# Systemprogrammierung - AIN/2

## Sommersemester 2022

---

## Übungsaufgabe 1: C Grundtypen

### Abgabe bis 7./8.4.2022

### Programmierung

Auf den Seiten 2-15 bis 2-19 der Vorlesungsunterlagen sind 15 Grundtypen aufgezählt.

Schreiben Sie ein C-Programm `aufgabe1.c`, das zu jedem Grundtyp je eine lokale Variable mit Initialisierung enthält. Zusätzlich soll das Programm eine Zeigerdefinition `char* s = "Hallo";` enthalten. Sie müssen also insgesamt 16 Variablen definieren.

Das Programm soll die Adresse, den Platzbedarf, den Typ, den Namen und den Wert dieser 16 Variablen ausgeben. Pro Variable soll genau eine Zeile mit den genannten fünf Spalten ausgegeben werden, immer mit der Variablenadresse beginnend.

Abschließend soll das Programm ausgeben, ob es mit dem C-Datenmodell `"ILP32"`, `"LP64"` oder `"LLP64"` arbeitet, oder ob keines der drei Modelle vorliegt:

- man spricht von einem **`ILP32`**-Datenmodell, wenn die Typen `int` (**`I`**), `long int` (**`L`**) und `char*` (**`P`**) je 32 Bit belegen
- ein **`LP64`**-Datenmodell liegt vor, wenn `int` 32 Bit belegt und sowohl `long int` (**`L`**) als auch `char*` (**`P`**) je 64 Bit belegen
- ein **`LLP64`**-Datenmodell liegt vor, wenn `int` sowie `long int` je 32 Bit belegen und sowohl `long long int` (**`LL`**) als auch `char*` (**`P`**) je 64 Bit belegen

*Hinweis:*
*Achten Sie bei den Werten auf die richtigen von C unterstützen `printf`-Formatangabe. Bei den Formatangaben für Zahlen brauchen Sie eine Kombination aus "length modifier" und "conversion specifier". Ein übersichtliche Darstellung dazu finden Sie unter* ***https://en.cppreference.com/w/c/io/fprintf****. Andere Quellen sind das Linux Manual (Kommando `man 3 printf` ) oder einschlägige Internetseiten wie* ***https://man7.org/linux/man-pages/index.html*** *(siehe auch die Literaturseite der Vorlesung).*

### Test und Qualitätssicherung

Speichern Sie die Datei ↴ **Makefile** in Ihr Arbeitverzeichnis der Aufgabe 1. Übersetzen und binden Sie dann Ihr Programm mit dem Befehl
*Diesen und alle weiteren Befehle abtippen, nicht mit der Maus kopieren, wenn Sie etwas lernen wollen!*

```
make aufgabe1
```

*Das Programm `make` automatisiert mittels der Beschreibungen in `Makefile` das Übersetzen und Binden Ihres Programms.*
*Die Entsprechung dazu bei den Java-Übungen in Programmiertechnik 1 war das Automatisierungswerkzeug `ant` mit `build.xml`.*

Führen Sie Ihr Programm aus und lassen Sie die Ausgaben auch mal vom `sort`-Kommando aufsteigend oder absteigend sortieren:

```
./aufgabe1
./aufgabe1 | sort
./aufgabe1 | sort -r
```

Betrachten Sie die Ausgaben und beantworten Sie folgende Fragen:

- Sind die Variablen in der Reihenfolge ihrer Definition im Hauptspeicher abgelegt?
- Liegen die Variablen direkt hintereinander oder gibt es Lücken? Falls es Lücken gibt, wo liegen sie? Können Sie sich erklären, warum es eventuell Lücken gibt?
- Wie viel Speicher belegen die Variablen insgesamt, also inklusive eventueller Lücken?

Prüfen Sie, ob das Analysewerkzeug `cppcheck` Probleme in Ihrem Quellcode meldet. Versuchen Sie, wenn möglich, die Probleme zu beheben. Welche Problemen lassen sich nicht beheben?

```
cppcheck --enable=warning,style --std=c11 aufgabe1.c
```

Sie können `cppcheck` auch vereinfacht über `make` aufrufen:

```
make cppcheck
```

*Die Entsprechung zu `cppcheck` bei den Java-Übungen in Programmiertechnik 1 war das Analysewerkzeug `spotbugs`.*

**Protokoll**

Erstellen Sie ein Protokoll Ihres Test-Schritts. Öffnen Sie dazu eine neue Konsole oder setzen Sie Ihre aktuelle Konsole zurück (Konsolenmenü `Terminal -> Zurücksetzen und leeren`).

Rufen Sie in der neuen bzw. zurückgesetzten Konsole `make clean` auf und wiederholen Sie alle Kommandos aus dem Test-Schritt.

Markieren Sie die Konsolenausgabe abschließend per Menüpunkt `Bearbeiten -> Alles markieren`. Öffnen Sie einen Texteditor und kopieren Sie die markierte Konsolenausgabe durch drücken der mittleren Maustaste (des Rads) in das Editorfenster. Speichern Sie mit dem Editor die kopierte Konsolenausgabe als Datei `protokoll-aufgabe1.txt` im gleichen Verzeichnis wie Ihr Programm.

Ergänzen Sie mit dem Texteditor in `protokoll-aufgabe1.txt` Ihre Antworten auf die Fragen aus dem Test-Schritt.

### Abgabe

Führen Sie Ihr Programm und Ihre Protokolldatei vor.

*Hinweis:*
*Der Compiler `gcc` darf für Ihr Programm keine Fehler oder Warnungen mehr ausgeben.*
*Ihr Programm muss außerdem ordentlich formatierte sein. Bessern Sie die Formatierung gegebenenfalls mit `astyle` nach:*

```
astyle -p -H --style=ansi aufgabe1.c
```

### Freiwillige Zusatzaufgabe (pro Spiegelpunkt 1 Bonuspunkt)

- Erstellen Sie eine Version `aufgabe1-int.c` Ihres Programms, die Variablen mit den in C99 eingeführten 28 ganzzahligen Typen fester Länge `int8_t`, `int16_t` usw. definiert. Initialisieren Sie die Variablen jeweils mit der größten darstellbaren Zahl`INT8_MAX`, `INT16_MAX` usw.. Das Programm soll wie im Pflichtteil jeweils Adresse, Platzbedarf, Typ, Name und Wert der 28 Variablen ausgeben. Verwenden Sie für die Ausgabe des Werts die ebenfalls in C99 eingeführten Formatangaben `PRId8`, `PRId16` usw..

  *Hinweis: die Typen mit den zugehörigen symbolischen Konstanten und Formatangaben finden Sie unter* ***https://en.cppreference.com/w/c/types/integer***

  Übersetzen, binden, prüfen und testen Sie Ihr Programm mit den folgenden Aufrufen:

  ```
  make aufgabe1-int
  cppcheck --enable=warning,style --std=c11 aufgabe1-int.c
  ./aufgabe1-int
  ```

- Erstellen Sie eine Version `aufgabe1-ansi.c` Ihres Programms, die sich mit dem Sprachstandard C89 (auch ANSI-C genannt) begnügt. ANSI-C war auch nach der Veröffentlichung von C99 in der Praxis in weitem Gebrauch.

  Variablendefinitionen und Anweisungen dürfen in C89 nicht gemischt werden. Definieren Sie deshalb alle Variablen am Anfang von `main` gleich nach der öffnenden geschweiften Klammer.

  Die ganzzahlige Datentypen `long long int` und `unsigned long long int` sowie den Datentyp `bool` bzw. `_Bool` gab es in C89 noch nicht.

  Es gab in C89 auch noch nicht für alle Zahltypen passende `printf`-Formatangaben. Verwenden Sie deshalb für `sizeof`-Werte `%lu` und fügen Sie vor dem zugehörigen Argument eine Typanpassung `(unsigned long)` ein. Ein-Byte-Zahlen geben Sie einfach mit `%d` bzw. `%u` aus. Die Typanpassung des zugehörigen Arguments funktioniert hier automatisch.

  Übersetzen, binden, prüfen und testen Sie Ihr Programm mit den folgenden Aufrufen:

```
make "CFLAGS=-g -W -Wall -ansi -pedantic" aufgabe1-ansi
cppcheck --enable=warning,style --std=c89 aufgabe1-ansi.c
./aufgabe1-ansi
```

---

***Prof. Dr. H. Drachenfels***
***Hochschule Konstanz*** - ***Impressum*** - ***Datenschutzerklärung***

*Letzte Änderung: 23.2.2022*