dbis
Institut für Informatik
Humboldt-Universität zu Berlin

Die Programmiersprache C

2. Strukturen, Felder und Funktionen

Vorlesung des Grundstudiums
Prof. Johann-Christoph Freytag, Ph.D.
Institut für Informatik, Humboldt-Universität zu Berlin
SoSe 2018

If-Anweisung

- Grundform:

  if (*expression*) statement

  if (*expression*) $statement_1$ else $statement_2$

- Schachtelung möglich:

  if (*expression*) $statement_1$

  else if (*expression*) $statement_2$

  else $statement_3$

- Beispiel:

```
int main()
{  int x, y, w, z;
   if (x>0) { z=w; ... }
    else     { z=y; ...   }   }
```

Der »? :«-Operator

- Der »? :«-Operator (»ternary condition«) ist die effizientere Form, um einfache if-Anweisungen auszudrücken
- syntaktische Form:
  $expression_1$? $expression_2$: $expression_3$
- Semantik:
  if e$xpression_1$ then $expression_2$ else $expression_3$
- Beispiel: Zuweisung des Maximums von a und b auf z

```
z = (a>b) ? a : b;
```

äquivalent zu:

```
if (a>b) z = a; else z=b;
```

Die switch- Anweisung

- Die switch- Anweisung erlaubt mehrfache Alternativen einer Selektion auf einer »Ebene«

switch (*expression*)
{ case $item_1$: $statement_1$ break;
case $item_2$: $statement_2$ break;
case $item_n$: $statement_n$ break;
default : *statement* break; }

- In jeder Alternative muss der Wert von $item_i$ eine Konstante sein, Variablen sind nicht erlaubt

- »Null«-Anweisung durch ein »;«

Die switch-Anweisung (Forts.)

- Beispiel:

```
switch (letter)
{
        case 'A': ;
        case 'E': ;
        case 'I' : ;
        case 'O': ;
        case 'U': numberofvowels++; break;
        case ' ': numberofspaces++; break;
        default: numberofothers++; break;
}
```

Schleifen und Iterationen

- for- Anweisung
- while- Anweisung
- do-while- Anweisung
- break- Anweisung
- continue- Anweisung
- Rekursion

Die for-Anweisung

- Die for-Anweisung hat die folgende Form:

  for (*for-init-statement; expression*$_1$; *expression*$_2$) *statement*

  Erklärung:
  - *for-init-statement* initialisiert die Iteration
  - *expression*$_1$ ist der Test zur Beendigung der Iteration
  - *expression*$_2$ modifiziert eine Schleifenvariable (mehr als nur das Erhöhen eine Schleifenvariablen um 1)
- Bemerkung:

  C benutzt for-Anweisung oft anstelle von while-Schleifen
- Beispiel:

```
int main() { int x; for (x=3;x>0;x--) { printf("x=%d \n",x); } }
```

  ... erzeugt als Ausgabe:

  x=3 x=2 x=1 (auf 3 Zeilen)

Die while-Anweisung

- Die while-Anweisung hat die folgende Form:

  while (*expression*) *statement*

  Beispiel:

  ```
  int main() {int x=3; while (x>0) { printf("x=%d \n",x); x--; } }
  ```

  ...erzeugt als Ausgabe:

  ```
  x=3 x=2 x=1
  ```

- legale while-Anweisungen:
  - while (x--);
  - while (x=x+1);
  - while (x+=5);

Die while-Anweisung (Forts.)

vollständige Ausführung von Operationen im while-Ausdruck:

- while (i++ < 10);
- while ((ch = getchar()) != 'q')
  putchar(ch);

Die do-while-Anweisung

do-while-Anweisung hat die Form:

do *statement* while (*expression*); <- hier Semikolon explizit !

Beispiel:

```
int main()
{int x=3; do { printf("x=%d \n", x--); } while (x>0); }
```

... erzeugt als Ausgabe:

x=3 x=2 x=1

break und *continue*

C enthält zwei Möglichkeiten zur Schleifensteuerung:

- *break*: Verlassen der Schleife oder der switch-Anweisung.
- *continue*: Überspringen einer Schleifeniteration

Beispiel:

```
while (scanf("%d", &value ) == 1 && value != 0)
    { if (value < 0)
        { printf("Illegal value \n"); break; /* Abandon the loop */ }
      if (value > 100)
        { printf("Invalid value \n"); continue; /* Skip to start loop again */ }
      /* Process the value read */
      /* guaranteed to be between 1 and 100 */
            ....;
      ....; } /* end while value != 0 */
```

Rekursion

Beispiel (kein Problem in C):

```
/* eg. use of functions factorials */
/* fact(n) = n*(n-1)*....2*1 */
#include <stdio.h>
int fact(int n)
{
    if (n == 0) return 1;
    return n * fact(n-1);
}

int main()
{    int n, m;
     printf("Enter a number: "); scanf("%d", &n);
     m = fact(n);
     printf("The factorial of %d is %d.\n", n, m);
     return 0;
 }
```

dbis
INSTITUT FÜR INFORMATIK
HUMBOLDT-UNIVERSITÄT ZU BERLIN

Felder und Zeichenketten

Felder (Arrays)

- Beispiel:

  int listofnumbers[50];

- Achtung:
  - In C-Arrays beginnt Indizierung bei 0 and endet mit Index, der um eins kleiner ist als seine Größe.
  - Vorheriges Beispiel: Index umfasst den Wertebereich 0 bis 49

- auf Elemente des Arrays kann man folgendermaßen zugreifen:
  - thirdnumber= listofnumbers[2];
  - listofnumbers[5]= 100;
  - Aber auch: undefined = listofnumbers[50];

Felder (Forts.)

Multi-dimensionale Arrays können wie folgt definiert werden:

```
int tableofnumbers[50][50];  // zwei Dimensionen
```

- für weitere Dimensionen werden weitere »[ ]« hinzugefügt:

  ```
  int bigD[50][50][40][30].....[50];
  ```

- auf Elemente kann man wie folgt zugreifen:
  - anumber= tableofnumbers[2][3];
  - tableofnumbers[25][16] = 100;

Initialisierung von Feldern

```
int felda[5]= {0, 1, 2, 3, 4};  // semantischer Fehler: {0, 1, 2, 3, 4, 5}
    // Speicherbereitstellung für Datenobjekt mit 5 int-Elementen
    // Initialisierung
```

oder...

```
int feldb[]= {6, 7, 8, 9, 10};
    // Speicherbereitstellung für berechnete Größe des Datenobjektes
    // Initialisierung
```

Im Beispiel sind felda und feldb gleich groß

Initialisierung von Feldern (Forts.)

```
int felda[5]= {0, 1};
    // Speicherbereitstellung für Datenobjekt mit 5 int-Elementen
    // Initialisierung aller Feldelemente: 0, 1, 0, 0, 0
```

aber...

```
int feldb[]= {0, 1};
    // Speicherbereitstellung für 2 int-Elemente
    // Initialisierung
```

Im Beispiel sind felda und feldb nicht gleich groß

Zeichenketten (Strings)

- in C werden Strings definiert als Felder von char

- Beispiel: String mit 50 Zeichen

```
char name[50];
```

- C hat »per se« keine String-Operationen
- somit sind folgende Anweisung nicht möglich:

```
char firstname[50], lastname[50], fullname[100];
firstname= "Arnold"; /* Illegal */
lastname= "Schwarzenegger"; /* Illegal */
fullname= "Mr"+firstname +lastname; /* Illegal */
```

- es gibt jedoch eine String-Bibliothek (später)
- um einen String zu drucken, wird "%s" als Formatangabe benutzt:

```
printf("%s", name);
```

Funktionen

Funktionsdefinition

- Form einer Funktion

```
returntype fn_name (paramdef1, paramdef2, ...)
    { localvariables
      functioncode   }
```

bei C99 (und C++) nicht zwingend nur am Anfang

- Beispiel: Durchschnitt zweier float-Werte

```
float findaverage (float a, float b)
        {   float average;
            average= (a+b)/2;
            return average; }
```

- Aufruf der Funktion

```
void foo() { float a=5, b=15, result;
  result=findaverage(a,b);
  printf("average=%f n",result); }
```

void in Funktionen

- falls kein Wert zurückgegeben wird,
  - sollte der Rückgabewert der Funktion *void* sein
  - außerdem keine return-Anweisung mit einem Ausdruck benutzen
- Beispiel:

```
void squares() { int loop;
   for (loop=1; loop<10; loop++)
            printf("%d \n", loop*loop);
}
int
main() {
   squares(); return 1;
}
```

- Bemerkung:

  selbst wenn keine Parameter übergeben werden, müssen Klammern nach dem Funktionsnamen folgen: ()

Funktionen und Arrays

- eindimensionale Arrays können wie folgt übergeben werden:

```
float findaverage(int size, float list[]) {
    float sum=0.0; int i;
    for (i=0;i<size;i++) sum+=list[i];
    return(sum/size);
}
```

  - float list[] gibt in C an, dass *list* ein »array of float« ist, wobei die Dimension nicht aus der übergebenen Variable extrahierbar ist (wie in Java)

Funktionen und Arrays

- Multi-dimensionale Arrays werden wie folgt übergeben:

```
void printtable(int xsize, int ysize, float table[][5]) {
    int x, y;
    for (x=0;   x<xsize;  x++) {
        for (y=0; y<ysize; y++) printf(" t%f", table[x][y]);
        printf(" \n");
     }
}
```

Weitere Datentypen

Aufzählungstypen

- Aufzählungstypen setzen sich aus einer Liste von Konstanten zusammen, die auch als Integer-Werte benutzt werden können

  Beispiel:

  ```
  enum days {mon, tues, ..., sun} week;
  enum days week1, week2;
  ```

- Bemerkung: wie bei Arrays hat der erste Name den Indexwert 0
  - mon hat Wert 0, tues den Wert 1, usw.
  - week1 und week2 sind Variablen

- auch andere Werte sind möglich:

  ```
  enum escapes { bell = `\a', backspace = `\b', tab = `\t',
                 newline = `\n', vtab = `\v', return = `\r'};
  ```

- Anfangswert für Index kann auch überschrieben werden:

  ```
  enum months {jan = 1, feb, mar, ....., dec};
  ```

Strukturen

- Beispiel:

```
struct toy {  char name[50];
              int price;
              float size;
 };

struct toy arnies;
```

Deklaration einer Nutzer-Struktur *toy* und Erzeugung einer Instanz dieser Struktur *arnies*

- Bemerkung: *toy* ist nur ein **tag (Bezeichnung)** für die Struktur zur späteren Referenzierung in weiteren Deklarationen und KEIN Typname

Strukturen (Forts.)

- Variablen können auch zwischen dem } und ; einer struct-Deklaration definiert werden, i.e.:

```
struct toy { char name[50];
             int price;
             float size;
} arnies;
```

- Strukturvariablen können während der Definition initialisiert werden:

```
struct toy arnies= {"gameboy", 30, 3.5};
```

- Um auf ein Element einer Struktur zuzugreifen, wird der ».«-Operator benutzt

  Beispiel:

```
arnies.price=100;
```

Definition neuer Datentypen

- *typedef* kann auch für Strukturdeklarationen benutzt werden
  Beispiel:

```
typedef struct toy
        { char name[50];
          int price;
          float size; } atoy;
atoy arnies={"gameboy",30,7.4};
```

- C erlaubt auch Felder von Strukturen

```
typedef struct toy {
  char name[50];
  int price;
  float size;
} atoy;
atoy arniestoys[1000];
```

- auf die kann wie folgt zugegriffen werden:
  - arniestoys[50].price= 100;
  - myprice= arniestoys[0].price;

Fragen