Speicherklassen

Einführung in die Programmierung

Michael Felderer

Institut für Informatik, Universität Innsbruck

Gültigkeitsbereich etc.

- Speicherobjekte (Variablen oder Funktionen) haben einen Gültigkeitsbereich, eine Lebensdauer, eine Bindung und einen Speicherort.
- Mit bestimmten Schlüsselwörtern können diese Eigenschaften einzelner Bezeichner modifiziert werden.

Adressraum eines Programms

- Codesegment
  - Maschinencode des Programms
- Datensegment
  - Globale (externe) Variablen
- Stack
  - Lokale Variablen
  - Rücksprungadresse
  - Parameter einer Funktion
- Heap
  - Dynamische Variablen
  - Wird noch ausführlich im Foliensatz „Dynamische Speicherverwaltung“ besprochen.

Gültigkeitsbereiche für Speicherobjekte

- Anweisungsblock (block scope)
  - Wird ein Speicherobjekt (z.B. Variable) innerhalb eines Anweisungsblocks definiert, dann kann auf dieses Speicherobjekt innerhalb des Anweisungsblocks (beginnend mit der Definition) zugegriffen werden.
  - Beispiele: Lokale Variablen, formale Parameter
- Funktionsprototyp (function prototype scope)
  - Variablen, die innerhalb eines Funktionsprototyps angegeben werden, haben einen Geltungsbereich bis zum Ende des Funktionsprototyps.
  - Beispiel: Bei Verwendung von VLAs
- Datei bzw. Modul (file scope)
  - Speicherobjekte, die außerhalb von Funktionen deklariert werden, können ab dem Zeitpunkt der Definition bis zum Dateiende angesprochen werden.
  - Beispiel: Globale Variablen
- Funktion (function scope)
  - Labels (bei goto) sind innerhalb einer Funktion überall sichtbar.

Lebensdauer

- Automatische Lebensdauer
  - Klasse von Speicherobjekten, die zur Ausführungszeit einer Funktion (eines Blocks) definiert sind.
  - Beim Aufruf einer Funktion wird ein sogenannter Stack-Frame mit solchen Speicherobjekten angelegt, beim Verlassen der Funktion wird dieser Stack-Frame wieder verworfen.
    - Der Inhalt dieser Speicherobjekte ist nach dem Verlassen der Funktion verloren!
- Statische Lebensdauer
  - Dazu gehören Speicherobjekte, die zur gesamten Programmlaufzeit einen festen Platz und somit immer einen gültigen Wert im Speicher haben.
  - Statische Speicherobjekte werden nicht in einem Stack-Frame sondern im Datensegment des Programms gespeichert.

Speicherort und Bindung

- Speicherort
  - Ein Speicherobjekt kann im Speicher oder in einem Prozessorregister gehalten werden.
  - Beim Speicher wird nochmals zwischen Datensegment und Stack unterschieden.
- Bindung
  - Extern
    - Ein Speicherobjekt kann überall in einem Programm, das aus mehreren Dateien besteht, verwendet werden.
  - Intern
    - Ein Speicherobjekt kann in einem Programm, das aus mehreren Dateien besteht, nur in der Datei verwendet werden, in der es definiert wird.
  - Keine
    - Lokale Variablen

auto

- Dieser Spezifizierer kann nur für lokale Variablen verwendet werden.
- Dieses Schlüsselwort ist meist überflüssig, da lokale Variablen innerhalb von Blöcken **standardmäßig** mit diesem Spezifizierer versehen werden.
  - Beispiel: `auto int i = 2;`
- Variablen sind nur in diesem Block verwendbar.
  - Verstecken Variablen mit gleichen Bezeichnern in einem umschließenden Block.
  - Beispiele: Bisherige Beispiele (+ C99-Features!)
- Automatische Variablen müssen explizit initialisiert werden.

register

- Dieses Schlüsselwort wird bei Variablen verwendet.
- Damit wird der Compiler angewiesen, diese Variable möglichst lange in einem Prozessorregister zu halten.
  - Register arbeiten schneller als der Arbeitsspeicher.
  - Beispiel: `register int i = 0;`
- Auswirkung hängt aber vom Compiler ab!
  - Compiler kann eine `register`-Variable auch wie eine gewöhnliche `auto`-Variable behandeln.
  - Dann hat diese Variable auch die gleichen Eigenschaften wie eine `auto`-Variable.
- Auf eine `register`-Variable kann man nicht mit dem Adressoperator zugreifen.
  - Wird möglicherweise in einem Register gespeichert.
  - Register kann nicht adressiert werden.

static

- Innerhalb einer Funktion kann dieses Schlüsselwort für eine Variable verwendet werden.
  - Damit bleibt der Wert der lokalen Variablen nach der Rückkehr aus dieser Funktion erhalten.
  - Die Variable kann aber nur lokal innerhalb der Funktion angesprochen werden.
- Dieses Schlüsselwort kann auch außerhalb von Funktionen vor eine Variable oder eine Funktion gestellt werden.
  - Die Sichtbarkeit wird damit auf die aktuelle Übersetzungseinheit (Datei) beschränkt.
  - Es wird kein Linkersymbol erzeugt.
  - Somit können zum Beispiel Funktionen mit denselben Bezeichnern in verschiedenen Quelldateien verwendet werden.

Beispiel (Variablen statisch)

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

int a = 3;           // Kann in anderen Dateien verwendet werden
static int b = 2;    // Kann nur in dieser Datei verwendet werden

void test(void) {
    static int c = 7;
    int d = 0;
    printf("%d %d %d %d\n", a++, b++, c++, d++);
}

int main(void) {
    test();
    test();
    return EXIT_SUCCESS;
}
```

Ausgabe:

```
3 2 7 0
4 3 8 0
```

extern (1)

- Mit diesem Schlüsselwort kann ein Bezug zu einem Speicherobjekt, das an einer anderen Stelle (in einer anderen Datei) definiert wurde, hergestellt werden.
  - Der Compiler gibt dann dem Linker Bescheid, dass er die Verweise dazu in einer anderen Übersetzungseinheit und/oder Bibliothek auflösen muss.
- `extern` bei Variablen
  - Bedeutet, dass sich der Bezeichner auf eine Variable bezieht, die in einer anderen Datei definiert wurde.
  - Es wird also keine neue Variable definiert, sondern eine bereits definierte Variable wird deklariert.
  - Eine externe Variable kann nur in einer einzigen Datei definiert werden.
    - In den anderen Dateien wird sie nur mit Hilfe der extern-Deklaration referenziert.
    - Die Definition legt die Adresse einer externen Variable fest.
    - Der Linker setzt in den anderen Dateien, die über die extern-Deklaration diese Variable referenzieren, die Adresse dieser Variable ein.

extern (2)

- `extern` bei Funktionen
  - Bedeutet, dass die Funktion global im Programm (in allen Dateien) sichtbar ist.
  - Das ist die Voreinstellung und braucht daher nicht jedes Mal angegeben werden.

Beispiel für extern (2 Dateien – gemeinsam übersetzt)

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

extern void f2(void);
static void f1(void);
extern int zahl;

int main(void) {
	printf("Hier ist main, zahl = %d\n", zahl);
	f1();
	f2();
	return EXIT_SUCCESS;
}

int zahl = 6;

static void f1(void){
	printf("Hier ist f1, zahl = %d\n", zahl);
}
```

datei1.c

```
Ausgabe:
Hier ist main, zahl = 6
Hier ist f1, zahl = 6
Hier ist f2, zahl = 6
```

```c
#include <stdio.h>

extern int zahl;

void f2(void) {
	printf("Hier ist f2, zahl = %d\n", zahl);
}
```

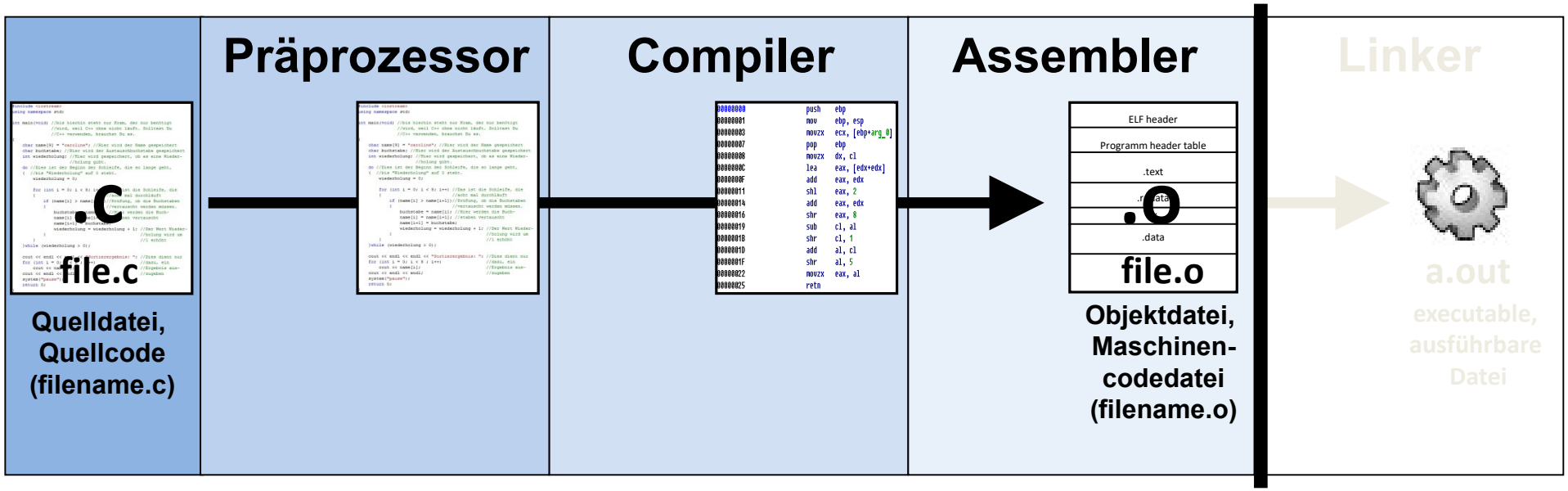
datei2.c

gcc-Aufrufe

- Übersetzung mit einem Aufruf
  - `datei1.c`, `datei2.c` – in einem Unterverzeichnis
  - `gcc -Wall -Werror -std=c99 -o test datei1.c datei2.c`
  - Erzeugt ausführbare Datei `test`
- Übersetzung mit getrennten Aufrufen
  - `datei1.c`, `datei2.c` – in einem Unterverzeichnis
  - `gcc -Wall -Werror -std=c99 -c datei1.c` *(erzeugt datei1.o)*
  - `gcc -Wall -Werror -std=c99 -c datei2.c` *(erzeugt datei2.o)*
  - `gcc -o test datei1.o datei2.o`
  - Erzeugt auch eine ausführbare Datei `test`.
  - Zwei Object-Dateien werden erzeugt (gcc mit Option -c aufrufen)!
  - Dritter gcc-Aufruf verbindet die beiden Object-Dateien zu einer ausführbaren Datei.

gcc – Wo wird die Übersetzung unterbrochen?

- Das Quellprogramm durchläuft den Präprozessor, den Compiler und den Assembler, aber der Linker wird nicht ausgeführt.
- Standardmäßig wird der Dateiname durch Ersetzen des Suffix .c mit .o erzeugt.



Zusammenfassung – Speicherklassen für Variablen

Speicherklasse	Lebensdauer	Gültigkeitsbereich	Bindung	Segment	Initialisierung	Vereinbarung
`automatic`	automatisch	Block	Keine	Stack	Nein	In einem Block
`register`	automatisch	Block	Keine	Stack oder Register	Nein	In einem Block, mit `register`
`static` (ohne Bindung)	statisch	Block	Keine	Daten	Ja	In einem Block, mit `static`
`static` (interne Bindung)	statisch	Datei	Intern	Daten	Ja	Außerhalb von Funktionen, mit `static`
`static` (externe Bindung)	statisch	Datei	Extern	Daten	Keine notwendig (da extern)	Außerhalb von Funktionen