SAARLAND UNIVERSITY
COMPUTER SCIENCE

C/C++ Programmierung

Grundlagen: Anweisungen

Sebastian Hack
Christoph Mallon

(hack|mallon)@cs.uni-sb.de

Fachbereich Informatik
Universität des Saarlandes

Wintersemester 2009/2010

Anweisungen (Statements)

§6.8

SAARLAND UNIVERSITY
COMPUTER SCIENCE

```
statement:
  labeled-statement
  compound-statement
  expression-statement
  selection-statement
  iteration-statement
  jump-statement
```

Ausdrucksansweisung

§6.8.3

```
expression-statement:
  expression? ;
```

- Beliebiger Ausdruck gefolgt von Semikolon
- Ohne den optionalen Ausdruck: Nullanweisung
  - Wird benötigt, wenn Grammatik eine Anweisung verlangt, aber man dort nichts ausführen möchte
  - Falle: Kann auch zu schwer auffindbaren Fehlern führen; besser `{}` verwenden (gleich)

```
x = x + 1;
++x;
f(x);
42;             // Huch?

if (x < 0); // Autsch!
  x = 0;
```

if-Anweisung

§6.8.4.1


```
selection-statement:
  if ( expression ) statement-1
  if ( expression ) statement-1 else statement-2
```

- Wenn `expression != 0`, dann führe `statement-1` aus
- Ansonsten führe `statement-2` aus (falls vorhanden)
- Falle: `if (defcon = 1) launch_nuclear_missiles();` weist `1` an `defcon` zu beginnt einen Atomkrieg, da `1 != 0`; kein Vergleich von `defcon` mit `1`
- Achtung: `else` gehört zum lexikalisch nächstmöglichen `if`

Baumelndes else (Dangling else)

SAARLAND UNIVERSITY
COMPUTER SCIENCE

```
if (a)
  if (b)
    printf("a␣und␣b\n");
else
  printf("a␣und␣nicht␣b!\n");
```

- Falle: `else` gehört zum zweiten `if`, auch wenn die Einrückung anderes suggeriert

Blockanweisung

§6.8.2

```
compound-statement:
  { block-item* }

block-item:
  declaration
  statement
```

- Gruppiert mehrere Anweisungen zu einer Anweisung
- Anweisungen werden der Reihe nach ausgeführt
- Unterscheidung zwischen Deklaration und Anweisung:
  So ist `if (expr) int i;` nicht erlaubt (jedoch in C++)
  `if (expr) { int i; }` ist erlaubt
- Vor C99: `{ declaration* statement* }`

switch-Anweisung

§6.8.4.2

```
selection-statement:
  switch ( expression ) statement

labeled-statement:
  case constant-expression : statement
  default : statement
```

- Berechne Wert von `expression`
- Springe zu entsprechender `case`-Marke
- Werte der `case`-Marken müssen ganzzahlige Konstanten sein
- Ansonsten springe zu `default`-Marke (falls vorhanden)
- Verlassen des switch durch `break`-Anweisung
  Achtung: Ansonsten durchfallen zum nächsten Fall!
- Ist flexibler als in anderen Sprachen, man kann damit wilde Dinge treiben, z.B. Duff's Device

Beispiel: switch

SAARLAND UNIVERSITY
COMPUTER SCIENCE

```
switch (x) {
  case 0:
    printf("keiner\n");
    break;
  case 1:
    printf("einer\n");
    break;
  case 2:
    printf("zwei\n");
    break;
  default:
    printf("viele\n");
    break;
}

switch (x) { // dasselbe nochmal, aber kompakter
  default: printf("viele\n");  break; // Position egal
  case 0:  printf("keiner\n"); break;
  case 1:  printf("einer\n");  break;
  case 2:  printf("zwei\n");   break;
}
```

Schleifen: while, do-while

§6.8.5.1+2

```
iteration-statement:
  while ( expression ) statement
  do statement while ( expression ) ;
```

- Solange `expression != 0` führe `statement` aus
- Bei `do-while` wird `statement` zuerst mindestens einmal ausgeführt
- Falle: Vergessenes `do` ist wieder syntaktisch gültiges Programm

```
do
{
  something(i);
}
while (i *= 2, --n != 0);

// oder in einer Zeile:
do something(i); while (i *= 2, --n != 0);
```

Schleifen: for

§6.8.5.3

```
iteration-statement:
  for ( expression-1? ; expression-2? ; expression-3? )
    statement
  for ( declaration expression-2? ; expression-3? )
    statement
```

- Werte zuerst einmal `expression-1` aus (falls vorhanden)
- Solange `expression-2 != 0` (oder nicht vorhanden), führe `statement` aus
- Außer beim ersten Durchlauf wird `expression-3` vor `expression-2` ausgewertet.
- Zweite Variante ab C99 und C++
- So deklarierte Variablen nur innerhalb der Schleife gültig

```
// Typischer Anwendungsfall
for (int i = 0; i != 10; ++i) { /* ... */ }
```

Sprunganweisungen: break, continue

§6.8.6.2+3

```
jump-statement:
  continue ;
  break ;
```

- Beziehen sich ausschließlich auf direkt umgebende Schleife oder `switch`
- `break` verlässt diese sofort
- `continue` fährt mit nächster Schleifenitertion fort
  - `while` und `do`: Sprung zur Bedingung
  - `for`: Sprung zu `expression-3`, dann Bedingung

```
for (int i = 0; i != n; ++i)
{
  if (!good(i)) continue;
  if (!nice(i)) continue;
  if (bad(i))   break;
  return i;
}
return -1;
```

Sprunganweisungen: while statt for[1]

The `for` statement has the form

```
for ( expression-1? ; expression-2? ; expression-3? )
  statement
```

This statement is equivalent to

```
expression-1 ;
while (expression-3) {
  statement
  expression-3 ;
}
```

[1] Aus „The C Programming Language“, Abschnitt 9.6

Sprunganweisungen: while statt for[1]

SAARLAND UNIVERSITY
COMPUTER SCIENCE

The `for` statement has the form

```
for ( expression-1? ; expression-2? ; expression-3? )
  statement
```

This statement is equivalent to

```
expression-1 ;
while (expression-3) {
  statement
  expression-3 ;
}
```

- Falsch: `continue` als `statement`
  $\longrightarrow$ `expression-3` wird nicht ausgeführt!

[1] Aus „The C Programming Language“, Abschnitt 9.6

Sprunganweisungen: return

§6.8.6.4

```
jump-statement:
  return expression? ;
```

- Verlässt die umschließenden Funktion sofort
- Keine `expression` bei `void`-Funktion
- Sonst `expression` mit zum Rückgabetyp der Funktion passenden Typ
- Kann an beliebigen Stellen und beliebig oft in einer Funktion verwendet werden
- Falle: C erzwingt nicht das Vorhandensein, dann ist der Rückgabewert undefiniert (Übersetzer warnen heutzutage)

Sprunganweisungen: goto

§6.8.6.1

```
jump-statement:
  goto identifier ;

labeled-statement:
  identifer : statement
```

- Führt Ausführung bei angegebener Sprungmarke fort
- Nur Sprünge innerhalb der umschließenden Funktion möglich
- Sprungmarke muss in umschließender Funktion eindeutig sein
- Damit kann mehrstufiges `break`/`continue` realisiert werden
- Schleifen mit mehreren Eingängen möglich (auch mit `switch`)
- Oft genügt `break`, `continue`, `return`
- Manchmal ist `goto` aber das Mittel der Wahl
- Häufige Verwendung: Fehlerbehandlung und Resourcenfreigabe

Beispiel: goto, Resourcenfreigabe

SAARLAND UNIVERSITY
COMPUTER SCIENCE

```
int allocate(void)
{
  if (!allocate_a())
  {
    return -1;
  }
  if (!allocate_b())
  {
    free_a();
    return -1;
  }
  if (!allocate_c())
  {
    free_b();
    free_a();
    return -1;
  }
  /* do things */
  return 0;
}
```

- Schreibaufwand $O(n^2)$, fehleranfällig

Beispiel: goto, Resourcenfreigabe

SAARLAND UNIVERSITY
COMPUTER SCIENCE

```
int allocate(void)
{
  if (allocate_a())
  {
     if (allocate_b())
     {
       if (allocate_c())
       {
         /* do things */
         return 0;
       }
       free_b();
     }
     free_a();
   }
   return -1;
}
```

- Viele Einrückungen

Beispiel: goto, Resourcenfreigabe

SAARLAND UNIVERSITY
COMPUTER SCIENCE

```
int allocate(void)
{
  if (!allocate_a()) goto out_a;
  if (!allocate_b()) goto out_b;
  if (!allocate_c()) goto out_c;
  /* do things */
  return 0;

  /* error handling */
out_c:
  free_b();
out_b:
  free_a();
out_a:
  return -1;
}
```