Fehlererkennung

C - Kurs 2011

16. September 2011

Inhaltsverzeichnis

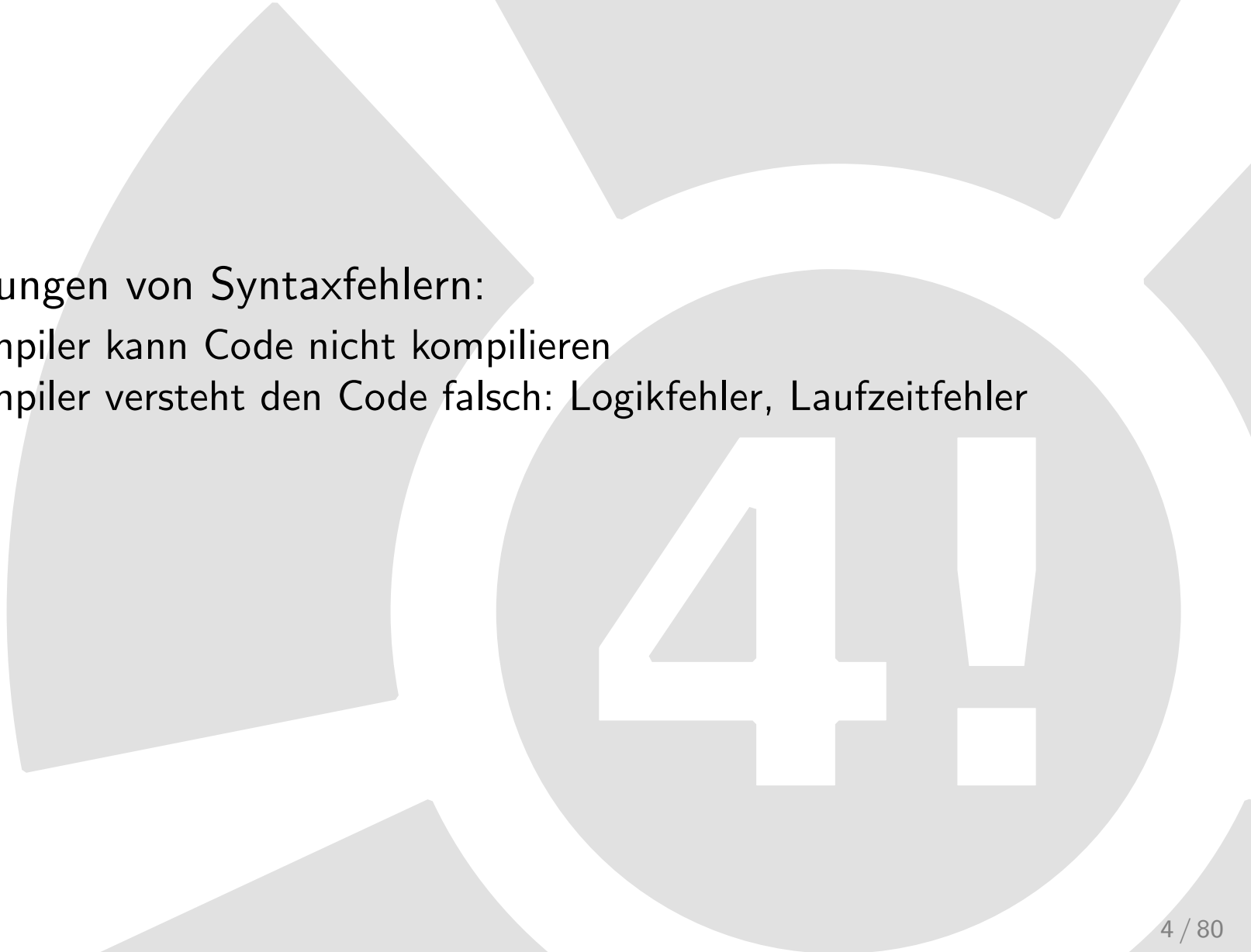
1. Fehlerarten
   - Syntaxfehler
   - Logikfehler
   - Laufzeitfehler
2. Compilerwarnungen
3. Assertions
4. GDB
   - Was ist ein Debugger?
   - Der GNU Debugger GDB
   - Wichtigsten Kommandos des GDB
   - First Steps

Inhaltsverzeichnis

1. Fehlerarten
   - Syntaxfehler
   - Logikfehler
   - Laufzeitfehler
2. Compilerwarnungen
3. Assertions
4. GDB
   - Was ist ein Debugger?
   - Der GNU Debugger GDB
   - Wichtigsten Kommandos des GDB
   - First Steps

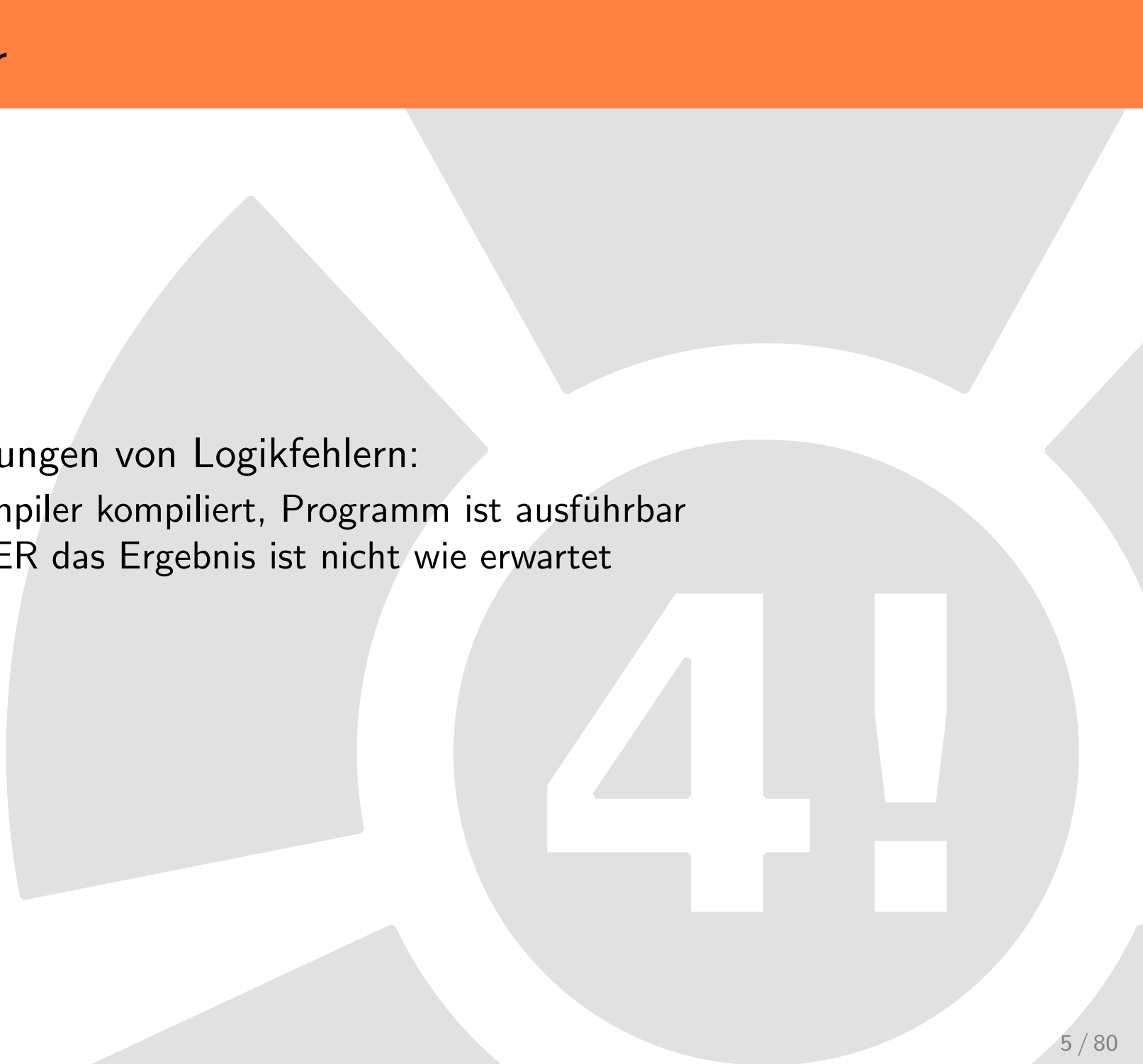
Syntaxfehler

- Auswirkungen von Syntaxfehlern:
  - Compiler kann Code nicht kompilieren
  - Compiler versteht den Code falsch: Logikfehler, Laufzeitfehler



Logikfehler

- Auswirkungen von Logikfehlern:
  - Compiler kompiliert, Programm ist ausführbar
  - ABER das Ergebnis ist nicht wie erwartet



Beispiel Logikfehler: Wo steckt der Fehler?

Listing 1: cbugs/bugEx_1.c

```
1 a=b; /* this is a bug
2 c=d; /* c=d will never happen */
```

Beispiel Logikfehler: Nichtbeendeter Kommentar

Listing 2: cbugs/bugEx_1.c

```
1 a=b; /* this is a bug
2 c=d; /* c=d will never happen */
```

- Fehler eigentlich veraltet.
- Wird verhindert durch Editor mit C-Highlighting.

Beispiel Logikfehler: Wo steckt der Fehler?

Listing 3: cbugs/bugEx_2.c

```
1 int x = 5;
2 if ( x = 6 )
3   printf("x equals 6\n");
```

Beispiel Logikfehler: Zuweisung statt Vergleich

Listing 4: cbugs/bugEx_2.c

```
1 int x = 5;
2 if ( x = 6 )
3   printf("x equals 6\n");
```

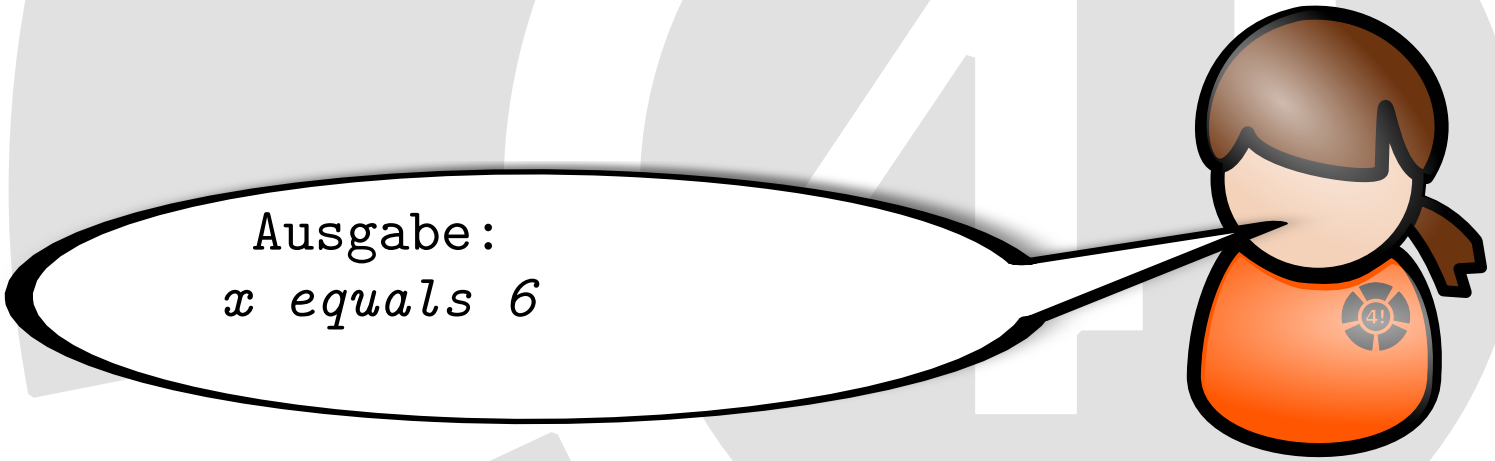
- Häufiger Fehler.
- Die Zuweisung ist immer wahr.

Beispiel Logikfehler: Zuweisung statt Vergleich

Listing 5: cbugs/bugEx_2.c

```
1 int x = 5;
2 if ( x = 6 )
3   printf("x equals 6\n");
```

- Häufiger Fehler.
- Die Zuweisung ist immer wahr.



Beispiel Logikfehler: Wo steckt der Fehler?

Listing 6: cbugs/bugEx_3.c

```
1 int foo(int a){
2   if (a) return (1);
3 }
```

Beispiel Logikfehler: Kein Return

Listing 7: cbugs/bugEx_3.c

```
1 int foo(int a){
2   if (a) return (1);
3 }
```

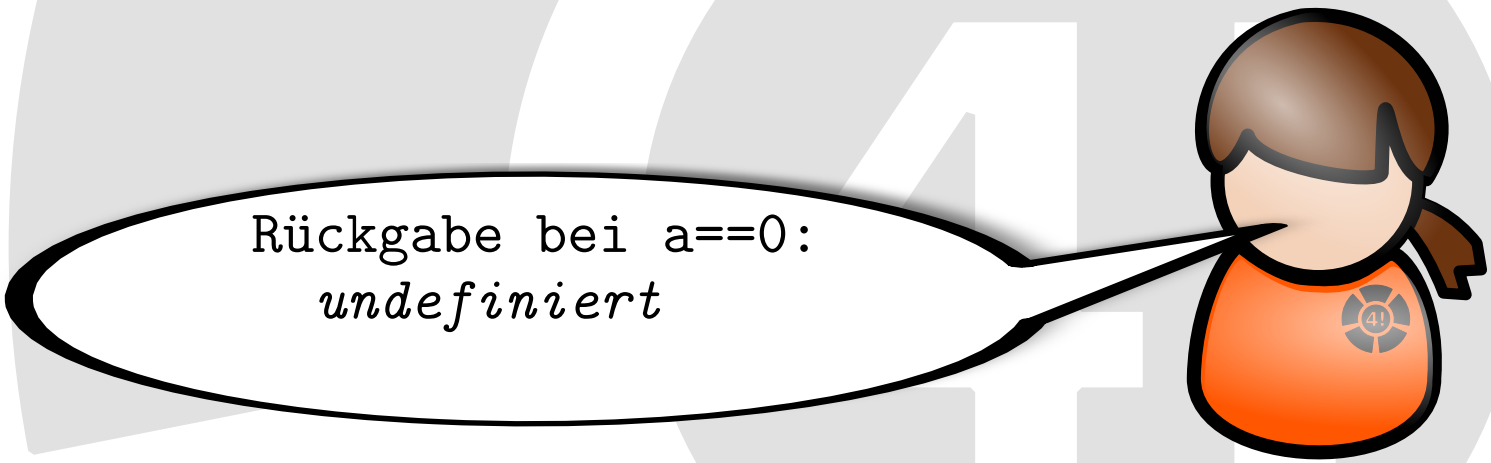
- Hier existieren Bedingungen, bei denen kein return erreicht wird.

Beispiel Logikfehler: Kein Return

Listing 8: cbugs/bugEx_3.c

```
1 int foo(int a){
2   if (a) return (1);
3 }
```

- Hier existieren Bedingungen, bei denen kein return erreicht wird.



Beispiel Logikfehler: Wo steckt der Fehler?

Listing 9: cbugs/bugEx_4.c

```
1   if (a == 1)
2     if (b == 1)
3       printf("a ist 1");
4   else
5     printf("a ist nicht 1");
```

Beispiel Logikfehler: Dangling else

Listing 10: cbugs/bugEx_4.c

```
1  if (a == 1)
2    if (b == 1)
3      printf("a ist 1");
4  else
5    printf("a ist nicht 1");
```

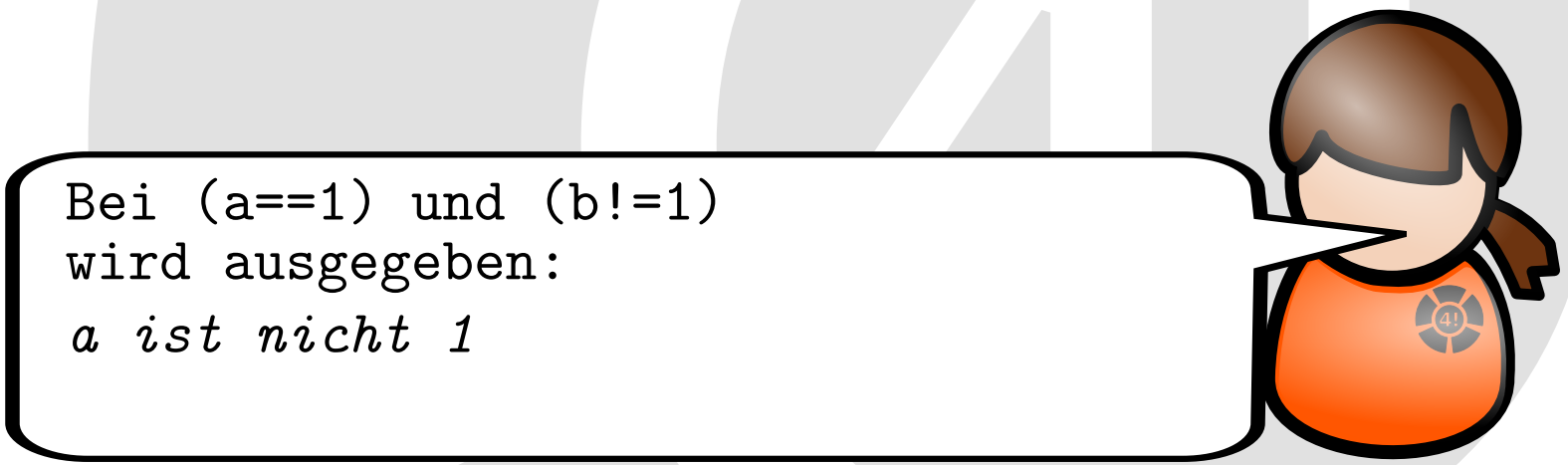
- Das else gehört immer zum letzten if.
- Besser: Klammern setzen.

Beispiel Logikfehler: Dangling else

Listing 11: cbugs/bugEx_4.c

```
1  if (a == 1)
2    if (b == 1)
3      printf("a ist 1");
4  else
5    printf("a ist nicht 1");
```

- Das else gehört immer zum letzten if.
- Besser: Klammern setzen.



Beispiel Logikfehler: Wo steckt der Fehler?

Listing 12: cbugs/bugEx_5a.c

```
1     if( ... )
2         fct_one();
3     else
4         printf( "Calling fct_two()" );
5         fct_two();
```

Beispiel Logikfehler: Fehlende Klammern

Listing 13: cbugs/bugEx_5.c

```
1     if( ... )
2         fct_one();
3     else
4         printf( "Calling fct_two()" );
5           /* oops! the else stops here */
6         fct_two();
7           /* oops! fct_two() is always executed */
```

- Auch hier:
- Klammern notwendig.

Beispiel Logikfehler: Wo steckt der Fehler?

Listing 14: cbugs/bugEx_6.c

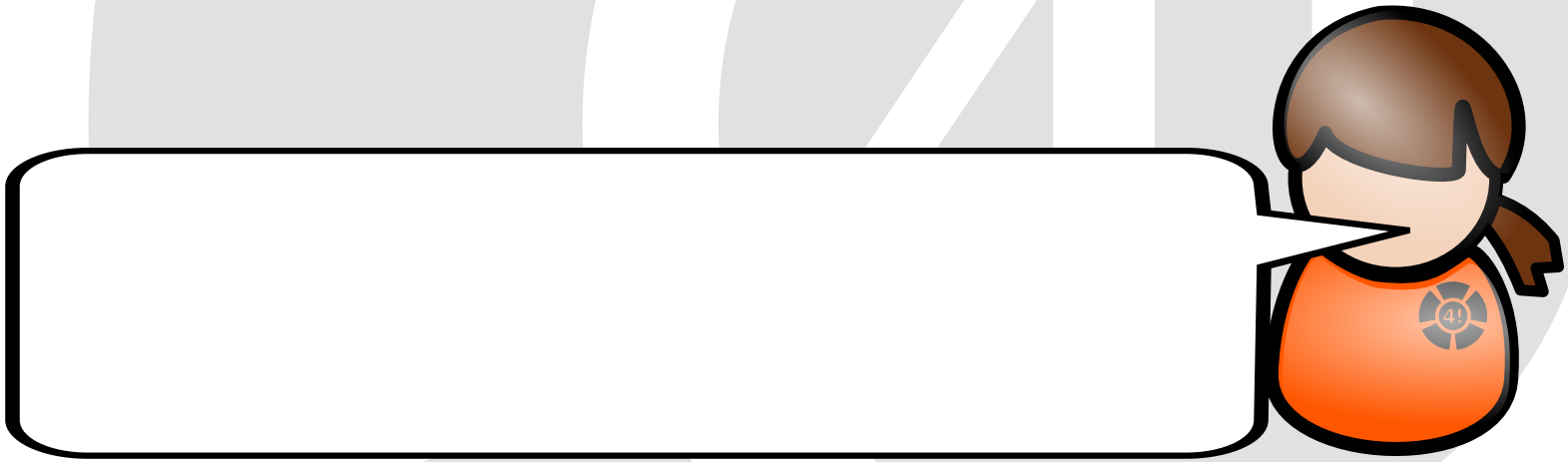
```
1 int x = 2;
2 switch(x) {
3 case 2:
4   printf("Two\n");
5 case 3:
6   printf("Three\n");
7 }
```

Beispiel Logikfehler: Fehlendes break

Listing 15: cbugs/bugEx_6.c

```
1 int x = 2;
2 switch(x) {
3 case 2:
4   printf("Two\n");
5 case 3:
6   printf("Three\n");
7 }
```

- Ohne break werden auch alle nachfolgenden cases ausgeführt.

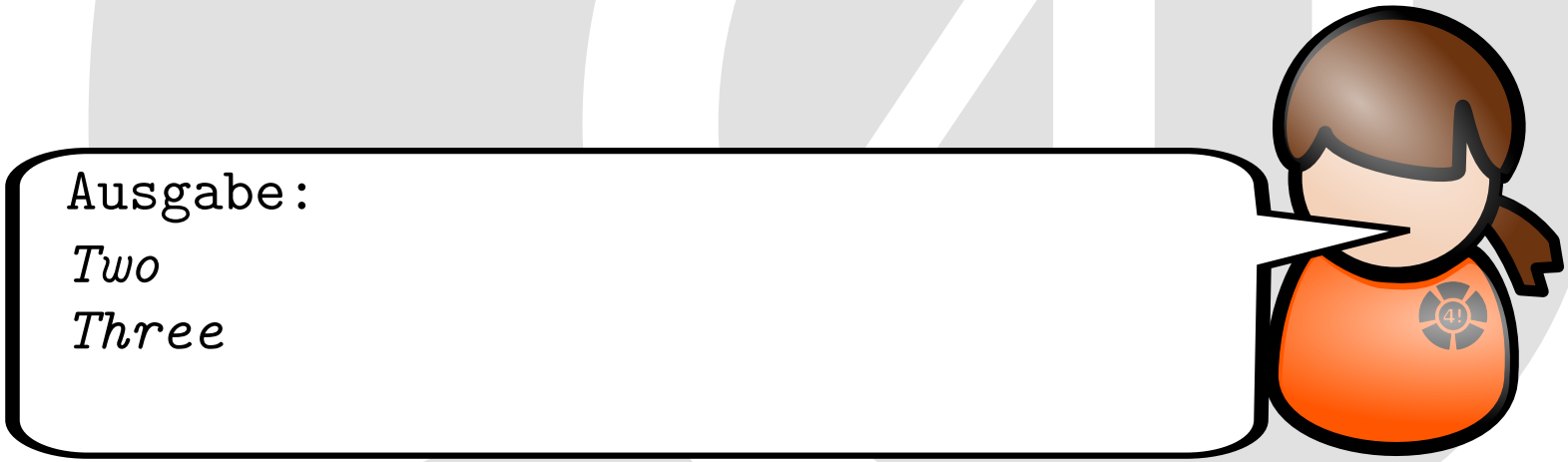


Beispiel Logikfehler: Fehlendes break

Listing 16: cbugs/bugEx_6.c

```
1 int x = 2;
2 switch(x) {
3 case 2:
4   printf("Two\n");
5 case 3:
6   printf("Three\n");
7 }
```

- Ohne break werden auch alle nachfolgenden cases ausgeführt.



Beispiel Logikfehler: Wo steckt der Fehler?

Listing 17: cbugs/bugEx_7.c

```
1 int main ()
2 {
3   unsigned char var = 255;
4   printf ("Wert = %d\n", var);
5   var++;
6   printf ("Wert = %d\n", var);
7   printf ("Wert = %d\n", 10/var);
8   return 0;
9 }
```

Beispiel Logikfehler: Overflow

Listing 18: cbugs/bugEx_7.c

```c
int main ()
{
   unsigned char var = 255;
   printf ("Wert = %d\n", var);
   var++;
   printf ("Wert = %d\n", var);
   printf ("Wert = %d\n", 10/var);
   return 0;
}
```

- Der Wertebereich von unsigned char ist 0 - 255.

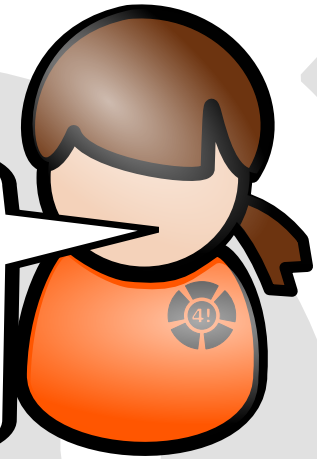
Beispiel Logikfehler: Overflow

Ausgabe:

Beispiel Logikfehler: Overflow

Ausgabe:

```
Wert = 255
Wert = 0
floating point exception (core dumped)
```



Beispiel Logikfehler: Wo steckt der Fehler?

Listing 19: cbugs/bugEx_8a.c

```
1  char s = 127;
2  unsigned char u = 127;
3  s++;
4
5  if (s<u)
6   { /* is this done? */ }
7  if(s>127)
8   { /* is this done? */ }
9  if(u<0)
10  { /* is this done? */ }
```

Beispiel Logikfehler: Overflow

Listing 20: cbugs/bugEx_8.c

```
1  char s = 127;
2  unsigned char u = 127;
3  s++;
4  /* the result is a negative number! */
5  if (s<u)
6   { /* true!*/ }
7  if(s>127)
8   { /* this can never be true */  }
9  if(u<0)
10  {  /* this can never be true*/  }
```

- Ein char ist signed!

Beispiel Logikfehler: Was passiert hier?

Listing 21: cbugs/bugEx_9.c

```
1 int x;
2 char st[31];
3
4 printf("Enter an integer: ");
5 scanf("%d", &x);
6 printf("Enter a line of text: ");
7 fgets(st, 31, stdin);
```

Beispiel Logikfehler: Falsche Eingabe

Listing 22: cbugs/bugEx_9.c

```
1 int x;
2 char st[31];
3
4 printf("Enter an integer: ");
5 scanf("%d", &x);
6 printf("Enter a line of text: ");
7 fgets(st, 31, stdin);
```

- fgets wird übersprungen.

Fehlerbehebung: Löschen des Input-Buffers

Funktion um den Input-Buffer zu löschen:

Listing 23: cbugs/bugEx_9.c

```
1 void dump_line( FILE *fp )
2 {
3   int ch;
4   while(((ch = fgetc(fp)) != EOF) && ( ch != '\n'));
5 }
```

Fehlerbehebung: Löschen des Input-Buffers

Anwendung:

Listing 24: cbugs/bugEx_9.c

```
1 int x;
2 char st[31];
3 printf("Enter an integer: ");
4 scanf("%d", &x);
5 dump_line(stdin);
6 printf("Enter a line of text: ");
7 fgets(st, 31, stdin);
```

Beispiel Logikfehler: Wo steckt der Fehler?

- Ein Makro wird definiert:

Listing 25: cbugs/bugEx_10.c

```
1 #define PI_PLUS_ONE 3.14 + 1
```

- So wird das Makro verwendet:

Listing 26: cbugs/bugEx_10.c

```
1 x=PI_PLUS_ONE *5;
```

- Was passiert?

Beispiel Logikfehler: unsaubere Makrodefinition

- Mit dem folgenden Makro:

Listing 27: cbugs/bugEx_10.c

```
1 #define PI_PLUS_ONE 3.14 + 1
```

- wird:

Listing 28: cbugs/bugEx_10.c

```
1 x=PI_PLUS_ONE *5;
```

- zu:

Listing 29: cbugs/bugEx_10.c

```
1 x= 3.14 + 1 * 5;
```

Beispiel Logikfehler: Welche Werte stehen in diesem Array?

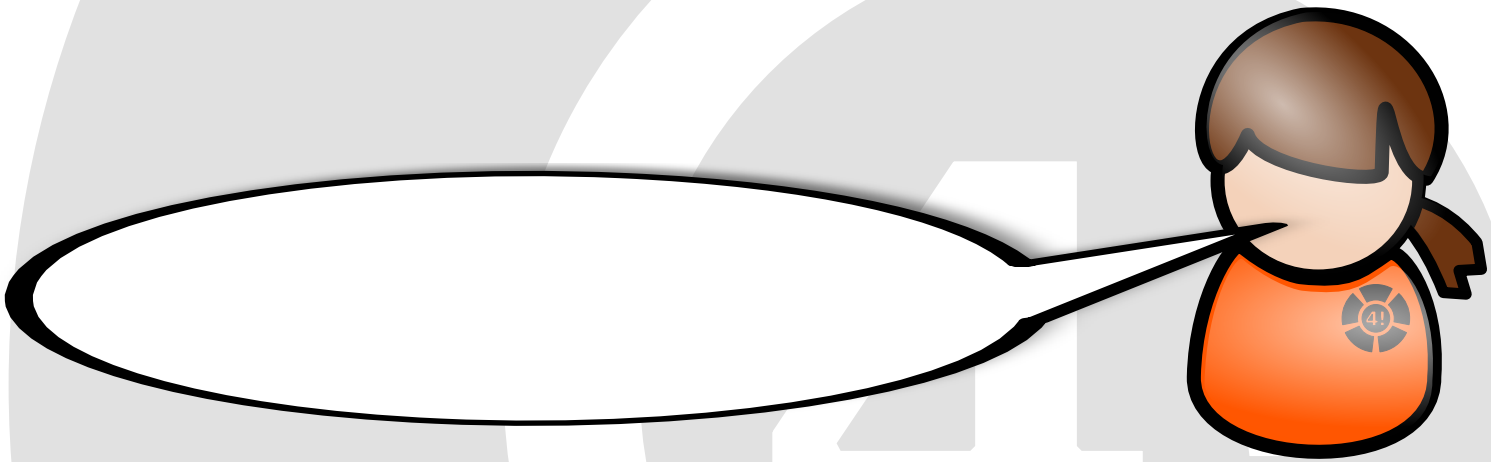
Listing 30: cbugs/bugEx_11.c

```
1 int numbers[] = { 001, 010, 014 };
```

Beispiel Logikfehler: Oktalzahlen

Listing 31: cbugs/bugEx_11.c

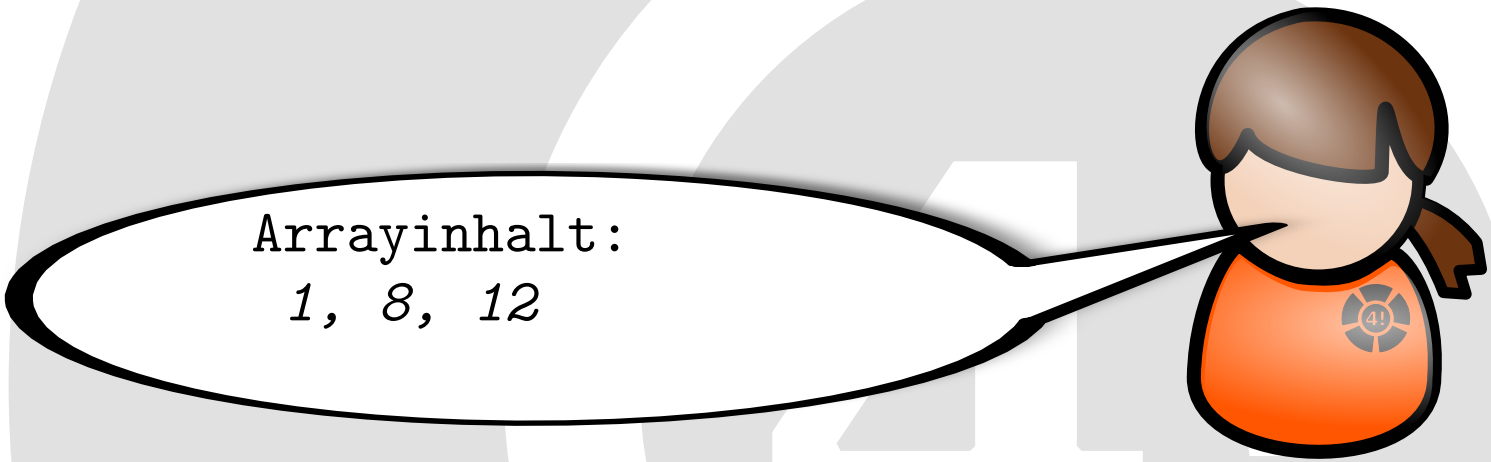
```
1 int numbers[] = { 001, 010, 014 };
```



Beispiel Logikfehler: Oktalzahlen

Listing 32: cbugs/bugEx_11.c

```
1 int numbers[] = { 001, 010, 014 };
```



Beispiel Logikfehler: Was passiert hier?

Listing 33: cbugs/bugEx_12.c

```
1 if (  a  =!  b)
```

Beispiel Logikfehler: Eine weitere Zuweisung

Listing 34: cbugs/bugEx_12.c

```
1 if(  a  =!  b)
```

Beispiel Logikfehler: Eine weitere Zuweisung

Listing 35: cbugs/bugEx_12.c

```
1 if(  a  =!  b)
```

Inhaltsverzeichnis

1. Fehlerarten
   - Syntaxfehler
   - Logikfehler
   - Laufzeitfehler
2. Compilerwarnungen
3. Assertions
4. GDB
   - Was ist ein Debugger?
   - Der GNU Debugger GDB
   - Wichtigsten Kommandos des GDB
   - First Steps

Laufzeitfehler

Mögliche Ursachen für einen Programmabsturz:

- Zugriff auf nichtreservierten Speicher.
- Beschreiben eines Arrays über die Grenzen hinaus (Folgefehler: Stack wird überschrieben)
- Speicherzugriff im falschen Scope (Rückgabe eines Zeigers auf lokale Variable)
- Endlosschleifen

Beispiel Laufzeitfehler: Was wird ausgegeben?

Listing 36: cbugs/bugEx_13.c

```c
int main(int argc, char * argv[]){
  int temp=5;
  int *ptr = &temp;

  printf("Wert: %d\n", ptr);
  printf("Wert: %d\n", &ptr);
  printf("Wert: %d\n", *ptr);

  ptr = 5;
  printf("Wert: %d\n", *ptr);

  return 0;
}
```

Beispiel Laufzeitfehler: Pointer/Integer

Ausgabe:

Beispiel Laufzeitfehler: Pointer/Integer

Ausgabe:

Weiter: *segmentation fault (core dumped)*

Beispiel Laufzeitfehler: Was passiert hier?

Listing 37: cbugs/bugEx_14.c

```
1 int main(int argc, char * argv[]){
2   int i;
3       int a[8];
4       for(i=0; i <= sizeof(a) ; i++)
5             a[i]=0;
6
7   return 0;
8 }
```

Beispiel Laufzeitfehler: Zu viele Iterationen

Listing 38: cbugs/bugEx_14.c

```
1 int main(int argc, char * argv[]){
2   int i;
3       int a[8];
4       for(i=0; i <= sizeof(a) ; i++)
5             a[i]=0;
6
7   return 0;
8 }
```

Korrektur:

Listing 39: cbugs/bugEx_14.c

```
1 for(i=0; i < sizeof(a)/sizeof(int) ; i++)
```

Beispiel Laufzeitfehler: Was passiert hier?

Listing 40: cbugs/bugEx_15.c

```
1 char * copy_str = malloc(strlen(orig_str));
2 strcpy(copy_str, orig_str);
```

Beispiel Laufzeitfehler: Fehlendes Stringendezeichen

Häufiger Fehler. Speicherplatz für Stringendezeichen fehlt.

Listing 41: cbugs/bugEx_15.c

```
1 char * copy_str = malloc(strlen(orig_str));
2 strcpy(copy_str, orig_str);
```

Korrigiert:

Listing 42: cbugs/bugEx_15.c

```
1 char * copy_str = malloc(strlen(orig_str) + 1);
```

Beispiel Laufzeitfehler: Was passiert hier?

Listing 43: cbugs/bugEx_16.c

```
1  char *f() {
2      char result[80];
3      sprintf(result,"Lokal auf dem Stack der Funktion f");
4      return(result);
5   }
6
7  int g()
8  {
9      char *p;
10     p = f();
11     printf("f() returns: %s\n",p);
12 }
```

Beispiel Laufzeitfehler: Zeiger auf lokale Variable

Listing 44: cbugs/bugEx_16.c

```
1  char *f() {
2      char result[80];
3      sprintf(result,"Lokal auf dem Stack der Funktion f");
4      return(result);
5   }
6
7  int g()
8  {
9      char *p;
10     p = f();
11     printf("f() returns: %s\n",p);
12 }
```

Problematik bei der Fehlersuche: Das Programm scheint zu funktionieren, falls der Speicherbereich im Stack noch nicht überschrieben wurde.

Beispiel Laufzeitfehler: Was passiert hier?

Listing 45: cbugs/bugEx_17.c

```
1 int x = 5;
2 while( x > 0 );
3   x--;
```

Beispiel Laufzeitfehler: Eine Endlosschleife

Listing 46: cbugs/bugEx_17.c

```
1 int x = 5;
2 while( x > 0 );
3   x--;
```

Inhaltsverzeichnis

1. Fehlerarten
   - Syntaxfehler
   - Logikfehler
   - Laufzeitfehler
2. Compilerwarnungen
3. Assertions
4. GDB
   - Was ist ein Debugger?
   - Der GNU Debugger GDB
   - Wichtigsten Kommandos des GDB
   - First Steps

Aktivierung der Anzeige von Compiler-Warnungen

```
gcc -Wall -Wextra -o beispiel beispiel.c
```

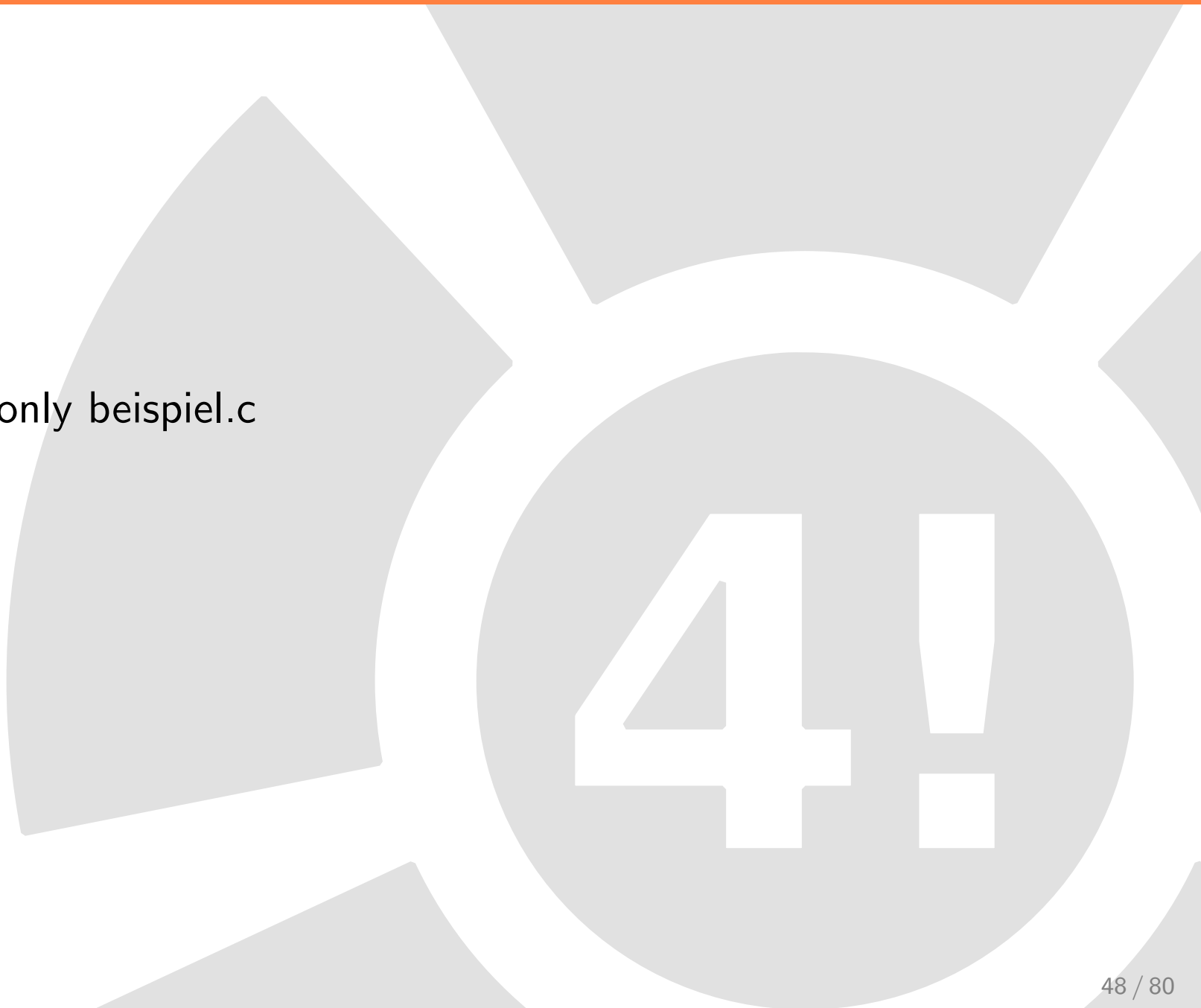
Aktivierung Anzeige der Compiler-Warnungen

ANGEZEIGTE WARNUNGEN:

- Wahrscheinlicher Type mismatch zwischen Integer und Pointer
- Zuweisung anstatt Boolscher Ausdruck
- Nichtinitialisierte Variable
- Unused Variable
- Leere Bodies von Schleifen und If-Abfragen
- Falscher oder Fehlender Rückgabetyp einer Funktion
- Dangling else

Syntaxüberprüfung

gcc -fsyntax-only beispiel.c



Beispiel Compilerwarnungen: Selbst versuchen!

Listing 47: cbugs/warningEx.c

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

int main (int argc, char* argv[]){
   int a,b;
   a=b;
   if (a=45);
   printf("anything\n");
}
```

Inhaltsverzeichnis

1. Fehlerarten
   - Syntaxfehler
   - Logikfehler
   - Laufzeitfehler
2. Compilerwarnungen
3. Assertions
4. GDB
   - Was ist ein Debugger?
   - Der GNU Debugger GDB
   - Wichtigsten Kommandos des GDB
   - First Steps

Assertions: Applikationen

- Prüfung der Übergabeparameter am Anfang einer Funktion (pre-condition)
- Prüfung des Rückgabewertes (post-condition)
- Prüfung von Invarianten

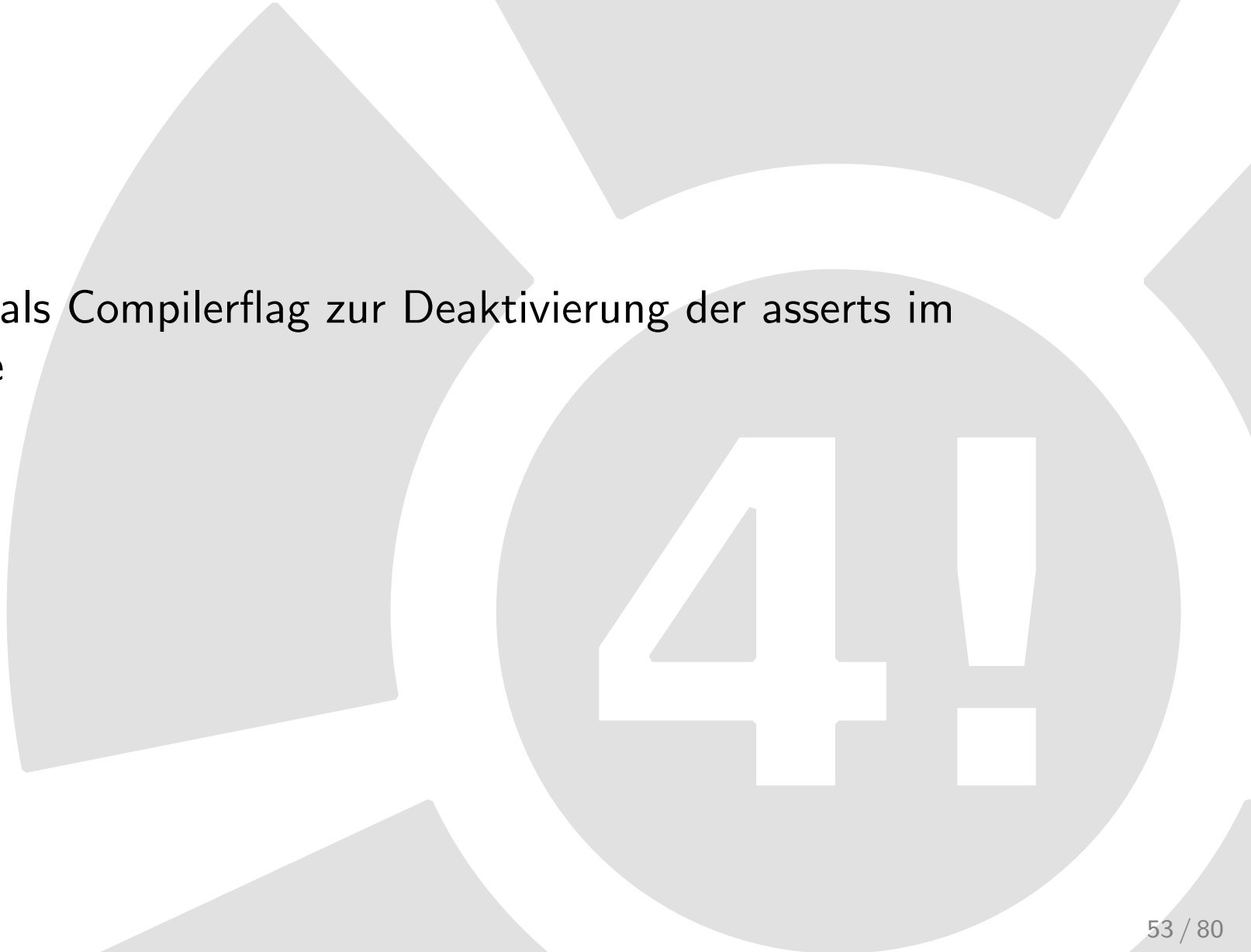
Assertions: Prüfung des Übergabeparameters

Listing 48: cbugs/assert.c

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

int mystrlen (char * pszStr){
    assert(pszStr!=NULL);

    printf("alles ok");
    return 0;
}

int main(void){
  char * p = {"Hallo"};
  mystrlen(NULL);
  return 0;
}
```

Assertions: Deaktivierung

-DNDEBUG als Compilerflag zur Deaktivierung der asserts im Release-Code

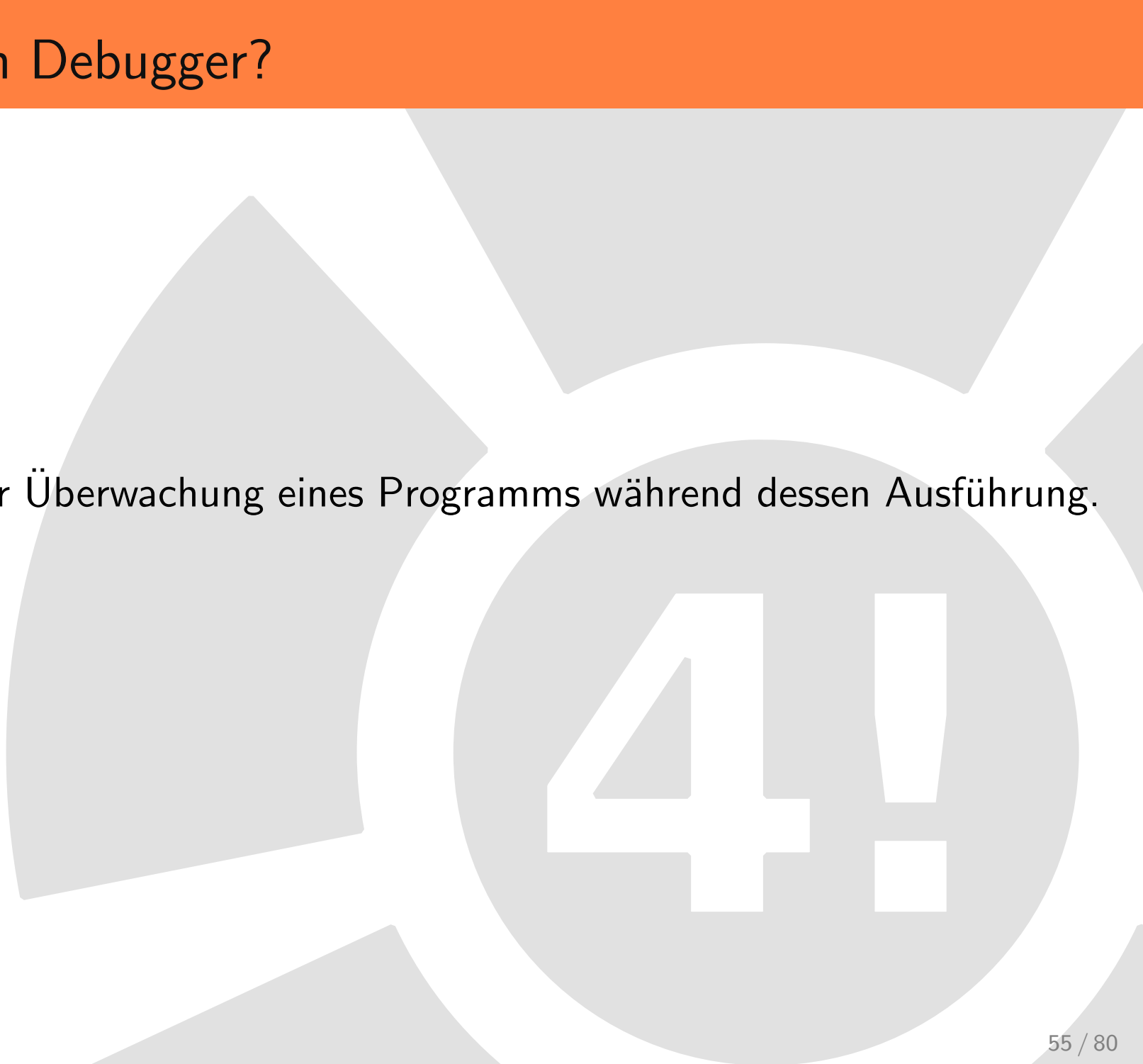


Inhaltsverzeichnis

1. Fehlerarten
   - Syntaxfehler
   - Logikfehler
   - Laufzeitfehler
2. Compilerwarnungen
3. Assertions
4. GDB
   - Was ist ein Debugger?
   - Der GNU Debugger GDB
   - Wichtigsten Kommandos des GDB
   - First Steps

Was ist ein Debugger?

Werkzeug zur Überwachung eines Programms während dessen Ausführung.



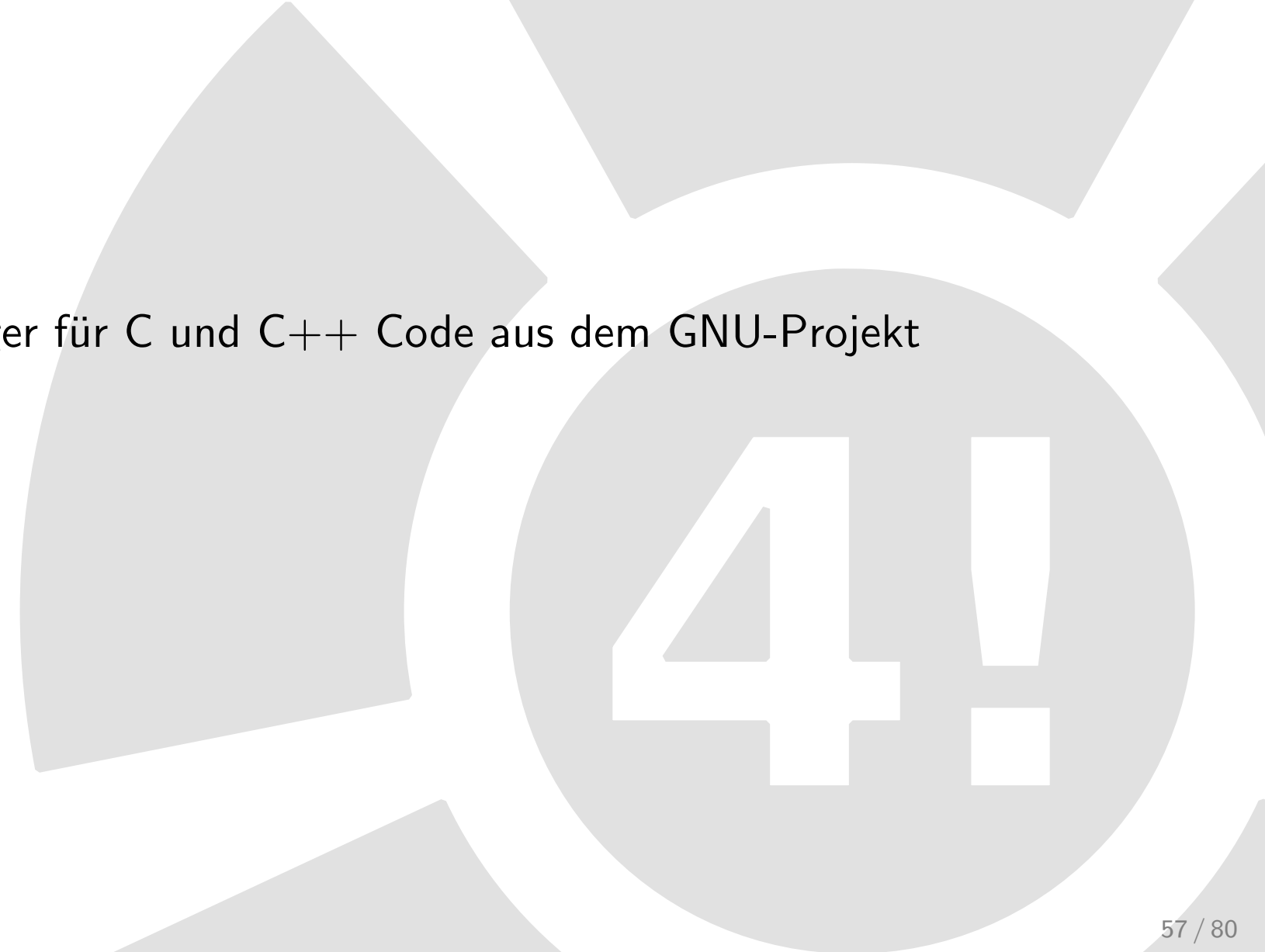
Was ist ein Debugger?

Seine Features:

- Breakpoint setzen, Programm definiert anhalten
- Schritt-für-Schritt-Ausführung
- Inhalt von Variablen überwachen, anzeigen und verändern
- Stack anzeigen
- Anzeige wo genau der Crash verursacht wurde

Der GNU Debugger: GDB

- Debugger für C und C++ Code aus dem GNU-Projekt



Wichtigsten Kommandos des GDB

- file : laden des exefiles und der zugehörigen Debugging-Symbole
- run<args>: starten des Programms, Übergabe der Parameter
- break <lineNr>: Breakpoint in Programmzeile setzen
- break <function>: Breakpoint bei Funktionsaufruf setzen
- break <function/lineNr> <condition>: bedingter break
- tbreak <function/lineNr>: temporärer break, einmalig
- watch <var>: Watchpoint setzen
- call <function>: Aufruf einer Bibliotheksfunktion oder im Projekt definierten Funktion
- print <expr>: Ausgabe eines Variableninhalts oder eines gesamten Ausdrucks
- list : Ausgabe der nächsten 10 Zeilen des Quellcodes
- list <lineNr>: Ausgabe des Quellcodes im Umfeld der lineNr

Wichtigsten Kommandos des GDB

- cont: Fortsetzung der Ausführung des Programms, bis zum nächsten break
- next: Ausführung des nächsten Befehls (step over)
- step: Ausführung des nächsten Funktion (step into)
- finish : Beendigung der aktuellen Funktion (step out)
- bt oder where: Anzeige des aktuellen Stacks (backtrace)
- up: Geht zur Aufrufenden Funktion den Stack hoch
- down: Gegenteil von up
- delete <nr>: Löscht den breakpoint, falls seine nr angegeben ist. Ohne nr werden alle Breakpoints gelöscht.
- disable <nr>: Deaktiviert den nr. Breakpoint
- info breakpoints: Anzeige aller Breakpoints
- kill : Beendet die Ausführung des Programms
- quit: Beendet GDB

First Steps - Ein Beispiel

UNSER BEISPIEL:

Listing 49: cbugs/debug.c

```
1  #include <stdio.h>
2
3  int main()
4  {
5     int i, num, j;
6           printf ("Enter the number: ");
7     scanf ("%d", &num );
8
9     for (i=1; i<num; i++)
10       j=j*i;
11
12    printf("The factorial of %d is %d\n",num,j);
13
14          return 0;
15 }
```

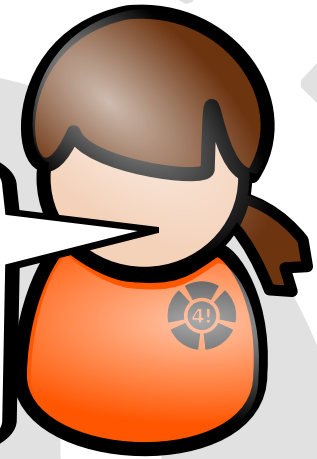
Das Verhalten unseres Beispiels

AUSGABE für die Testzahl 4:

Das Verhalten unseres Beispiels

AUSGABE für die Testzahl 4:

```
 ./debug
Enter the number: 4
The factorial of 4 is 116688563
```



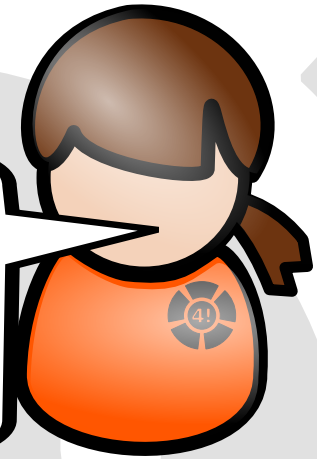
1. Schritt: Compilieren mit -g

Beim Kompilieren muss das Compilerflag -g gesetzt werden:

1. Schritt: Compilieren mit -g

Beim Kompilieren muss das Compilerflag -g gesetzt werden:

```
gcc -g -Wall -Wextra -o debug debug.c
```

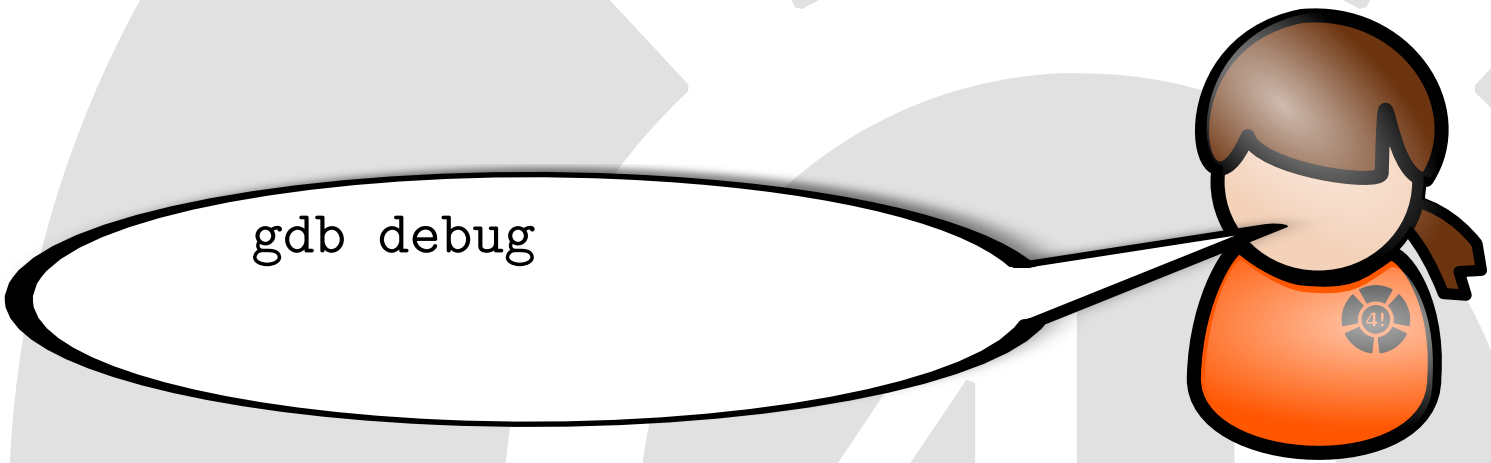


2. Schritt: GDB starten

Wir starten den GDB und laden unsere Datei:

2. Schritt: GDB starten

Wir starten den GDB und laden unsere Datei:



3. Schritt: Setzen eines Breakpoints

Wir setzten einen Breakpoint auf die for-Schleife

3. Schritt: Setzen eines Breakpoints

Wir setzten einen Breakpoint auf die for-Schleife

Anzeige alle Breakpoints: `info breakpoints`

4. Schritt: Starten des Programms

Wir starten das Programm:

4. Schritt: Starten des Programms

Wir starten das Programm:

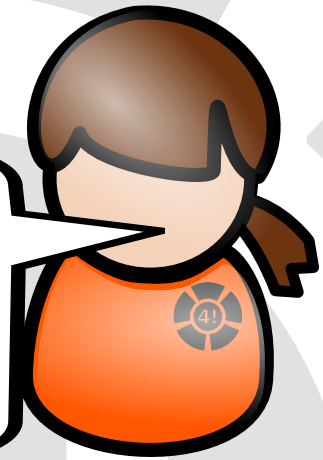
5. Schritt: Werte der Variablen ausgeben

Mit dem Befehl print lassen wir uns die Werte für j, i, num ausgeben:

5. Schritt: Werte der Variablen ausgeben

Mit dem Befehl print lassen wir uns die Werte für j, i, num ausgeben:

```
print i, i=1
print j, j = 12359452
```



- Wert für j ist falsch! j wird nicht initialisiert.

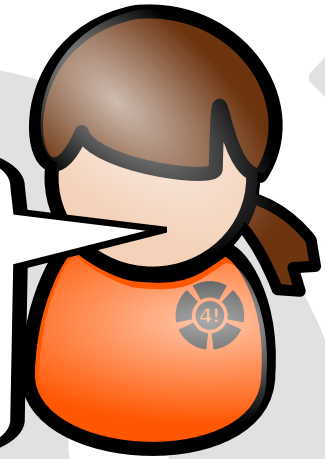
6. Schritt: Verändern des Variablenwertes

Wir setzen die Variable j manuell auf 1.

6. Schritt: Verändern des Variablenwertes

Wir setzen die Variable j manuell auf 1.

```
set variable j = 1
print j, j=1
```



7. Schritt: Weitere Ausführung des Programms

Weiteres Ausführen des Programms mit:

7. Schritt: Weitere Ausführung des Programms

Weiteres Ausführen des Programms mit:

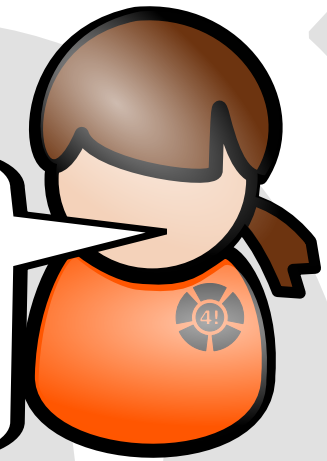
Das neue Verhalten des Beispiels

Wir korrigieren das Programm und initialisieren die Variable j. Kompilieren neu und erhalten folgendes Verhalten:

Das neue Verhalten des Beispiels

Wir korrigieren das Programm und initialisieren die Variable j. Kompilieren neu und erhalten folgendes Verhalten:

```
 ./debug
Enter the number: 4
The factorial of 4 is 6
```



8. Schritt: Setzen eines Watchpoints

Zum Setzen eines Watchpoints auf j sind folgende Schritte notwendig:

- Setzen eines Breakpoints direkt nach Initialisierung der Variablen j.
- Ausführen des Programms mit run.
- Setzen des Watchpoints mit watch j
- Fortsetzen des Programms mit cont

Werte der Variablen j

Durch das Setzen des Watchpoints sehen wir, dass j folgende Werte annimmt:

Werte der Variablen j

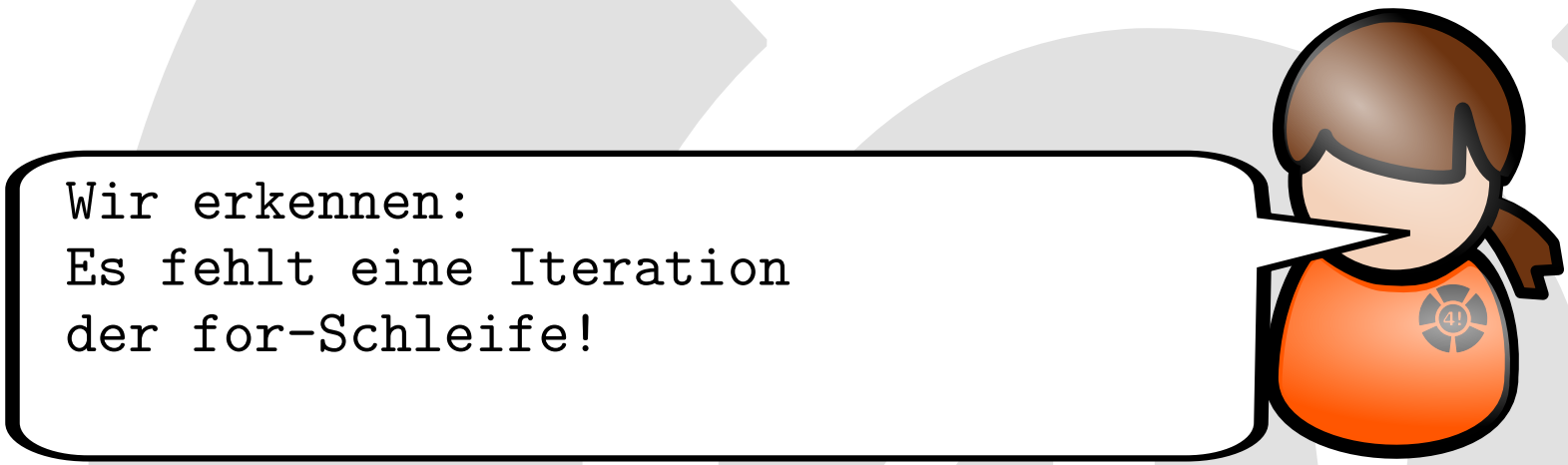
Durch das Setzen des Watchpoints sehen wir, dass j folgende Werte annimmt:

9. Schritt: Kontrolle des Programm-Codes

Mit list können wir uns immer 10 Zeilen des Programm-Codes anzeigen lassen.

9. Schritt: Kontrolle des Programm-Codes

Mit list können wir uns immer 10 Zeilen des Programm-Codes anzeigen lassen.



Korrektur

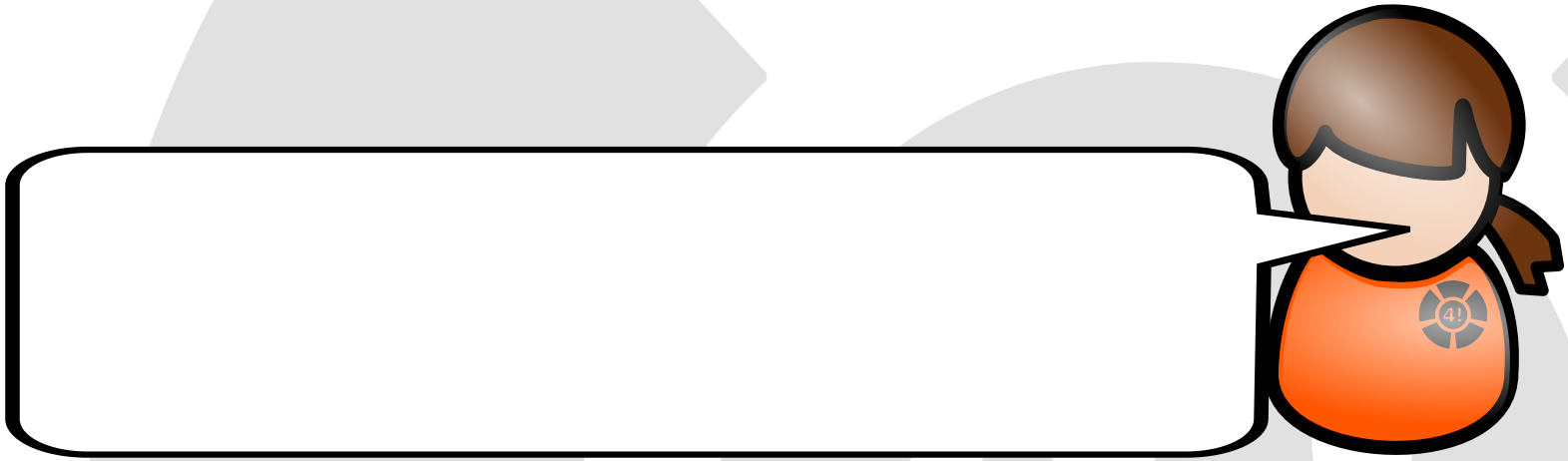
Wir korrigieren den Programmcode zu:

Listing 50: cbugs/debug.c

```
1 for (i=1; i<=num; i++)
2         j=j*i;
```

Korrigiertes Beispiel

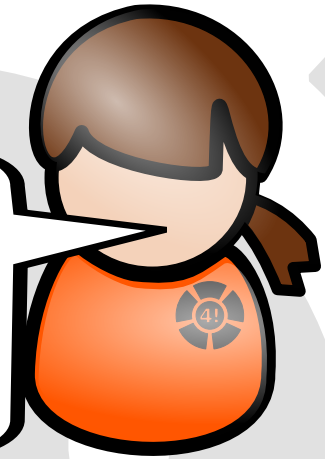
Und endlich bekommen wir das korrekte Ergebnis:



Korrigiertes Beispiel

Und endlich bekommen wir das korrekte Ergebnis:

```
 ./debug
Enter the number: 4
The factorial of 4 is 24
```



Wichtig für weitere Beispiele:

Nicht vergessen! Wichtige Befehle zur Beobachtung des Stacks:

- bt oder where: Anzeige des aktuellen Stacks (backtrace)
- up: Geht zur Aufrufenden Funktion den Stack hoch
- down: Gegenteil von up

Weitere Beispiele zum Debuggen:

Listing 51: cbugs/debEx_1.c

```
1  // http://www.unknownroad.com/rtfm/gdbtut/gdbsegfault.
       html
2  #include <stdio.h>
3  #include <stdlib.h>
4
5  int main(int argc, char **argv)
6  {
7    char *buf;
8
9    buf = malloc(1<<31);
10
11   fgets(buf, 1024, stdin);
12   printf("%s\n", buf);
13
14   return 1;
15 }
```

Weitere Beispiele zum Debuggen:

Listing 52: cbugs/bubbleSort.c

```
1 // http://wiki.freitagsrunde.org/Ckurs2009/
    BubbleSortDebug
```

Weitere wichtige Tools zur sicheren Fehlererkennung

Zum weiteren Selbststudium:

- splint: http://www.splint.org/
- valgrind: http://www.valgrind.org/

Quellen

Listing 53: cbugs/Sources

```
1 www.drpaulcarter.com/cs/common-c-errors.php
2
3 www.cplusplus.com/reference/clibrary/cassert/assert/
4
5 www.thegeekstuff.com/2010/03/debug-c-program-using-gdb/
6
7 pronix.linuxdelta.de/C/standard_C/c_programmierung_6.
    shtml
```

Danke