               // ergibt 5
    y2 = (int)(x2 + 0.5);                // ergibt 6
    // auf z.B. 3 Kommastellen Runden:
    y3 = (int)(x1 * 1000 + 0.5)/1000.0;  // ergibt 5.235000
    y4 = (int)(x2 * 1000 + 0.5)/1000.0;  // ergibt 5.894000
    // Kommastellen abschneiden:
    y1 = (int)x1;                        // ergibt 5
    // Zahl nach z.B. der 3. Kommastelle abschneiden:
    y3 = (int)(x1 * 1000)/1000.0;        // ergibt 5.234000
}
```

9. Kommentare

Regeln für Kommentare:

- Mehrzeilige Kommentare werden mit `/*` eingeleitet und mit `*/` abgeschlossen.
- Einzeilige Kommentare beginnen mit `//` und erstrecken sich bis zum Ende der Zeile.
- Kommentare können an beliebiger Stelle stehen.

10. Character- und Stringliterale

10.1. Characterliterale

Die Zuweisung erfolgt durch genau ein Zeichen, das in einfachen Anführungszeichen eingeschlossen wird. Auf der char-Variablen wird der entsprechende ASCII-Wert des Zeichens gespeichert.

```
char c1, c2, c3;

c1 = 'A';       // numerischer Wert 65
c1 = 65;        // identisch zur vorherigen Anweisung
c2 = 'F';       // numerischer Wert 70
c3 = '1';       // numerischer Wert 49
c4 = '\n';      // Zeilenumbruch
```

Zu den Characterliteralen gehören auch die Sonder- und Steuerzeichen, die mit einem Backslash '\' eingeleitet werden (Escape-Sequenz) und trotzdem als einzelne Zeichen gelten.

Das Umwandlungszeichen (conversion), um Characterliterale einzulesen bzw. auszugeben lautet „c“:

```
char zeichen;
scanf("%c", &zeichen);
printf("%c", zeichen);
```

10.2. Stringliterale

Für Strings gibt es in C keinen eigenen Datentyp, daher müssen Zeichenketten in Form von `char`-Arrays behandelt werden.

Folgende Eigenschaften gelten für Zeichenketten:

- Sie bestehen aus einer Folge von Einzelzeichen.
- Sie werden in doppelte Anführungszeichen eingeschlossen: "Das ist eine Zeichenkette".
- Am Ende muss ein NULL-Zeichen ('`\0`') zur Kennzeichnung des Endes angehängt werden. Es gelten die gleichen Escape-Sequenzen wie bei einzelnen Zeichen.
- Zeichenketten können über Zeilengrenzen gehen, wenn unmittelbar vor dem Zeilentrenner ein Backslash '`\`' steht. Sie können einem Charaterfeld (char-Array) zugewiesen werden. Sie dürfen nur bei der Initialisierung oder beim Einlesen zugewiesen werden, nicht aber zu einem späteren Zeitpunkt.

```
char str1[100] = "";
char str2[] = { "Hallo" };
char str3[] = { 'H', 'a', 'l', 'l', 'o', '\0' };
char str4[] = { 72, 97, 108, 108, 111, 0 };
char str2[20] = "Teil 1\
                 Teil2";       // Der Zeilenumbruch ist auf dem String nicht sichtbar

str1 = "ungültige Zuweisung";  // Compilerfehler!
```

Das Umwandlungszeichen (conversion), um Stringliterale auszugeben lautet „%s“:

```
char str1[] = { "Hallo" };
printf("%s", str1);
```

Fürs Einlesen von Stringliteralen von der Konsole stehen die beiden Befehle `gets(s, max)` und `scanf("%s, s)` zur Verfügung:

```
char str1[100] ="";
gets(str1, 100);
scanf("%s", str1);
```

Um die Länge eines Stringliterals feszustellen, steht der Befehl `strlen(s)` zur Verfügung:

```
char str[] = { "Hallo" };
int strLaenge;
strLaenge = strlen(str);
```

Der Zugriff auf einzelne Zeichen eines Stringliterals erfolgt folgendermaßen:

```
char str[] = { "Hallo" };
char c;
c = str[0];    // Weist der Variablen c das Zeichen ‚H‘ zu
```

Zur Umwandlung von Zeichenketten in numerische Werte stehen in der C-Standard Library `<stdlib.h>` unter anderem folgende Konvertierungsfunktionen zu Verfügung:

```
int atoi(<String>)      // Konvertierung von String in int
double atof(<String>)   // Konvertierung von String in double
```

11. Operatoren – 2. Teil