Programmierung I

Vorlesungsskript

Falk Jonatan Strube

Vorlesung von Prof. Dr.-Ing. Beck

28. Oktober 2015

Inhaltsverzeichnis

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 $\rightarrow$ 2.

2.) Lösche Counter $\rightarrow$ 3.

3.) Gibt es eine Zahl an Stelle Count?

- Ja: $\rightarrow$ 4.
- Nein: $\rightarrow$ 6.

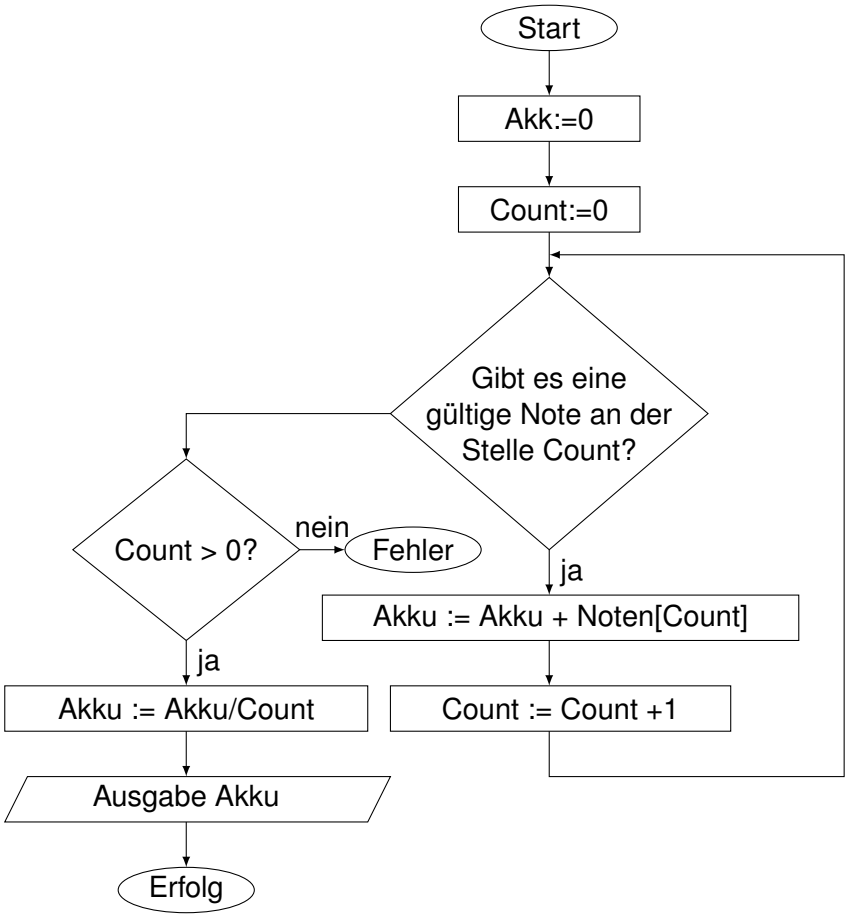
4.) Addiere markierte Zahl zu Akku $\rightarrow$ 5.

5.) Addiere 1 zu Counter $\rightarrow$ 3.

6.) Dividiere Wert in Akku durch Wert in Counter und speichere Akku $\rightarrow$ 7.

7.) Ergebnis: Ausgabe des Akku $\rightarrow$ ENDE

1.2 Programmablaufplan (PAP)



1.3 Struktogramm / Nassi-Shneiderman-Diagramm

Akku = 0	
while (Noten[Count]!=0)	
Akku = Akku + Noten[Count]	
Count := Count + 1	
if (Count>0)	
ja	nein
Akku = Akku/Count Ausgabe Akku	Fehler

1.4 Quelltext in C

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

int Noten []={5,2,3,4,5,5,2,3,4,5,0}; //38/10
```

```
int main(){
  int Akku=0, Count=0;
  while (Noten[Count]!=0){
    Akku = Akku+Noten[Count];
    Count = Count+1;
  }
  if(Count>0){
    Akku = Akku/Count;
    printf("Durchschnitt: %d\n",Akku);
  } else
    printf("Fehler - Division durch 0\n");
  return 0;
}
```

Compilieren durch:

```
gcc SOURCE.c -0 DESTINATION
```

Ergebnis:
„10“ ... aber: 38/10 = 3,8. Integer im Source-Code ⇒ abgerundet
Lösung:

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

Alternativ:

```
// anstatt int Akku=0, Count=0;
double Akku=0;
int Count =0;
```

2 gcc

gcc Ablauf für eine „hello.c“ Datei.

1.) Pre-Prozessor (hello.c → hello.e ⇒ gcc -E hello.c > hello.e)
Jede Zeile im Quelltext mit # werden hier interpretiert.

2.) Compiler (hello.e → hello.o ⇒ gcc -c hello.c)

3.) Linker (hello.o → a.out / hello.exe | gcc hello.c → a.out ⇒ 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 ⇒ „unsigned int i;“ = „unsigned i;“

```
unsigned int i; // Variablen-Definition
i = 12;   // Wertzuweisung

printf("Wert von i: %d - Adresse von i: %p\n", i, &i);
// Hinweis:
// %d - Dezimalwert,
// %p - Adresswert,
// &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.)

```
int i;
long d;

i=(int)d; // cast: Typwandlung

i++; // Postfixoperator (wird im Rahmen eines groesseren
       Ausdrucks als letztes ausgefuehrt:)
i=1;
j=6;
k=j+i++; // k=7, i=2
++i; // Praefixoperator (wird im Rahmen eines groesseren
       Ausdrucks als erstes ausgefuehrt:)
i=1;
j=6;
k=j+ ++i; // k=8, i=2

// Vorsicht! negativ-Bsp, wie ++ nicht zu verwenden ist:
i=2;
printf("%d\n", i++ + ++i);   \\ i= 6
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) $\Rightarrow$ 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:
$\wedge$ = XOR
$\sim$ = Bit-weise Negation
<< = shift (nach links) (bsp. i=4; i= i << 2; $\Rightarrow$ i wird 16:
00000100 << 2 $\Rightarrow$ 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:

```
<expr_stmnt>:: <expr> ';' .
```

Bsp.:

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

Zu Ausdrucksanweisungen gehören:

- Berechnungen
- Aufrufe von Funktionen

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

- Vereinbarungen
- Anweisungen

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

Bsp.:

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

char vbuf[128]; // Vereinbarung

int main() {
```

```
  int i;  // Vereinbarung
  double x; // Vereinbarung
  fgets(vbuf, 128, stdin);  // Anweisungen ...
  x=atof(vbuf);
  i = 1;
  x=x*10+i;
  printf("x: %lf\n",x);
}
```

3.3.2 Alternativanweisung (if-statement)

```
<if-stmnt>:: 'if' '(' <condition> ')'
        <statement>
      ['else' <statement>] .
```

Bsp.:

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

char vbuf[128];

int main() {
  double x;
  fgets(vbuf, 128, stdin);
  x=atof(vbuf);
  printf("x: %lf\n",x);
  if (x>1) printf("Groesser als 1\n");
  else printf("Kleiner als 1\n"); // optional
  puts("Hier geht es weiter");
  // " " Strings (Zeichenketten), einzelnes Zeichen: '*'
}
```

3.3.3 Leeranweisung

```
<empty_stmnt>:: ';'
```

3.3.4 Iteration (Schleife/Loop)

abweisende Schleife (kopfgesteuert) while-Schleife

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

Beispiel: e^x
e hoch x = 1 + x/1! + x*x /2! + x*x*x /3! ...

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.

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

char vbuf[128];

int main() {
  int i=1;
  double x, y=1.0, summand = 1.0;
  printf("Eingabe von x: ");
  fgets(vbuf, 128, stdin);
  x=atof(vbuf);
  while (summand>0,00005){
    summand = summand *x/i;
    y += summand;
    printf("Summand %d: %lf\n", i, summand);
    i++;
  }
  printf("e^%lf: %lf\n", x, y);
  return 0;
}
```

Nicht abweisende Schleife (fußgesteuert) do-while-Schleife

```
<do_stmnt>:: 'do' <statement> 'while' '(' <condition> ')' ';' .
```

Bsp.:

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

char vbuf[128];

int main() {
  int i=1;
  double x, y=1.0, summand = 1.0;
  printf("Eingabe von x: ");
  fgets(vbuf, 128, stdin);
  x=atof(vbuf);
  do{
    summand = summand *x/i;
    y += summand;
    printf("Summand %d: %lf\n", i, summand);
    i++;
  } while (summand>0,00005);
  printf("e^%lf: %lf\n", x, y);
  return 0;
}
```

```
<for-stmnt>:: 'for' '(' <expr>';' <expr>';' <expr> ')' <statement> .
```

Bsp.:

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

char vbuf[128];

int main() {
  int i=1;
  double x, y=1.0, summand = 1.0;
  printf("Eingabe von x: ");
  fgets(vbuf, 128, stdin);
  x=atof(vbuf);
  for (i=1;   // Schleifeninitialisierung
  summand > 0.0005;   // Abbruchbedingung / Condition
  i++){ // Iterationsausdruck
    summand = summand *x/i;
    y += summand;
    printf("Summand %d: %lf\n", i, summand);
  }
  printf("e^%lf: %lf\n", x, y);
  return 0;
}
```

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

```c
int main() {
  int i=1;
  double x, y, summand;
  printf("Eingabe von x: ");
  fgets(vbuf, 128, stdin);
  x=atof(vbuf);
  for (i=1, y=1.0, summand=1.0;
  summand > 0.0005;
  summand*=x/i, y+=summand,
     printf("Summand %d: %lf\n", i, summand), i++){
  }
  printf("e^%lf: %lf\n", x, y);
  return 0;
}
```

test