

Kontrollstrukturen in C

19.11.2004

Hochschule für angewandte Wissenschaften Hamburg
Prof.Dr.Ing.U.Vogelsang, Prof.Dr.U.Sauvagerd



Programmieren in ANSI C:
K5_C-Kontrollstrukturen.ppt, Seite 1

Kontrollstrukturen in C

- In diesem Kapitel wollen wir zeigen,
 - ➡ wie man in C den Programmablauf steuern kann
 - ➡ was Programmschleifen sind
 - ➡ auf welche unterschiedliche Weise man in C Schleifen programmieren kann
 - ➡ wodurch sich die unterschiedlichen Schleifenkonstrukte unterscheiden.

19.11.2004

Hochschule für angewandte Wissenschaften Hamburg
Prof.Dr.Ing.U.Vogelsang, Prof.Dr.U.Sauvagerd



Programmieren in ANSI C:
K5_C-Kontrollstrukturen.ppt, Seite 2

Was sind Kontrollstrukturen?

- Kontrollstrukturen sind C-Befehle und Konstrukte, die den **Programmablauf steuern**.
- Dies sind:
 - ➔ **bedingte Anweisungen**, bei denen ein Programmteil **in Abhängigkeit von Bedingungen** ausgeführt wird.
 - ➔ **Programmschleifen**, in denen die gleichen Anweisungen mehrfach durchlaufen werden.
 - ➔ **unbedingte Programmverzweigungen**, bei denen das Programm **in jedem Falle** an einer anderen Stelle fortgesetzt wird.
 - ➔ **bedingte Programmverzweigungen**, bei denen das Programm **unter bestimmten Bedingungen** an einer anderen Stelle fortgesetzt wird.

19.11.2004

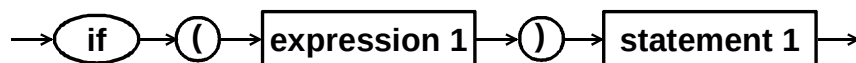
Hochschule für angewandte Wissenschaften Hamburg
Prof.Dr.Ing.U.Vogelsang, Prof.Dr.U.Sauvagerd



Programmieren in ANSI C:
K5_C-Kontrollstrukturen.ppt, Seite 3

Bedingte Anweisung mit *if* in C

➔ **if ...** ➔



Beispiel:

Bedingte Anweisung

➔ **if (a > max)**
 max = a ;

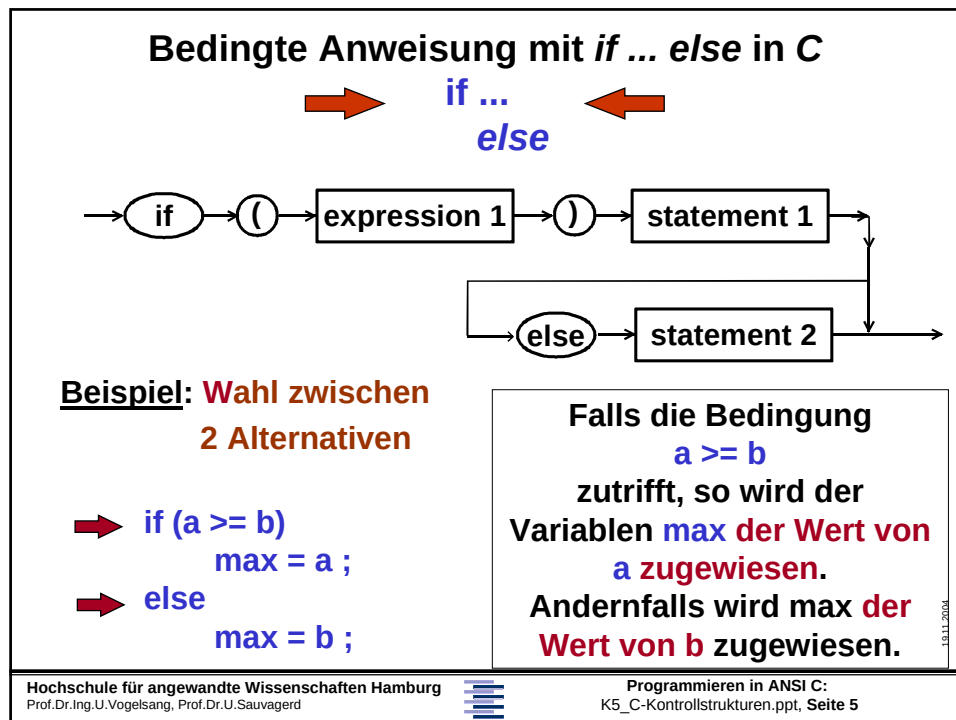
Falls die Bedingung **a > max** zutrifft, so wird der Variablen **max** der Wert von **a** zugewiesen. Andernfalls wird **max** **hier kein Wert** zugewiesen.

19.11.2004

Hochschule für angewandte Wissenschaften Hamburg
Prof.Dr.Ing.U.Vogelsang, Prof.Dr.U.Sauvagerd



Programmieren in ANSI C:
K5_C-Kontrollstrukturen.ppt, Seite 4



Wann verwendet man Programmschleifen?

Programmschleifen wendet man an, wenn sich

➡ **Variable abzählbar** ändern.

Beispiele:

- Vektoren
- Matrizen
- Felder
- Tabellen
- Zeichenketten (strings)

Hochschule für angewandte Wissenschaften Hamburg
 Prof.Dr.Ing.U.Vogelsang, Prof.Dr.U.Sauvagerd

Programmieren in ANSI C:
 K5_C-Kontrollstrukturen.ppt, Seite 6

while-Schleife in C (1)

➡ while ⬅

```

→ (while) ( ( expression ) ) statement →

```

- Die Bedingung (*expression* = Ausdruck) wird **am Schleifenanfang** geprüft.
- Falls die Bedingung erfüllt ist, wird die Anweisung (*statement*) **ausgeführt**, gegebenenfalls auch gar nicht.

Beispiel:

```

short i = 10, min = 0, sum = 0;
while ( i > min ) ⬅
{
    sum += i ;
    i-- ;
}

```

Ist *i > min*, so wird das statement (hier der Block)

```

{ summe += i ;
  ...
}

```

ausgeführt, sonst nicht.

Hochschule für angewandte Wissenschaften Hamburg
Prof.Dr.Ing.U.Vogelsang, Prof.Dr.U.Sauvagerd



Programmieren in ANSI C:
 K5_C-Kontrollstrukturen.ppt, Seite 7

while-Schleife in C (2)

➡ while ⬅

- Sonderfall:**

while (1) ... ;

ist eine **Endlosschleife**.

Beispiel:

```

short i = 0 ;
while ( 1 )
{
    i++ ;
    /* weitere Befehle */
    if ( i > 1000 ) break ;
}

```

Ist *i > 1000*, so wird die while-Schleife abgebrochen.
Die *break*-Anweisung wird später erläutert.

Hochschule für angewandte Wissenschaften Hamburg
Prof.Dr.Ing.U.Vogelsang, Prof.Dr.U.Sauvagerd



Programmieren in ANSI C:
 K5_C-Kontrollstrukturen.ppt, Seite 8

do ... while-Schleife in C

➡ **do ... while** ⬅



- Die Bedingung (*expression* = Ausdruck) wird **am Schleifenende** geprüft
- Die Anweisung (*statement*) bzw. der Block (*compound statement*) wird **immer mindestens 1 Mal ausgeführt**.

Beispiel:

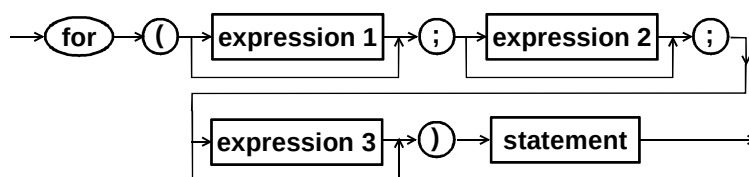
```
short j = 0, sum = 0,
ende = 25;
```

```
➡ do
    { sum += j;
      j += 2; }
➡ while (j < ende);
```

Mindestens 1 mal wird **sum** um **j** erhöht.
Danach wird **j** um **2** erhöht.
Ist **j ≥ ende**, so wird die Schleife abgebrochen, andernfalls fortgesetzt.

**for-Schleife in C (1)**

➡ **for** ⬅

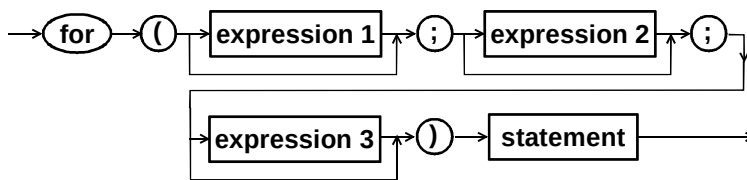


- Der 1. Anweisung (*expression 1*) wird **am Schleifenanfang** eine Variable **zugewiesen**.
- Mit der 2. Anweisung (*expression 2*) wird die **Schleifenbedingung vor der Schleife überprüft**.
 - Der 3. Ausdruck (*expression 3*) **verändert den Schleifenzähler**.
 - Der Schleifenbefehl (*statement*) **führt die Schleifenoperation aus**.



for-Schleife in C (2)

➡ **for** ⬅



- Die Programmschleife wird **gegebenenfalls nicht ein einziges Mal** ausgeführt.

Beispiel:

```
short n, prod = 1 ;
↓
for ( n = 1 ; n < 8 ; n++ )
    { prod *= n ;
      n++ ; }
```

Zunächst wird **n** auf **1** gesetzt. Danach wird geprüft, ob **n < 8** ist. Falls nicht, so wird die Schleife verlassen. Andernfalls wird der Ausdruck {...} berechnet und **danach n** um **1** erhöht.

19.11.2004

Hochschule für angewandte Wissenschaften Hamburg
Prof.Dr.Ing.U.Vogelsang, Prof.Dr.U.Sauvagerd



Programmieren in ANSI C:
K5_C-Kontrollstrukturen.ppt, Seite 11

for-Schleife in C (3)

➡ **for** ⬅

Sonderfall:

```
for ( ; ; )
    ... ;
```

ist eine **Endlosschleife**.

19.11.2004

Hochschule für angewandte Wissenschaften Hamburg
Prof.Dr.Ing.U.Vogelsang, Prof.Dr.U.Sauvagerd



Programmieren in ANSI C:
K5_C-Kontrollstrukturen.ppt, Seite 12

continue-Abbruchbedingung in C

➡ **continue** ⬅

- Den gegenwärtigen **Schleifendurchlauf abbrechen**, Schleife **gegebenenfalls** aber mit dem **nächsten Durchlauf** fortsetzen.

Beispiel:

```
int j = 1000 ;
```

```
while ( j != 0 )
{ /* ... */
  j-- ;
  ➡ if ( j % 7 == 0 )
    continue ;
  /* ... */
}
```

Falls die Abbruchbedingung
if (j % 7 == 0)
zutrifft,
so wird die Schleife
am Anfang fortgesetzt.

19.11.2004

Hochschule für angewandte Wissenschaften Hamburg
Prof.Dr.Ing.U.Vogelsang, Prof.Dr.U.Sauvagerd



Programmieren in ANSI C:
K5_C-Kontrollstrukturen.ppt, Seite 13

break-Abbruchbedingung in C

➡ **break** ⬅

- Den aktuellen **Schleifendurchlauf abbrechen** und das Programm **nach der Schleife fortsetzen**.

Beispiel:

```
long i = 100000l ;
```

```
while (1)
{ /* ... */
  i-- ;
  if ( i == 0 )
    ➡ break ;
  /* ... */
}
```

Falls die Abbruchbedingung
if (i == 0)
zutrifft, so wird das
Programm nach der Schleife
fortgesetzt.
Bei geschachtelten Schleifen
wird **nur die aktuelle** Schleife
verlassen.

19.11.2004

Hochschule für angewandte Wissenschaften Hamburg
Prof.Dr.Ing.U.Vogelsang, Prof.Dr.U.Sauvagerd



Programmieren in ANSI C:
K5_C-Kontrollstrukturen.ppt, Seite 14

✍ Äquivalente Schleifen

Gegeben ist das folgende Programm mit einer **for**-Schleife:

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

```
int main (void)
{
    long prod = 1 ;
    short n ;
    for ( n = 1 ; n < 8 ; n++ )
    {
        prod *= n ;
        n++ ;
        printf ( "\n n = %hd, prod = %li", n, prod ) ;
    }
    return 0 ;
}
```

1. Welche Ausgabe wird auf dem Bildschirm erzeugt?
2. Ersetzen Sie die **for**-Schleife durch äquivalente Anweisungen, die eine **while**-Schleife beinhalten!
3. Ersetzen Sie die **for**-Schleife durch äquivalente Anweisungen, die eine **do-while**-Schleife beinhalten!

Hochschule für angewandte Wissenschaften Hamburg
Prof.Dr.Ing.U.Vogelsang, Prof.Dr.U.Sauvagerd



Programmieren in ANSI C:
K5_C-Kontrollstrukturen.ppt, Seite 15

✓ Äquivalente Schleifen (1)

1. BildschirmAusgabe:

```
n = 2, prod = 1
n = 4, prod = 3
n = 6, prod = 15
n = 8, prod = 105
```

2. äquivalente Anweisungen mit einer while-Schleife:

```
/* ... */
short n = 1 ;
while ( n < 8 )
{
    prod *= n ;
    n++ ;
    printf ( "\n n = %hd, prod = %li", n, prod ) ;
    n++ ;
}
/* ... */
```

Hochschule für angewandte Wissenschaften Hamburg
Prof.Dr.Ing.U.Vogelsang, Prof.Dr.U.Sauvagerd



Programmieren in ANSI C:
K5_C-Kontrollstrukturen.ppt, Seite 16

✓ Äquivalente Schleifen (2)

3. äquivalente Anweisungen mit einer do-while-Schleife:

```
...
short n = 1 ;
do
{
    if ( n >= 8 ) break ;
    prod *= n ;
    n++ ;
    printf ( "\n n = %hd, prod = %li", n, prod ) ;
    n++ ;
}
while (1) ;
/* ... */
```



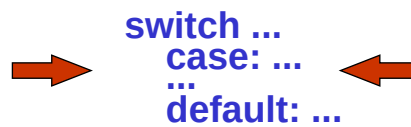
switch ... case ... default -Verzweigung in C (1)

switch ...
 case: ...
 ...
 default: ...

- Dem Schlüsselwort **switch** folgt ein Ausdruck.
- Es wird untersucht, ob der Ausdruck einen von mehreren konstanten ganzzahligen Werten hat und entsprechend verzweigt.
- Dem Schlüsselwort **case** folgt jeweils eine ganzzahlige Konstante oder ein konstanter Ausdruck.
Alle **case**-Konstanten müssen sich unterscheiden.
- Hat der **switch**-Ausdruck den Wert einer der **case**-Konstanten, so wird das Programm an dieser Stelle fortgesetzt.
- Andernfalls geht es nach dem optionalen **default** weiter.



switch ... case ... default -Verzweigung in C (2)



```

switch ...
case: ...
...
default: ...

```

- Gibt es kein **default** und paßt keine der Konstanten, dann wird keine der **switch**-Anweisungen ausgeführt.
 - Tritt ein **break** auf, so wird der switch sofort verlassen.
 - Wenn sich der **switch** innerhalb einer Schleife befindet und ein **continue** auftritt, so wird der **switch** verlassen und die Schleife mit dem nächsten Durchlauf fortgesetzt.

19.11.2004

Hochschule für angewandte Wissenschaften Hamburg
Prof.Dr.Ing.U.Vogelsang, Prof.Dr.U.Sauvagerd



Programmieren in ANSI C:
K5_C-Kontrollstrukturen.ppt, Seite 19

switch ... case ... default -Verzweigung in C (3)



```

switch ...
case: ...
...
default: ...

```

Beispiel Wahl zwischen mehr als 2 Alternativen

```

switch ( i )
{ case 0 :
  printf ( "i = 0" ) ;
  case 1 :
  printf ( "i = 0 oder i = 1" ) ;
  break ;
  case 2 :
  printf ( "i = 2" ) ;
  break ;
  default :
  printf ( "i != 0, 1, 2" ) ; }

```

Die
BildschirmAusgabe
mit **printf (..)** zeigt
an, welche **case**-
Anweisungen
durchlaufen
wurden.

19.11.2004

Hochschule für angewandte Wissenschaften Hamburg
Prof.Dr.Ing.U.Vogelsang, Prof.Dr.U.Sauvagerd



Programmieren in ANSI C:
K5_C-Kontrollstrukturen.ppt, Seite 20

Sprungbefehl goto in C (1)



```
goto <Marke> ...
...
<Marke> ...
```

- **Sprunganweisungen:** goto <Marke> sollten in C **vermieden werden**.
- Sie sind nicht notwendig, führen zu **unübersichtlicher Software** und sind **fehlerträchtig**.

19.11.2004



Sprungbefehl goto in C (2)

- Das folgende **schlechte Programmbeispiel** dient lediglich der Erläuterung von goto <Marke> -Befehlen. Es ermittelt das Maximum der eingelesenen Messwerte.

```
#include <stdio.h>
#include <float.h>      /* Fuer FLT_MAX */
int main ( void )
{
    float Max_Wert = -FLT_MAX; /* kleinster darstellbarer float-Wert */
    float Messwert ;
    int ok ;
    while ( 1 )
    {
        printf ( "Messwert im float-Format = ?: " ) ;
        ok = scanf( "%f", &Messwert ) ;
        if ( ok!=1 ) break ;
        if ( Max_Wert >= Messwert ) goto next ;
        Max_Wert = Messwert ;
        next: printf ( "aktueller Maximalwert = %f \n", Max_Wert ) ;
    }
    return 0 ; }
```

19.11.2004





C-typische Programmierung

Schreiben Sie das folgende **C-untypische** Beispielprogramm aus der Vorlesung ohne den `goto <Marke>` -Befehl in einer **modularen Struktur!**

```
#include <stdio.h>
#include <float.h>      /* Fuer FLT_MAX */
int main ( void )
{
    float Max_Wert = -FLT_MAX;
    float Messwert ;
    int ok ;
    while ( 1 )
    {
        printf ( "Messwert im float-Format = ?: " ) ;
        ok = scanf( "%f", &Messwert ) ;
        if ( ok!=1 ) break ;
        if ( Max_Wert >= Messwert ) goto next ;
        Max_Wert = Messwert ;
        next: printf ( "aktueller Maximalwert = %f \n", Max_Wert ) ;
    }
    return 0 ; }
```

19.11.2004

Hochschule für angewandte Wissenschaften Hamburg
Prof.Dr.Ing.U.Vogelsang, Prof.Dr.U.Sauvagerd



Programmieren in ANSI C:
K5_C-Kontrollstrukturen.ppt, Seite 23



C-typische Programmierung

- Beispielprogramm aus der Vorlesung ohne
`goto <Marke>` (modulare Struktur)

```
#include <stdio.h>
#include <float.h>      /* Fuer FLT_MAX */
int main ( void )
{
    float Max_Wert = -FLT_MAX;
    float Messwert ;
    int ok ;
    while ( 1 )
    {
        printf ( "Messwert im float-Format = ?: " ) ;
        ok = scanf( "%f", &Messwert ) ;
        if ( ok!=1 ) break ;
        if ( Max_Wert < Messwert )      Max_Wert = Messwert ;
        next: printf ( "aktueller Maximalwert = %f \n", Max_Wert ) ;
    }
    return 0 ; }
```

19.11.2004

Hochschule für angewandte Wissenschaften Hamburg
Prof.Dr.Ing.U.Vogelsang, Prof.Dr.U.Sauvagerd



Programmieren in ANSI C:
K5_C-Kontrollstrukturen.ppt, Seite 24

✎ Programm- und Fehleranalyse in C (1)

Das folgende Programm **enthält Fehler**:

```
/* Dateiname: T_mean_f.c */
#include <stdio.h>;
void main ( void )
{
    char ok ;
    float mean = 0, x ;
    unsigned char Anzahl = 0 ;
    do
    {
        printf ( "Messwert im long double-Format = ? : " )
        ok = scanf ( "%f",x ) ;
        mean *= Anzahl ;
        Anzahl++ ;
        mean = (mean + x) / Anzahl;
        printf ( "Der ??? wert ueber", %i, "Messwerte betraegt:", %f /n,
                Anzahl, mean ) ;
    } ;
    while ( ok == 1 && Anzahl <= 255 ) ;
    return 0 ; };
```

Hochschule für angewandte Wissenschaften Hamburg
Prof.Dr.Ing.U.Vogelsang, Prof.Dr.U.Sauvagerd



Programmieren in ANSI C:
K5_C-Kontrollstrukturen.ppt, Seite 25

✎ Programm- und Fehleranalyse in C (2)

Das Programm auf der vorherigen Seite **enthält Fehler**.

1. Untersuchen Sie das Programm auf **syntaktische Fehler**, **logische Fehler** und **Formatfehler**!
Erläutern Sie die Fehlerursache! (Begründung)
Auf der nächsten Folie sind die **Fehler** und **Unsauberkeiten** markiert und fehlende Zeichen eingesetzt. Vergleichen Sie die Fehler und Unsauberkeiten mit den von Ihnen gefundenen und machen Sie sich die Fehlerursache klar!
2. Welche Funktion hat das Programm? – Beschreiben Sie die Programmablauf! Erläutern Sie den Sinn der Befehle!
3. Modifizieren Sie das Programm so, dass keine Fehler durch Zahlbereichsüberschreitungen oder falsche mathematische Operationen auftreten!
Sorgen Sie für eine verständliche und übersichtliche Darstellung der Ergebnisse auf dem Bildschirm!
Sorgen sie dafür, dass zuviel eingegebene Zeichen nicht zu einem unerwünschten Programmabbruch führen!

Hochschule für angewandte Wissenschaften Hamburg
Prof.Dr.Ing.U.Vogelsang, Prof.Dr.U.Sauvagerd



Programmieren in ANSI C:
K5_C-Kontrollstrukturen.ppt, Seite 26

✓ Programm- und Fehleranalyse in C (1)

1. Syntaktische Fehler, logische Fehler & Formatfehler

```

/* Dateiname: T_mean_f.c */
#include <stdio.h>;
→ void main(void) ↑
{
→ char ok;
float mean = 0, x;
unsigned char Anzahl = 0;
do
{
printf("Messwert im double-Format = ?: ")
→ ok = scanf("%f", x);
mean *= Anzahl;
Anzahl++;
mean = (mean + x) / Anzahl;
printf("Der ??? wert ueber", %i, "Messwerte betraegt:", %f /n,
Anzahl, mean);
}; ←
while (ok == 1 && Anzahl <= 255);
return 0;}; ←

```

Diagramm zur Fehleranalyse: Rote Pfeile markieren syntaktische Fehler (z.B. fehlende Semikolons, falsche Formatierer, falsche Schleifenbedingung). Blaue Pfeile markieren logische Fehler (z.B. falsche Variablen in printf, falsche Berechnung des Mittelwerts).

Hochschule für angewandte Wissenschaften Hamburg
Prof.Dr.Ing.U.Vogelsang, Prof.Dr.U.Sauvagerd

Programmieren in ANSI C:
K5_C-Kontrollstrukturen.ppt, Seite 27

✓ Programm- und Fehleranalyse in C (2)

1. Funktion des Programms:

- Das Programm soll einen Messwert im float-Format einlesen.
- Es soll den gleitenden Mittelwert über die Messwerte bilden: Der Mittelwert soll jeweils über alle bisher eingelesenen Messwerte gebildet werden.
- Die Anzahl der Messwerte und der gleitende Mittelwert sollen ausgegeben werden.
- In einer Schleife sollen solange Messwerte eingelesen werden bis ein falsches Eingabezeichen auftritt solange die Anzahl der Messwerte kleiner als 255 ist.

2. Unsauberkeiten im Programm:

- void main (void) : in MSVC möglich, aber kein ANSI-Standard.
- scanf (..) gibt Variable vom Typ int zurück.
- vor scanf (..) fehlt fflush (stdin).
- Man sollte nicht auf <= abfragen, sondern nur auf <.

Hochschule für angewandte Wissenschaften Hamburg
Prof.Dr.Ing.U.Vogelsang, Prof.Dr.U.Sauvagerd

Programmieren in ANSI C:
K5_C-Kontrollstrukturen.ppt, Seite 28

✓ Programm- und Fehleranalyse in C (2)

2. Fehler:

- Nach Präprozessoranweisungen (#..) steht kein Semikolon.
- Formatanweisungen in 1. `printf (..)` und nachfolgendem `scanf (..)` sind inkonsistent!
- Bei `scanf (..)` fehlt vor dem `x` ein `&`.
- Hinter 1. `printf (..)` fehlt ein Semikolon.
- Im 2. `printf (..)` müssen die Formatanweisungen im string stehen.
- Im 2. `printf (..)` ist `/n` falsch, richtig: `\n`.
- Im 2. `printf (..)` ist `%i` falsch, richtig: `%u`.
- Hinter Blockklammern steht kein Semikolon.
- Die Abfrage nach `ok` in `while (..)` ist an dieser Stelle zu spät. Es entsteht beim Abbruch der Schleife ein Fehler.
- Die Abfrage auf `<=255` führt zum Zahlbereichsüberlauf für `Anzahl >255`.

Hochschule für angewandte Wissenschaften Hamburg
Prof.Dr.Ing.U.Vogelsang, Prof.Dr.U.Sauvagerd



Programmieren in ANSI C:
K5_C-Kontrollstrukturen.ppt, Seite 29

✓ Programm- und Fehleranalyse in C (4)

• Version eines korrekten Programms (1):

/* 1. Teil des Programms, 2. Teil => nächste Folie

Datum: 7.11.99

Dateiname: K5_Mittelwert.c, korrektes Programm

Kurzbeschreibung: Programm- und Fehleranalyse in C, Messwert wird eingelesen; gleitender Mittelwert wird gebildet bis 254 Messwerte eingelesen sind oder falsches Zeichen eingegeben worden ist.

Änderungen:

Name	Datum

Vogelsang	18.10.04 überarbeitet.

Vogelsang 18.10.04 überarbeitet.

*/

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

```
int main ( void )
```

```
{
```

```
    int ok ;
```

```
    float mean = 0., x ;
```

```
    double m1 = 0. ;
```

```
    unsigned char Anzahl = 0 ;
```

Hochschule für angewandte Wissenschaften Hamburg
Prof.Dr.Ing.U.Vogelsang, Prof.Dr.U.Sauvagerd



Programmieren in ANSI C:
K5_C-Kontrollstrukturen.ppt, Seite 30

✓ Programm- und Fehleranalyse in C (5)

• Version eines korrekten Programms (2):

```

/* 2. Teil des Programms, 1. Teil => vorherige Folie */
do
{
    printf ("Messwert im float-Format = ?: ");
    fflush (stdin);    ok = scanf ("%f",&x);
    if ( ok!=1 ) break ; /* Hier ist der Schleifenabbruch korrekt */
    if ( Anzahl == 0 )
    {
        mean = x ;    Anzahl++ ;
    }
    else
    {
        m1 = (double) mean * Anzahl ; /* cast wg. Überlaufgefahr */
        Anzahl++ ;
        mean = ( m1 + x ) / Anzahl ;
    }
    printf ( "Der Mittelwert ueber %u Messwerte betraegt: %f \n\n",
            Anzahl, mean ) ;
}
while ( Anzahl < 255 ) ; /* <= kann zu Anzahl > 255 führen! */
return 0 ;

```

