



Prozedurale Programmierung

Speicherklassen

Hochschule Rosenheim - University of Applied Sciences

WS 2018/19

Prof. Dr. F.J. Schmitt



Speicherklassen

- Variablen haben einen gewissen **Gültigkeitszeitraum** und einen gewissen **Sichtbarkeitsbereich**
- Diese Eigenschaften werden mit **Speicherklassen** festgelegt
- Verschiedene Arten von Speicherklassen existieren
- Prinzipielle Unterscheidung zwischen
 - ⊞ lokalen Variablen
 - ⊞ globalen Variablen



Variablen

➤ Unterscheidung

⊞ Lokale Variablen

- ⊞ Sind nur innerhalb einer Funktion bekannt
- ⊞ häufig verwendet

⊞ Modulglobale Variablen

- ⊞ sind innerhalb eines Moduls bekannt

⊞ Globale Variablen

- ⊞ liegen außerhalb jeglicher Funktion
- ⊞ können von allen Ecken und Enden des Programms verändert werden
- ⊞ vorsichtige und seltene Verwendung – nur aus gutem Grund!



Lokale Variablen (1)

- Lokalität beschränkt sich dabei auf die Funktionen, in denen sie definiert werden

- Lokale Variablen sind **innerhalb einer Funktion gekapselt**
 - ⊞ sind nur lokal in der Funktion sichtbar und
 - ⊞ können nur in der Funktion modifiziert werden



Lokale Variablen (2)

- Lokale Variablen müssen zu **Beginn** einer Funktion definiert werden

```
long Fakultaet(long a)
{
    long f = 1, i;

    for(i = a; i > 1; i--)
        f = f * i;

    return f;
}
```

- ⊞ Variable `a`, `f` und `i` sind lokale Variablen der Funktion `Fakultaet`



Aufgabe

- Geben Sie an, welche Werte die lokalen Variablen nach Aufruf von **Fakultaet(4)** bis zum Ende der Funktion annehmen
- Verwenden Sie hierfür eine Tabelle der Form:

Schritt	a	f	i
1			
2			
3			
4			
5			
6			
...			

```
long Fakultaet(long a)
{
    long f = 1, i;

    for(i = a; i > 1; i--)
        f = f * i;

    return f;
}
```



Blockstruktur von C (1)

➤ Nachteil von lokalen Variablen:

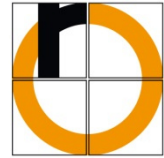
- ✚ Variablen werden erzeugt, auch wenn sich im Verlauf der Funktion herausstellt, dass sie nicht verwendet werden

```
long Fakultaet (long a)
{
    long f = 1, i;

    if(a < 0)
    {
        printf("Fehler!");
        return 0;
    }
    for(i = a; i > 1; i--)
        f = f * i;

    return f;
}
```

Variablen `f`, `i` werden erzeugt, obwohl sie für negative `a` nicht verwendet wird!



Blockstruktur von C (2)

- Einführung von **Blöcken**
- Block ist ein Programmbereich, der in geschwungenen Klammern eingefasst ist
 - ⊞ Funktionsrumpf
 - ⊞ Schleifenrumpf
 - ⊞ Sonstige Programmbereiche



Blockstruktur von C (3)

```
long Fakultaet (long a)
{
    if(a < 0)
    {
        printf("Fehler!");
        return 0;
    }

    {
        long f = 1, i;

        for(i = a; i > 1; i--)
            f = f * i;

        return f;
    }
}
```

Das ist schlechter
Programmierstil!

Variablendefinitionen müssen
zu Beginn eines Blocks stehen!

Variablen `f`, `i` gelten lokal
innerhalb des Blocks und
werden bei Verlassen des
Blocks zerstört



Das ist schlechter
Programmierstil!

Blockstruktur von C (4)

```
int main(void)
{
    long a = 1;
    {
        long a = 2;
        printf("a = %ld\n", a);
    }

    {
        printf("a = %ld\n", a);
    }
}
```

Nicht empfohlen:

Variablen mit gleichen Namen
verdecken Variablen aus
äußeren Blöcken

Blöcke übernehmen die
Variablen von Blöcken in denen
sie stehen

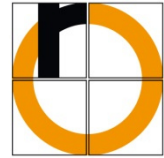
Ausgabe: erst 2 - dann 1

Fazit: Vermeiden Sie gleichlautende Variablen in
inneren Blöcken



Blockstruktur – Anmerkungen

- Blöcke sollten nur verwendet werden, wenn das durch die Kontrollstruktur sowieso gemacht werden muss (z.B. if, while, for, ...)
 - die Einführung von Blöcken, nur um dort lokale Variablen anlegen zu können, sollte vermieden werden – den Nachteil, dass eine angelegte Variable evtl. nicht verwendet wird, nimmt man in Kauf
- besser nicht so machen, wie in den Beispielen auf den vorherigen beiden Folien



Lokale Variablen – Speicherklasse `auto`

- **Automatische** Variablen werden zu Blockanfang automatisch für den Block angelegt und zu Blockende wieder gelöscht.
- Alle lokalen Variablen, denen kein spezielles Schlüsselwort, wie `register` oder `static` vorangestellt ist, sind automatische Variablen



Lokale Variablen – Speicherklasse `register`

- Variablen der Speicherklasse `register` werden nicht im Speicher angelegt, sondern in einem Register des Prozessors gespeichert
- es sind nur Variablen der Typen `char`, `int` (`long`, `short`), `float`, `double` sowie Zeiger erlaubt
- Beispiel:

```
double Mittelwert(register double x, register double y)
{
    return (x + y) / 2;
}
```

Speicherklasse `register`

Anmerkungen



- geht nur für lokale Variablen
- sollte nur verwendet werden, wenn schneller Zugriff unbedingt erforderlich ist (z.B. für Zähler)
- Adressoperator (&) kann nicht verwendet werden (die Variable hat ja keine Speicheradresse)
- das Schlüsselwort `register` ist nur ein Hinweis für den Compiler, dass die Variable in ein Register sollte – sie wird nicht notwendigerweise dort landen



Lokale Variablen – Speicherklasse `static` (1)

- Lokale statische Variablen werden analog zu automatischen Variablen definiert und verwendet
- bei Definition ist jedoch Schlüsselwort `static` vor dem Datentyp anzuschreiben
- Lokale statische Variablen existieren solange das Programm läuft
- der Sichtbarkeitsbereich ist nur lokal
 - ⊞ Variable wird beim Einsprung in die Funktion sichtbar und
 - ⊞ beim Verlassen der Funktion unsichtbar



Lokale Variablen – Speicherklasse `static` (2)

```
void AufrufZaehler(void)
{
    static long aufrufe = 0;

    aufrufe = aufrufe + 1;
    printf("Die Funktion wurde %ld mal aufgerufen\n", aufrufe);
}
```

- Bei jedem Funktionsaufruf wird die Variable `aufrufe` sichtbar
- Wert wird um 1 erhöht und ausgegeben
- Beim Verlassen der Funktion wird die Variable `aufrufe` wieder unsichtbar
- Initialisierung erfolgt nur beim ersten Aufruf der Funktion



Globale Variablen

- Globale Variablen werden **außerhalb jeder Funktion** definiert
- Gültigkeitszeitraum erstreckt sich über die **gesamte** Programmlaufzeit
- Nachteil: können aus **verschiedenen Funktionen** heraus **modifiziert** werden → unüberschaubare Seiteneffekte
- Fazit: **Vermeidung** von globalen Variablen!



Globale Variablen – Speicherklasse `extern`

```
// Modul file1.c  
long oeffentlicheZahl;  
//...
```

Definition der externen, globalen Variablen `oeffentlicheZahl`

```
//Modul file2.c  
extern long oeffentlicheZahl;  
//...
```

Deklaration der externen, globalen Variablen `oeffentlicheZahl`

- Schlüsselwort `extern` wird bei der Definition nicht angegeben
- Sind innerhalb der Datei, in der sie definiert wurden, sichtbar
- Können auch exportiert werden (Namen und Datentyp in anderen Dateien bekannt machen)



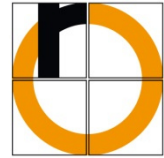
Definition/Deklaration

➤ Definition:

- ⊞ Reservierung von Speicher
- ⊞ letztendlich muss jede Variable irgendwo definiert sein

➤ Deklaration

- ⊞ Festlegen der Art der Variablen
- ⊞ Bekanntmachen von Name und Typ – **keine** Speicherreservierung
- ⊞ **Bekanntmachen** der Datenobjekte, die in anderen Übersetzungseinheiten definiert werden bzw. in derselben Übersetzungseinheit erst nach ihrer Verwendung definiert werden
- ⊞ nur **externe globale Variablen** können deklariert werden



Globale Variablen – Speicherklasse `static`

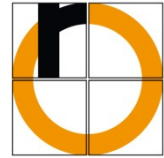
- soll eine globale Variable nicht exportiert werden, so ist das Schlüsselwort `static` anzugeben
- Gültigkeitszeitraum: globale statische Variable existiert solange das Programm läuft
- Sichtbarkeit: nur innerhalb eines Moduls ab der Position ihrer Definition
- Beispiel:

```
static long statischeVariable;
```



Globale Variablen

- Welche Konsequenz ergibt sich aus dem Exportieren einer globalen Variablen?
 - ⊞ Variable kann dann ohne Kontrolle auch in anderen Modulen modifiziert werden
- Bei globalen Variablen sind statische Variablen gegenüber externen zu bevorzugen!
- Für Zugriff in anderen Modulen: besser Verwendung einer Schnittstellen-Funktion



Zur Verwendung von static

➤ static wird in verschiedenen Zusammenhängen verwendet

⊞ Variablen

- ⊞ lokal: Sichtbarkeit lokal, Inhalt bleibt erhalten
- ⊞ global: nur im definierten Modul gültig/sichtbar, dort aber global

⊞ Funktionen

- ⊞ Funktion nur im definierten Modul gültig/sichtbar



Zusammenfassung

➤ Lokale Variablen

- ⊞ müssen zu Beginn eines Blocks definiert werden
- ⊞ Speicherklassen
 - ⊞ auto (Default, kann weggelassen werden)
 - ⊞ register
 - ⊞ static

➤ Globale Variablen

- ⊞ Export in andere Module: extern
 - ⊞ modulglobal: static
- bei gleichem Namen verdeckt die lokalste Variable alle anderen Variablen gleichen Namens