NewElements IT-Schulungen.com

C - Kurs

Dr.sc.nat. Michael J.M. Wagner, New Elements*

Revision _BUILD

Inhaltsverzeichnis

*michael@wagnertech.de

1 Einführung

1.1 Die Sprache C

Entstehung von C[1]

- Entstehung in den 70-ger Jahren
- Plattformunabhängige Programmiersprache
- Direkte Speicherzugriffe
- Prozedurale Sprache
- Systemprogrammierung
- Hohe Effizienz
- Betriebssystemprogrammierung (Windows, UNIX, ...)
- Steuergeräte, Embedded Devices
- Standardisierung → Abb. 1
- POSIX (Portable Operating System Interface): Standardisiert Schnittstelle zum Betriebssystem, stellt wechselseitige Compilierbarkeit sicher.

Standard	Bemerkung
K&R-C	Diese Version basiert hauptsächlich auf dem ersten Buch zu C von den beiden Autoren Kernighan und Ritchie von 1978.
C89	Die erste echte Standardisierung erfolgte über das American National Standards Institute (ANSI) im Jahre 1989.
C90	Ein Jahr nach dem Erscheinen des 1. Standards wurden kleine Änderungen hinzugefügt und die ISO-Norm C90 definiert. Sie ist Basis vieler heutiger C-Implementierungen. Die wichtigsten Verbesserungen waren die Einführung von Funktionsprototypen sowie die Normierung der C-Standardbibliothek.
C95	1995 wurden in einem neuen Standard Fehlerbehebungen, einige neue Makros sowie die Unterstützung weiterer Zeichensätze zusammengefasst. Obwohl dieser Standard schon relativ alt ist, wird er selten von den gängigen Compilern vollständig implementiert.
C99	Über die Jahre wurden einige häufig vermisste Sprachkonstrukte und Schreibweisen anderer Sprachen hinzugefügt wie der Datentyp `_Bool`, der einzeilige C++- Zeilenkommentator "//" oder die Möglichkeit, Variablen direkt in einer `for`-Schleife zu definieren.
C1X	Korrekturen und geplante Änderungen für den nächsten, noch nicht fertiggestellten Standard C1X (z. B. Multithreading).

Abbildung 1: C-Versionsgeschichte[2]

[1] Garcia (2013): Kap. 2.
[2] Garcia (2013): S. 7.

Sprachelemente

C besteht aus

- dem eigentlichen Sprachkern und
- den Standardbibliotheken

[W:19ff][3]

C wird mit einem Compiler in Maschinencode übersetzt [W:22]. Für effektives und professionelles Arbeiten bietet sich die Verwendung einer *Integrierten Entwicklungsumgebung* (IDE) an.

Hello World-Programm: [W:29]

> Aufgabe:
>
> Starten Sie die Entwicklungsumgebung und legen Sie ein neues Projekt `HelloWorld` an. Übersetzten und starten Sie das Programm.

1.2 Erste Schritte in C

Hello World

Möglichkeiten von `printf`: [W:34]

Möglichkeiten von `scanf`: [W:73f]

Grundsätzlich unterscheidet die C-Syntax zwischen

- Bezeichnern: Vom Benutzer festgelegte Namen
- Schlüsselwörtern [W:39]
- Literalen [W:41]

> Aufgabe:
>
> - Wolf (2016): Kap. 2.6, 1 - 3.
> - Lesen Sie zwei Zahlen unterschiedlichen Typs ein und geben sie diese wieder aus.

Variablen

Definition und Deklaration: [W:46]

Zuweisung und Initialisierung: [W:47]

[3] Diese Verweise beziehen sich auf Wolf (2016).

Numerische Datentypen

Mit Vorzeichen: [W:48]

Ohne Vorzeichen: [W:51f]

Datentyp `char`: [W:54]

Fließkommazahlen: [W:57]

Komplexe Zahlen (C99): [W:59]

Datentyp `bool` (C99): [W:60f]. Achtung: `bool` ist nicht sprachbestandteil. Es wird die Headerdatei `stdbool.h` benötigt.

Bei den Standardtypen ist es plattformabhängig, wieviel Byte für eine Variables des Typs verwendet wird. Die benötigte Größe kann mit dem `sizeof`-Operator abgefragt werden [W:62].

Darüberhinaus gibt es seit C99 auch Ganzzahltypen mit festgelegter Länge [W:66f].

Soll bereits zum Compile-Zeitpunkt eine Zusicherung überprüft werden, kann dies mit `static_assert` (aus `assert.h`) erfolgen [W:68].

Mit dem Schlüsselwort `const` kann der Compiler überwachen, dass eine Variable initialisiert, aber nie verändert wird. Dies ist aber nur mäßig Sicher, da der Speicherplatz der (konstanten) Variable mittels Zeigeroperationen überschrieben werden kann. Das oft verwendete `#define` ist auch keine bessere Lösung: Da es simples „Suchen und Ersetzen" durch den Precompiler dastellt, findet auch keinerlei Typprüfung statt, was wiederum zu Fehlern führen kann.

Zeichenketten

Zeichenketten werden in C als Zeiger auf der erste Zeichen der Kette verwaltet. Damit das System weiß, wo die Kette zu Ende ist, wird jede Zeichenkette mit einem Nullbyte (`\0`) beendet.

Konstante Zeichenketten können so hinterlegt werden:

```
const char* konstante_kette = "Konstante Zette";
```

Soll hingegen variabler Speicher (konstanter Länge) angelegt werden, so kann dies so geschehen:

```
char string_variable[256] = "Initiale Zeichenkette";
```

Zum Arbeiten mit Zeichenketten siehe Kap. TODO.

1.3 Speichermodell

C kennt drei Speicherbereiche, in denen Variablen angelegt werden können: Lademodul, Stapel und Freispeicher. Stapelvariablen werden innerhalb des Programmcodes definiert und sind nur im akteullen Programmblock gültig und zugreifbar [W:70f].

Variablen, die im Lademodul liegen können auf zwei Weisen definiert werden:

- Definition außerhalb eines Programmblocks: [W:162]

  Diese Variablen sind global zugreifbar. Damit sie in einem anderen Modul bekannt sind, müssen sie dort mit `extern` deklariert werden.

- `static`-Variable innerhalb/außerhalb eines Programmblocks: [W:164]

  Diese Variablen sind nur innerhalb des Moduls (c-Datei) zugreifbar.

Für Variablen im Freispeicher muss mit entsprechenden Funktionen Speicher angefordert werden. Der Entwickler ist dafür verantwortlich, dass der Speicher auch wieder frei gegeben wird.

2 Sprachgrundlagen

2.1 Rechnen mit C und Operatoren

Operatoren

Arithmetische Operatoren: [W:80]

Erweiterte Darstellung arithmetischer Operatoren: [W:81]

Inkrement- und Dekrement-Operator: [W:82]

Bit-Operatoren: [W:84f]

Vergleichsoperatoren: [W:102f]

Logische Verknüpfungen: [W:119]

Typumwandlung

Arithemtische Typumwandlung: [W:88f]

Explizites Casting von Typen: [W:91f]

Mathematische Funktionen in C

Headerdateien: [W:92f]

Mathe mit Linux: [W:93]

Beispiel „Wurzelziehen": [W:93-96]

> Aufgabe:
>
> - Legen Sie ein neues Projekt `Mathe` an und bringen Sie die Beispiele `kap004/listing005.c` und `kap004/listing006.c` zum Laufen.
> - Wolf (2016): Kap. 4.9, 1 - 2.

2.2 Bedingte Anweisung und Verzweigung

`if`-Verzweigung: [W:100f]

Alternative Verzweigung: [W:104ff]

Mehrfache Verzweigung mit `if` und `else if`: [W:108ff]

Der Bedingungsoperator ?: [W:107]

Mehrfache Verzweigung mit switch: [W:113-117]

> Aufgabe:
>
> Wolf (2016): Kap. 5.7: 7, 9

2.3 Schleifen – Programmteile wiederholen

Die Zählschleife `for`: [W:129f]

Die kopfgesteuerte while-Schleife: [W:133ff]

Die fußgesteuerte do-while-Schleife: [W:135ff]

Kontrollierte Sprünge aus Schleifen: [W:138ff]

> Aufgabe:
>
> Wolf (2016): Kap. 6.5: 2, 3, 5, 6, 7

2.4 Funktionen erstellen

Funktionen definieren: [W:143]

Funktionen aufrufen: [W:144]

Funktionsdeklaration: [W:145f]. Funktionsdeklarationen werden üblicherweise in Headerdateien verwaltet.

Funktionsparameter: [W:147f]. Es handelt sich hierbei um einen *call by value*, d.h. die gerufene Funktion erhält eine Kopie des Wertes. Änderungen daran haben keine Rückwirkung auf die aufrufende Funktion.

Rückgabewerte von Funktionen: [W:149f]

Parameter der `main`-Funktion: [W:377-380]

Rückgabewert an das Betriebssystem: [W:157f]

Programmabbruch mit `exit`: [W:158f]

2.5 Präprozessor-Direktiven

Dateien einfügen mit `#include`: [W:169f]

Konstanten und Makros mit `#define` und `#undef`: [W:171f]

Bedingte Kompilierung: [W:180u]

Generische Auswahl: [W:186ff]

Aufgabe:

- Wolf (2016): Kap. 7.12: 1, 9, 10
- Wolf (2016): Kap. 8.7: 1
- Testen Sie das Programm `argument.c` aus A.4

3 Arbeiten mit Dateien

Datei öffnen: [W:330f]. Anmerkung: Das Schlüsselwort `restrict` wird in Kap. TODO behandelt und kann an dieser Stelle ignoriert/weggelassen werden. Falls das Öffnen der Datei nicht erfolgreich war, wird der NULL-Zeiger zurückgegeben [W:332].

Datei schließen: [W:335]

Zeilenweises Lesen: [W:341]

Das Codebeispiel in Abb. 2 zeigt, wie die genannten Funktionen zusammenspielen.

[4] Garcia (2013): S. 116.

```
① #include <stdio.h>
  int main(void)
  {
②   char dateiname[] = "Testfgets.txt";
③   char inhalt[80] = " ";
    FILE *datei;
    datei = fopen(dateiname, "r");
    if(datei != NULL)
    {
      while (feof(datei) == NULL)
④      {
          fgets(inhalt, 45, datei);
          printf("%s", inhalt);
       }
    fclose(datei);
    }

    getchar();
    return 0;
  }
```

Abbildung 2: Zeilenweises Lesen von Dateien[4]

Zeilenweise Schreiben:

- `fputs`: [W:342]
- `fprintf`: [W:351-359]

Aufgabe:

Mit den nächsten Übungen soll Schritt für Schritt eine Bücherverwaltung für eine Bücherei erstellt werden. Führen Sie folgende Schritte aus:

- Legen Sie ein neues Projekt *buecherei* an.
- Legen Sie die Dateien `medienverwaltung.h` und `medienverwaltung.c` an.
- Schreiben Sie eine Funktion `add_medium` mit folgender Signatur:

  ```
  int add_medium(const char* signatur, const char* autor, const char* titel,
    char typ, int seitenzahl, int spieldauer)
  ```

  Die Deklaration kommt in die Header- die Definition in die Quelldatei.
- Vergessen Sie nicht, die Headerdateien in den Quelldateien zu inkludieren.
- `add_medium` soll die Daten an die Datei `medien.csv` kommasepariert anhängen.
- Rufen Sie die Prozedur aus dem Hauptprogramm (mit konstanten Werten) auf.

4 Arrays und Zeichenketten

4.1 Arrays

Arrays definieren: [W:193]

Arrays mit Werten versehen und darauf zugreifen: [W:194f]

Bei Arrays wird wird nur der Zeiger auf das erste Element im System abgelegt. Soll ein Array an eine Unterfunktion übergeben werden, muss neben der Startadresse auch die Länge des Arrays übergeben werden. [W:203f]

Mehrdimensionale Arrays: [W:205f]

4.2 Zeichenketten

Lesen von Zeichenketten

In Kap. 1.2 haben Sie `scanf` zum Einlesen von Daten kennengelernt, in Kap. 3 `fgets`. Was hat nun wo seine Vorteile?

- `scanf` kann Muster erkennen und Daten in den gewünschten Zieltyp umwandeln.
- `fgets` ist robust und kommt bei Leer- oder Falscheingaben nicht gleich außer Tritt.

Es wird daher folgender Zweischritt empfohlen: Mit `fgets` werden die Daten zeilenorientiert vom Eingabemedium übernommen, dann mit `sscanf` (Lesen aus einer Zeichenkette) analysiert und verarbeitet.

Stringfunktionen

Da in den Variablen für Zeichenketten nur der Zeiger auf das erste Zeichen steht, müssen für sämtliche Operationen auf Zeichenketten wie Vergleichen, Kopieren, Suchen Bibliotheksfunktionen verwendet werden.

- `strlen`: Länge der Zeichenkette ohne Endebyte
- `strcmp`: Vergleich zweier Zeichenketten
- `strncmp`: Vergleich der ersten `n` Zeichen zweier Zeichenketten
- `strncpy`: Kopiert bis zu `n` Zeichen aus einer Zeichenkette in eine andere
- `strchr`: Finde das erste Auftreten eines Buchstabens in einer Zeichenkette.

Beispiele: [W:216ff], für `strchr` siehe folgende Zeilen:

```
// suche erstes o in "Hallo Welt"
const char* str = "Hallo Welt";
char* ptr = strchr(str, 'o');
// ptr zeigt nun auf die Zeichenkette "o Welt"
```

Aufgabe:

Wolf (2016): Kap. 9.4: 1 - 4

Erweitern Sie die Funktion `add_medium` um eine Duplikatprüfung.

- Legen Sie in `medienverwaltung.h` `int`-Konstanten für die Rückgabewerte an (`RC_OK`, `RC_FILE_NOT_FOUND`, `RC_DUPLICATE`)
- Prüfen Sie, ob die Datei `medien.csv` zum Lesen geöffnet werden konnte. Falls nein, geben Sie entsprechenden Rückgabewert zurück.
- Lesen Sie die Datei Zeile für Zeile.
- Vergleichen Sie die übergebene Signatur mit den ersten Zeichen der eingelesenen Zeile.
- Falls die Signatur schon vorhanden ist, geben Sie den entsprechenden Rückgabewert zurück.
- Werten Sie den Rückgabewert im Hauptprogramm aus.

5 Zeiger

5.1 Zeiger auf Daten

Zeiger vereinbaren:

```
int* i1;
int * i2;
int *i3;
```

Für den Compiler ist es unwichtig, wo der Stern steht. Jede der gezeigten Schreibweisen hat ihre Anhänger!

Zeiger verwenden: [W:224f]

Zugriff auf den Inhalt von Zeigern: [W:226f]

Zeigerarithmetik: [W:235], Beispiel: [W:236f]

Array und Zeiger als Funktionsparameter: [W:239]. Wird einer Funktion ein Zeiger auf einen Speicherbereich übergeben und verändert die aufgerufene Funktion diesen, sind diese Änderungen auch in der aufrufenden Funktion sichtbar (*call by reference*). Ist dies aber nicht erwünscht, so kann dies mit dem Schlüsselwort `const` unterbunden werden [W:240].

`char`-Arrays und Zeiger: [W:241f]

Arrays von Zeigern: [W:242f]

`void`-Zeiger: [W:245f]

Konstanter Zeiger: [W:247]

Zeiger für konstante Daten: [W:247f]

`restrict`-Zeiger: [W:249f]

5.2 Zeiger auf Funktionen

Zeiger Auf Funktionen ermöglichen den Austausch der Implementierung zur Laufzeit. Typische Anwendungen:

- Übergabe einer Fehlerbearbeitung an eine große Programmbibliothek
- *call back*-Funktionen zu Forschrittsanzeige

Beispiel: [W:251-254]

> Aufgabe:
>
> Wolf (2016): Kap. 10.14: 3, 6, 7, 9, 10, 11
>
> Die Duplikatsprüfung der vorigen Aufgabe meldet bei der neuanzulegenden Signatur `A1` bei einer vorhandenen Signatur `A11` einen Duplikatsfehler. Erweitern Sie die Duplikatprüfung um eine Prüfung, dass der Stringvergleich nur dann ausgeführt wird, die übergebene Signatur und die zu prüfende Signatur gleich viele Buchstaben haben. Um die Länge der zu überprüfenden Signatur zu berechnen, verwenden Sie `strchr` und die Substraktion von Zeigern.

5.3 Dynamische Speicherverwaltung

Um mit Daten variabler Größe umgehen zu können, kann in C der Freispeicher benutzt werden. Es wird vom Betriebssystem Speicher „beliebiger" Größe angefordert. Für die Rückgabe des Speichers ist aber gleichfalls der Programmierer verantwortlich. Nachdem dies aber oftmals ein schwieriges Unterfangen ist, ist es empgehlenswert dynamische Speicher nur sparsam einzusetzen.

Speicher anfordern: [W:260ff]

Speicher freigeben: [W:269]. Achtung: Ein Speicher darf nicht zweimal freigegeben werden. Daher kann es nützlich sein, den Zeiger sofort nach dem `free` mit `NULL` zu belegen. Ein Beispiel für ein Speicherloch mit fehlerhafter Freigabe: [W:270f]

> Aufgabe:
>
> Ergänzen Sie die `medienverwaltung` um eine Funktion
>
> `int get_data(const char* signatur, char*[6])`, die
>
> - die Datei `medien.csv` zeilenweise durchsucht,
> - jede Zeile überprüft, ob sie mit der übergebnen Signatur beginnt,
> - falls nein, einen `RC_NOT_FOUND` zurückgibt,

- falls ja,
  - Freispeicher derselben Länge wie die gelesenen Zeile allokiert,
  - die Zeile dort hineinkopiert.
  - Die sechs Zeiger des übergebenen Zeigerarray sollen auf die Anfänge der Datenelemente zeigen.
  - Damit jeder Teilstring richtig endet, müssen schließlich die Kommas durch `\0` ersetzt werden.
- Im Hauptprogramm soll je nach Rückgabewert ein „nicht gefunden" oder die einzelnen Datenwerte ausgegeben werden.

6 Komplexe Datentypen

6.1 Strukturen und Unionen

Strukturtyp deklarieren: [W:276]

Definition einer Strukturvariablen: [W:277]

Deklaration und Definition: [W:278]

Synonyme für Strukturtypen erstellen: [W:279]

Zugriff über Punktoperator: [W:280]

Strukturen initialisieren: [W:283]

Zugriff über Pfeiloperator: [W:287]

Unionen: [W:302f]

6.2 Der Aufzählungstyp enum

Beispiel: [W:305]

Aufgabe:

Ergänzen Sie die `medienverwaltung` um eine Funktion

`int get_struct(const char* signatur, medium_t*).`

- Definieren Sie in `medienverwaltung.h` einen Typen `typ_t` für das `enum` bestehend aus `Buch` und `CD`.
- Definieren Sie `medium_t` als Typ für folgende Struktur:

```
struct medium{
  char signatur[4];
  char* autor;
  char* titel;
  typ_t typ;
  int seitenzahl;
  int spieldauer;
}
```

- Rufen Sie in der Funktion `get_data` auf und versorgen Sie die Werte der Struktur.
- Verwenden Sie `get_struct` im Hauptprogramm.

Aufgabe:

Alle bisherigen Beispiele, die ein `malloc` enthalten, kümmern sich nicht um die Wiederfreigabe des Speichers. So wie die Funktionen konzipiert sind, ist dies auch schwer. Daher sollen im Folgenden zwei Beispiele gezeigt werden, wie dies in den Griff bekommen werden kann.

Variante A): Aufrufer stellt Puffer zur Verfügung

Schreiben Sie eine Funktion

`int get_data1(const char* signatur, char*[6], char[] buffer, int length),`

die wie `get_data` funnktioniert. Anstatt des Allokierens von Speicher soll geprüft werden, ob die zu kopierende Zeile in den Puffer passt.

- Falls ja, wird die Zeile hineinkopiert und die Zeiger des Arrays passend belegt.
- Falls nein, wird ein `RC_BUFFER_TOO_SMALL` zurückgegeben.

Testen Sie die Funktion auch mit zu kleinem Puffer!

Variante B): Ergebnis liegt in einem zusammenhängenden Freispeicherbereich.

Für die Freigabe ist jetzt der Aufrufer zuständig. Schreiben Sie eine Funktion

`medium_t* create_struct(const char* signatur, int* rc)`. Diese soll

- Einen großen Puffer auf dem Stapel allokieren und `get_data1` aufrufen.
- Für das Ergebnis einen Freispeicher allokieren, der exakt Platz für die Ergebnisstruktur, sowie die Zeichenketten `autor` und `Titel` bietet. Vergessen Sie dabei nicht die Endebytes der Zeichenketten.
- Versorgen Sie das Ergebnis mit den richtigen Daten.
- Vergessen Sie nicht den `int`-Rückgabewert zu versorgen. Im Fehlerfall soll der `NULL`-Zeiger zurückgegeben werden.
- Verwenden Sie diese Funktion im Hauptprogramm.
- Geben Sie im Hauptprogramm den Freispeicher nach der Ausgabe der gefundenen Daten wieder frei.

7 Quellen

Garcia, Ricardo Hernandez	ANSI C. Grundlagen der Programmierung. Herdt 2013, 3. Ausgabe
Wolf, Jürgen	Grundkurs C, Rheinwerk 2016, 2. Auflage