



Programmierung I

Vorlesungsskript

Falk Jonatan Strube

Vorlesung von Prof. Dr.-Ing. Beck

11. November 2015

Inhaltsverzeichnis

1	Einführung	1
1.1	Algorithmus	1
1.2	Programmablaufplan (PAP)	2
1.3	Struktogramm	2
1.4	Quelltext in C	2
2	gcc	3
3	Grundlagen von C	3
3.1	Datentypen	3
3.2	Ausdrücke	4
3.2.1	Assoziativität	4
3.3	Anweisungen	5
3.3.1	Ausdrucksanweisung	5
3.3.2	Alternativanweisung	6
3.3.3	Leeranweisung	6
3.3.4	Iteration	6
3.4	Zusammenfassendes Beispiel	9
3.5	Zeichenketten	10
3.6	Funktionen	11
3.7	Gültigkeit	11
3.8	Header-File	12

Hinweise

Zugelassene Hilfsmittel Klausur: Spickzettel A-4 Blatt, doppelseitig (, man-page c++.com)

1 Einführung

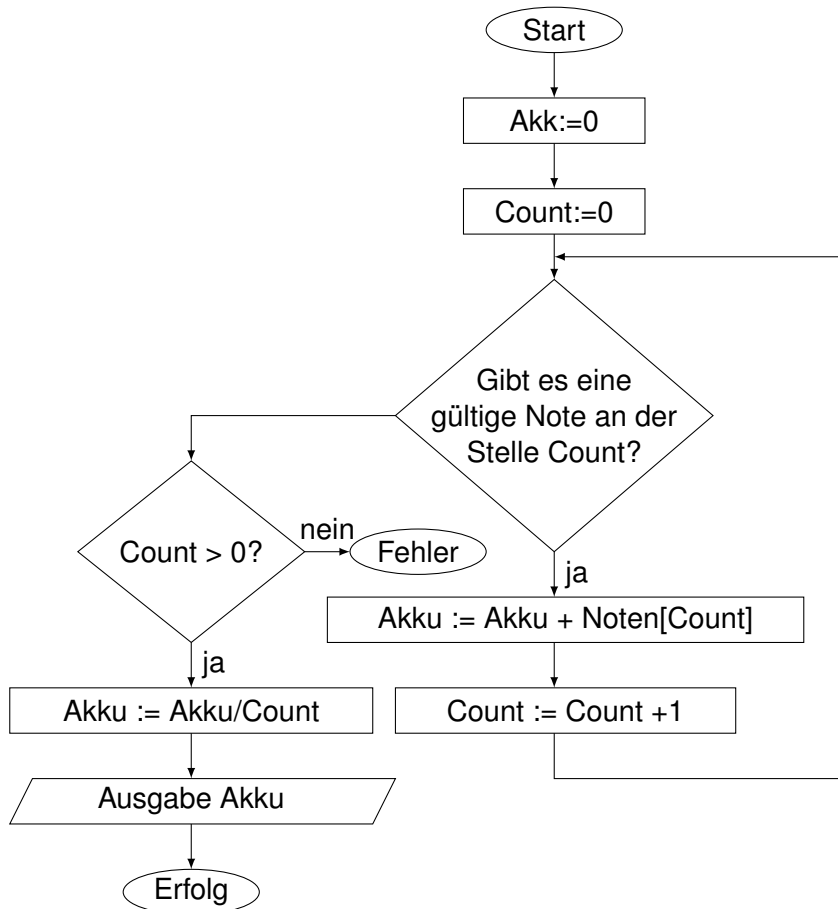
Bilde Durchschnitt aus folgender Notenübersicht:

Index	Note
0	3
1	4
2	1
3	3
4	3
5	5
6	3
7	4
8	0
9	-

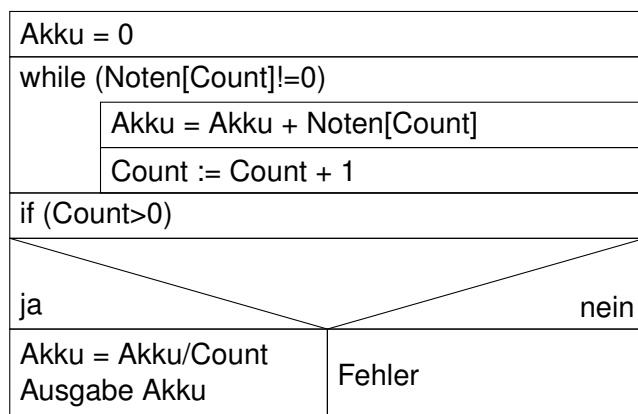
1.1 Algorithmus

- 1.) Lösche Akku → 2.
- 2.) Lösche Counter → 3.
- 3.) Gibt es eine Zahl an Stelle Count?
 - Ja: → 4.
 - Nein: → 6.
- 4.) Addiere markierte Zahl zu Akku → 5.
- 5.) Addiere 1 zu Counter → 3.
- 6.) Dividiere Wert in Akku durch Wert in Counter und speichere Akku → 7.
- 7.) Ergebnis: Ausgabe des Akku → ENDE

1.2 Programmablaufplan (PAP)



1.3 Struktogramm / Nassi-Shneiderman-Diagramm



1.4 Quelltext in C

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int Noten []={5,2,3,4,5,5,2,3,4,5,0}; //38/10
5
6 int main(){
7     int Akku=0, Count=0;

```

```

8  while (Noten[Count]!=0){
9      Akku = Akku+Noten[Count];
10     Count = Count+1;
11 }
12 if (Count>0){
13     Akku = Akku/Count;
14     printf("Durchschnitt: %d\n",Akku);
15 } else
16     printf("Fehler – Division durch 0\n");
17 return 0;
18 }

```

Compilieren durch:

```
1 gcc SOURCE.c -o DESTINATION
```

Ergebnis:

„10“ ... aber: $38/10 = 3,8$. Integer im Source-Code \Rightarrow abgerundet

Lösung:

```

1 // Ergebnis mit Runden (innerhalb der if (Count>0)–Klammer)
2 Akku = Akku*10/Count;
3 printf("Durchschnitt: %.1d\n",Akku/10,Akku%10);

```

Alternativ:

```

1 // anstatt int Akku=0, Count=0;
2 double Akku=0;
3 int Count =0;

```

2 gcc

gcc Ablauf für eine „hello.c“ Datei.

- 1.) Pre-Prozessor (hello.c \rightarrow hello.e \Rightarrow gcc -E hello.c > hello.e)
Jede Zeile im Quelltext mit # werden hier interpretiert.
- 2.) Compiler (hello.e \rightarrow hello.o \Rightarrow gcc -c hello.c)
- 3.) Linker (hello.o \rightarrow a.out / hello.exe | gcc hello.c \rightarrow a.out \Rightarrow gcc -o hello hello.c [oder auch gcc hello.c -o hello])
Bindet Objekt-Datei (xxx.o) mit Librarys zusammen.

3 Grundlagen von C

FOLIE „Grundlagen von C“

3.1 Datentypen

was in Folien grau markiert ist, kann weggelassen auch werden \Rightarrow „unsigned int i;“ = „unsigned i;“

```

1 unsigned int i; // Variablen-Definition
2 i = 12; // Wertzuweisung
3
4 printf("Wert von i: %d – Adresse von i: %p\n", i, &i);
5 // Hinweis:
6 // %d – Dezimalwert,
7 // %p – Adresswert,
8 // &i – Adresse vor Variable

```

Erstellung einer Variablen (int i;): *uninitialisierte Variable / Variablen-Definition*

Wertbelegung einer Variable während Definition einer Variablen (int i=0;): *Initialisierung*

Wertbelegung zu späterem Zeitpunkt (i=2;): *Wertzuweisung*

3.2 Ausdrücke

Programmiersprachliche Konstruktion zur Berechnung von Werten.

3.2.1 Assoziativität

(Folie Operatoren: Gewichtung der Operatoren von oben nach unten)

Unäre Operatoren (bspw. – (negativ-Zeichen), ++ (Inkrementierung) oder Klammern(cast))

Binäre Operatoren (bspw. +, – (Rechenzeichen), <= usw.)

```

1 int i;
2 long d;
3
4 i=(int)d; // cast: Typwandlung
5
6 i++; // Postfixoperator (wird im Rahmen eines groesseren
7     Ausdrucks als letztes ausgefuehrt:)
8 i=1;
9 j=6;
10 k=j+i++; // k=7, i=2
11 ++i; // Praefixoperator (wird im Rahmen eines groesseren
12     Ausdrucks als erstes ausgefuehrt:)
13 i=1;
14 j=6;
15 k=j+ ++i; // k=8, i=2
16
17 // Vorsicht! negativ-Bsp, wie ++ nicht zu verwenden ist:
18 i=2;
19 printf("%d\n", i++ + ++i);  \\ i= 6
20 printf("%d\n", i);  \\ i=4

```

Bei Division mit ganzen Zahlen wird der Rest abgeschnitten (nicht gerundet)!

Kurzschlussverfahren von Aneinanderkettung von Bedingungen (i<0 || i<6) ⇒ wenn die erste Prüfung wahr ist, wird der Test weiterer Bedingungen abgebrochen (bei && wenn das erste falsch ist).

&& im Vergleich zu & (& ist eine Bit-weise Operation):

01101100 & 00001111 = 00001100 bzw.

01101100 | 11110000 = 11111100

Andere Zeichen:

^ = XOR

~ = Bit-weise Negation

<< = shift (nach links) (bsp. i=4; i= i << 2; ⇒ i wird 16:

00000100 << 2 ⇒ 00010000

Achtung: bei negativen Zahlen (also Typ signed) bleibt bei Shift an der ersten Stelle das entsprechende Vorzeichenbit.

Bsp. für Abarbeitungsreihenfolge der Operatoren:

i*= 3+1 // i*(3+1)

3.3 Anweisungen

- Berechnungen
- Alternative
- Iteration
- Sequenz

3.3.1 Ausdrucksanweisung (Expressionstatement)

Eine Ausdrucksanweisung besteht aus einem Ausdruck gefolgt von einem Semikolon:

```
1 <expr_stmt>:: <expr> ';' .
```

Bsp.:

```
1 printf("%d\n", i);
```

Zu Ausdrucksanweisungen gehören:

- Berechnungen
- Aufrufe von Funktionen

Block Konstruktion, die Anweisungen kapselt – nach außen einzelne Anweisungen enthält

- Vereinbarungen
- Anweisungen

```
1 <block>:: '{' { <statement> } '}' .
```

Bsp.:

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 char vbuf[128]; // Vereinbarung
5
6 int main() {
7     int i; // Vereinbarung
8     double x; // Vereinbarung
9     fgets(vbuf, 128, stdin); // Anweisungen ...
10    x=atof(vbuf);
11    i = 1;
12    x=x*10+i;
13    printf("x: %lf\n",x);
14 }
```

3.3.2 Alternativanweisung (if-statement)

```

1 <if-stmnt>:: 'if' '(' <condition> ')'
2     <statement>
3     ['else' <statement>] .

```

Bsp.:

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 char vbuf[128];
5
6 int main() {
7     double x;
8     fgets(vbuf, 128, stdin);
9     x=atof(vbuf);
10    printf("x: %lf\n",x);
11    if (x>1) printf("Groesser als 1\n");
12    else printf("Kleiner als 1\n"); // optional
13    puts("Hier geht es weiter");
14    // " " Strings (Zeichenketten), einzelnes Zeichen: '*'
15 }

```

3.3.3 Leeranweisung

```

1 <empty_stmnt>:: ';'

```

3.3.4 Iteration (Schleife/Loop)

abweisende Schleife (kopfgesteuert) while-Schleife

```

1 <while_statement>:: 'while' '(' <condition> ')' <statement> .

```

Beispiel: e^x

e hoch $x = 1 + x/1! + x^2/2! + x^3/3! \dots$

1. Summand: $x^0 = 1$
2. Summand: $x^1/1! = x$
3. Summand: $(x^1/1!) * x/2 = x^2/2!$
4. Summand: $(x^2/2!) + x/3 = x^3/3!$

Vereinfachung der Rechnung (für den Rechner) \Rightarrow Nutzung des vorhergehenden Summanden.

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 char vbuf[128];
5
6 int main() {
7     int i=1;
8     double x, y=1.0, summand = 1.0;
9     printf("Eingabe von x: ");
10    fgets(vbuf, 128, stdin);
11    x=atof(vbuf);
12    while (summand>0,00005){
13        summand = summand *x/i;
14        y += summand;
15        printf("Summand %d: %lf\n", i, summand);
16        i++;
17    }

```



```

18  printf("e^%lf: %lf\n", x, y);
19  return 0;
20 }

```

Nicht abweisende Schleife (fußgesteuert) do-while-Schleife

```

1 <do_stmt>:: 'do' <statement> 'while' '(' <condition> ')' ';' .

```

Bsp.:

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 char vbuf[128];
5
6 int main() {
7     int i=1;
8     double x, y=1.0, summand = 1.0;
9     printf("Eingabe von x: ");
10    fgets(vbuf, 128, stdin);
11    x=atof(vbuf);
12    do{
13        summand = summand *x/i;
14        y += summand;
15        printf("Summand %d: %lf\n", i, summand);
16        i++;
17    } while (summand>0,00005);
18    printf("e^%lf: %lf\n", x, y);
19    return 0;
20 }

```

```

1 <for_stmt>:: 'for' '(' <expr>;' <expr>;' <expr> ')' <statement> .

```

Bsp.:

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 char vbuf[128];
5
6 int main() {
7     int i=1;
8     double x, y=1.0, summand = 1.0;
9     printf("Eingabe von x: ");
10    fgets(vbuf, 128, stdin);
11    x=atof(vbuf);
12    for (i=1; // Schleifeninitialisierung
13        summand > 0.0005; // Abbruchbedingung / Condition
14        i++){ // Iterationsausdruck
15        summand = summand *x/i;
16        y += summand;
17        printf("Summand %d: %lf\n", i, summand);
18    }
19    printf("e^%lf: %lf\n", x, y);
20    return 0;
21 }

```

Alternativ-Bsp der for-Schleife mit Komma-Operator:

```

1 int main() {
2     int i=1;
3     double x, y, summand;

```

```

4  printf("Eingabe von x: ");
5  fgets(vbuf, 128, stdin);
6  x=atof(vbuf);
7  for (i=1, y=1.0, summand=1.0;
8  summand > 0.0005;
9  summand*=x/i, y+=summand,
10     printf("Summand %d: %lf\n", i, summand), i++){
11  }
12  printf("e^%lf: %lf\n", x, y);
13  return 0;
14 }

```

Verlassen der Schleife break

```

1  #include <stdio.h>
2  #include <stdlib.h>
3
4  char vbuf[128];
5
6  int main() {
7  int i=1;
8  double x, y=1.0, summand = 1.0;
9  printf("Eingabe von x: ");
10 fgets(vbuf, 128, stdin);
11 x=atof(vbuf);
12 while (1){
13     summand = summand *x/i;
14     y += summand;
15     printf("Summand %d: %lf\n", i, summand);
16     if (summand<0,00005) break;
17     i++;
18 }
19 printf("e^%lf: %lf\n", x, y);
20 return 0;
21 }

```

break bezieht sich auf die (von innen nach außen) nächste zu findende Schleife. Also auf die Schleife, in deren *statement* sie vorkommt.

Neuberechnung der Bedingung continue

Verlässt den Schleifenkörper (der eingebettete Anweisung) und prüft die Bedingung erneut.

```

1  #include <stdio.h>
2  #include <stdlib.h>
3
4  char vbuf[128];
5
6  int main() {
7  int i=1;
8  double x, y=1.0, summand = 1.0;
9  printf("Eingabe von x: ");
10 fgets(vbuf, 128, stdin);
11 x=atof(vbuf);
12 while (summand>0,00005){
13     summand = summand *x/i;
14     y += summand;
15     i++;
16     if (summand > 0.00005) continue;
17     printf("Summand %d: %lf\n", i-1, summand);

```

```

18 }
19 printf("e^%lf: %lf\n", x, y);
20 return 0;
21 }

```

Wenn Summand größer als 0.00005 ist, startet er die Schleife neu. Die printf() wird erst ausgeführt, wenn er kleiner ist (also das letzte mal).

Fallunterscheidung switch-Anweisung

```

1 switch (i){ // i ist ganzzahliger Ausdruck
2     case 1:
3         ... break;
4     case 2:
5         ... break;
6     default:
7         ...
8 }

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 char buf[128];
5
6 int main(){
7     int wota;
8     printf("Wochentag (1...7): ");
9     fgets(buf, 128, stdin); wota=atoi(buf);
10    switch (wota){
11        case 1: puts("Montag");    break;
12        case 2: puts("Dienstag");  break;
13        case 3: puts("Mittwoch");  break;
14        case 4: puts("Donnerstag"); break;
15        case 5: puts("Freitag");   break;
16        case 6: puts("Samstag");   break;
17        case 7: puts("Sonntag");   break;
18
19        default: puts("Die Woch hat nur 7 Tage!");
20    }
21    return 0;
22 }

```

3.4 Zusammenfassendes Beispiel

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 char buf [128];
5
6 int main(){
7     int result=0;
8     char operator=0;
9     int value;
10    while (operator!=toupper('q')){
11        printf("Eingabe Operator: ");
12        fgets(buf, 128, stdin);
13        operator = buf[0];
14        printf("Eingabe Zahl: ");
15        fgets(buf, 128, stdin);
16        value = atoi(buf);

```

```

17  switch (operator) {
18      case '+': // Erinnerung: kein "+" – nur '+' für einzelne Zeichenketten
19          result += value;
20          break;
21      case '-':
22          result -= value;
23          break;
24      case '*':
25          result *= value;
26          break;
27      case '/':
28          if (value) // bzw. value!=0 – aber !=0 kann in C weggelassen werden
29              result /= value;
30          else
31              puts("Division durch 0 ist nicht erlaubt.");
32          break;
33      case '%':
34          if (value) // bzw. value!=0 – aber !=0 kann in C weggelassen werden
35              result %= value;
36          else
37              puts("Division durch 0 ist nicht erlaubt.");
38          break;
39      case 'q':
40          break;
41      default:
42          printf("unerlaubte Operation %c\n", operator);
43  }
44  printf("result: %d\n", result);
45  }
46  }

```

3.5 Zeichenketten

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4
5  char buf [128];
6
7  int main(){
8      printf("Eingabe Zeichenkette: ");
9      fgets(buf, 128, stdin);
10     printf("Len von Str: %d\n", strlen(buf));
11     buf[strlen(buf)-1]=0;
12     puts(buf);
13     while (buf[i]!=0)
14         printf("%c", buf[i++]);
15     printf("\n");
16     return 0;
17 }

```

Bei der Eingabe „Max“ wird bei puts() sowohl die Eingabe der Zeichenkette, als auch die Eingabe der Eingabetaste (neue Zeile) ausgegeben.

M	a	x	\ n	Ø			
---	---	---	-----	---	--	--	--

mit Ø: binäre, terminierende Null (0000 0000)

Hinweis: der Buffer muss immer noch Platz für „\ n“ und „Ø“ haben, d.h. man hat in einem Buffer der Größe von 128 nur Platz für 126 zeichen.

mit `buf[strlen(buf)-1]=0;` wird die Eingabetaste „\ n“ raus gelöscht.

Daraus ergibt sich eine Verbesserung für den Taschenrechner:

```

1 ...
2 printf("Eingabe Operator /Operand: ");
3 fgets(buf, 128, stdin);
4 operator = buf[0];
5 value = atoi(buf+1); // Buffer ab der Stelle 0+1: 1
6 ...

```

3.6 Funktionen

= Unterprogramme, zur Wiederholung von Codepassagen und zur besseren Strukturierung.

```

1 <function> :: <func_head><funcbody>.
2 <func_head> :: [<return_type>] name '('<param_list> void')'.
3 <param_list> :: [<type_name> name {'<type_name> name'}]
4 <func_body> :: <block>

```

Bsp.:

```

(func_head)      int    main()
                (return_type)
                {
(func_body)      return 0;
                }

```

Wenn kein return_type gewählt wurde, dann default: *int*.

Wenn kein return_type gebraucht wird, gibt man *void* an.

```

1 long fakult (int x) { // int x: Parameter
2     long f=1;
3     int i;
4     for (i=1; i<=x; i++){
5         f*=i;
6     }
7     return f;
8 }
9
10 char vBuf[128];
11
12 int main(){
13     double x,y;
14     printf("Eingabe x: ");
15     fgets(buf,128,stdin); x= atof(vBuf);
16     y=fakult(x);
17     printf("y: %ld \n", y);
18     return 0;
19 }

```

3.7 Gültigkeit

- Bereich im C-Quelltext, an dem ein Bezeichner sichtbar ist.
- Lebensdauer: Zeit von Erzeugung bis zur Vernichtung

```

1 static int count; // default wert 0
2 count++; // behält jedes Mal ihren Wert, im Gegensatz zu int count!

```

Speicherklassen: auto (automatische Variable): wird vom Stack erzeugt (Kellerspeicher)
lokale Variablen

extern: Variable, die in einem anderen Kontext vereinbart ist

static: leben bis zum Programmende, global-statische Variablen werden nicht exportiert, immer initialisiert, default 0

register: Variablen werden nach Möglichkeit in ein Prozessorregister gelegt (schnell)

volatile: Variablen werden immer im Speicher abgelegt

```
1 long fakult(int x); // Funktionsdeklaration
2           // (prototyp)
```

3.8 Header-File

enthält die Funktionsköpfe

```
1 #include "fe.h" // wie bspw. stdio.h kann die eigene Datei in anderen
2           // Quelltexten eingebunden werden
```