

# Fehlererkennung

C - Kurs 2012

13. September 2012

4!

This work is licensed under a Creative Commons AttributionNonComercialShareAlike 3.0 License.

- 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



4!

- 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



4!

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



4!

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



4!

# 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 */
```

4!

# 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.

4!

# 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");
```

4!

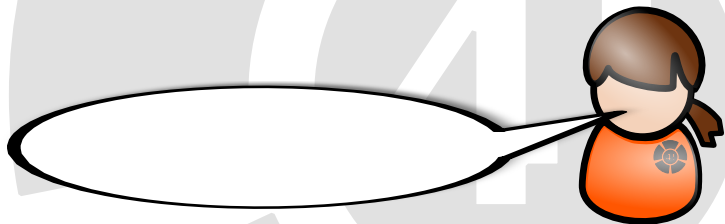


# Beispiel Logikfehler: Zuweisung statt Vergleich

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.



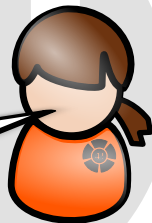
# Beispiel Logikfehler: Zuweisung statt Vergleich

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.

Ausgabe:  
*x equals 6*



## Beispiel Logikfehler: Wo steckt der Fehler?

Listing 6: cbugs/bugEx\_3.c

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

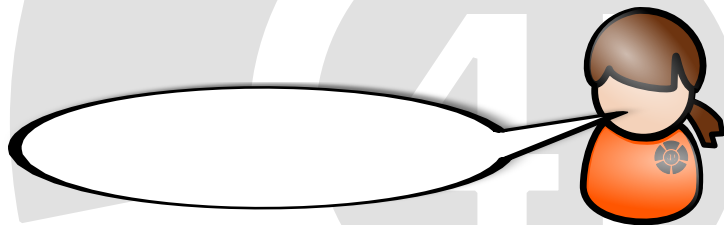
4!

# 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.

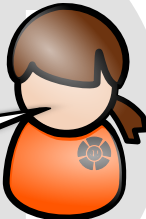


# Beispiel Logikfehler: Kein Return

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.



Rückgabe bei  $a==0$ :  
*undefiniert*

## Beispiel Logikfehler: Wo steckt der Fehler?

Listing 9: cbugs/bugEx\_4.c

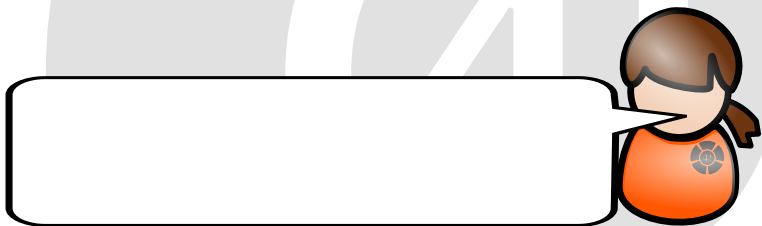
```
1  if (a == 1)
2      if (b == 1)
3          printf("a_ist_1");
4  else
5      printf("a_ist_nicht_1");
```

## 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.

Bei `(a==1)` und `(b!=1)`  
wird ausgegeben:  
*a ist nicht 1*





## 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

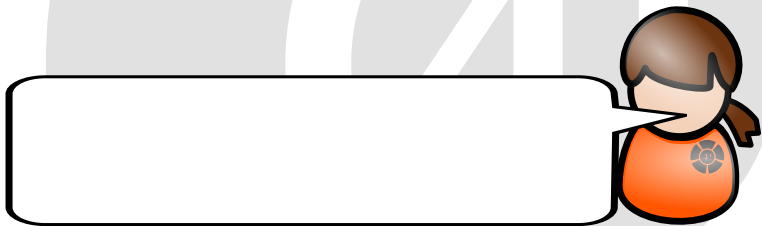
```
1 int x = 2;
2 switch(x) {
3 case 2:
4     printf("Two\n");
5 case 3:
6     printf("Three\n");
7 }
```

# 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

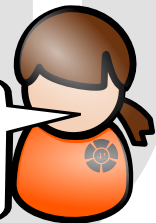
```
1 int x = 2;
2 switch(x) {
3 case 2:
4     printf("Two\n");
5 case 3:
6     printf("Three\n");
7 }
```

- Ohne `break` werden auch alle nachfolgenden cases ausgeführt.

Ausgabe:

*Two*

*Three*



## 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

```
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 }
```

- 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);
```

Es wird gelegentlich auch `fflush(stdin)` benutzt, um den Stream zu leeren. Dies ist nach Standard nicht definiert, aber in der Linux-Implementierung funktioniert es.

# 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 };
```

4!

# Beispiel Logikfehler: Oktalzahlen

Listing 31: cbugs/bugEx\_11.c

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

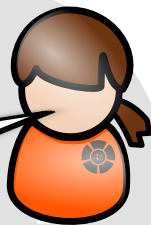


# Beispiel Logikfehler: Oktalzahlen

Listing 32: cbugs/bugEx\_11.c

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

Arrayinhalt:  
1, 8, 12



## Beispiel Logikfehler: Was passiert hier?

Listing 33: cbugs/bugEx\_12.c

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

4!

# 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)
```

(a!=b)  
*ungleich*  
(a!=b)



- 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



4!



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



4!

## Beispiel Laufzeitfehler: Was wird ausgegeben?

Listing 36: cbugs/bugEx\_13.c

```
1 int main(int argc, char * argv[]) {  
2     int temp=5;  
3     int *ptr = &temp;  
4  
5     printf("Wert: %d\n", ptr);  
6     printf("Wert: %d\n", &ptr);  
7     printf("Wert: %d\n", *ptr);  
8  
9     ptr = 5;  
10    printf("Wert: %d\n", *ptr);  
11  
12    return 0;  
13 }
```

# Beispiel Laufzeitfehler: Pointer/Integer

Ausgabe:



# Beispiel Laufzeitfehler: Pointer/Integer

Ausgabe:

```
Wert: 2280756  
Wert: 2280752  
Wert: 5
```



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--;
```

- 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



4!

# Aktivierung der Anzeige von Compiler-Warnungen

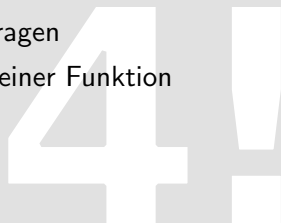
```
gcc -Wall -Wextra -o beispiel beispiel.c
```



4!

## ANGEZEIGTE WARNUNGEN:

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

A large, light gray graphic consisting of a circle containing a white number '4' followed by a white exclamation mark '!', positioned on the right side of the slide.

`gcc -fsyntax-only beispiel.c`



4!



# Beispiel Compilerwarnungen: Selbst versuchen!

Listing 47: cbugs/warningEx.c

```
1 #include <stdio.h>
2 #include <assert.h>
3
4 int main (int argc, char* argv []) {
5     int a,b;
6     a=b;
7     if (a=45);
8     printf(" anything\n");
9 }
```

- 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



4!

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



4!

# Assertions: Prüfung des Übergabeparameters

Listing 48: cbugs/assert.c

```
1 #include <stdio.h>
2 #include <assert.h>
3
4 int mystrlen (char * pszStr){
5     assert(pszStr!=NULL);
6
7     printf(" alles ok");
8     return 0;
9 }
10
11 int main(void){
12     char * p = {"Hallo"};
13     mystrlen(NULL);
14     return 0;
15 }
```

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

A large, light gray circular graphic containing a white number '4' followed by an exclamation mark '!', positioned on the right side of the slide.

4!

- 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



4!

# Was ist ein Debugger?

Werkzeug zur Überwachung eines Programms während dessen Ausführung.



4!

# Was ist ein Debugger?

Seine Features:

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

4!



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

A large, light gray graphic in the background consists of several overlapping circular and semi-circular shapes. In the center-right, there is a prominent circle containing the number '4' followed by an exclamation mark '!', both in white. The overall aesthetic is clean and modern, typical of a technical presentation.

# 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 }
```

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
```





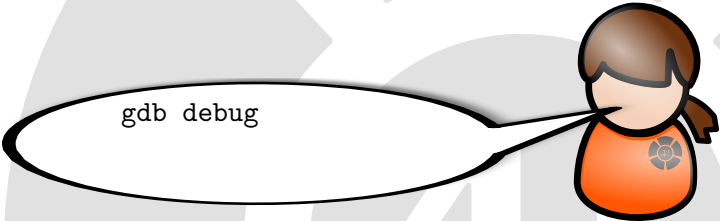
## 2. Schritt: GDB starten

Wir starten den GDB und laden unsere Datei:



## 2. Schritt: GDB starten

Wir starten den GDB und laden unsere Datei:



```
gdb debug
```

### 3. Schritt: Setzen eines Breakpoints

Wir setzen einen Breakpoint auf die for-Schleife



### 3. Schritt: Setzen eines Breakpoints

Wir setzen einen Breakpoint auf die for-Schleife



`break 9`

Anzeige alle Breakpoints: `info breakpoints`

## 4. Schritt: Starten des Programms

Wir starten das Programm:



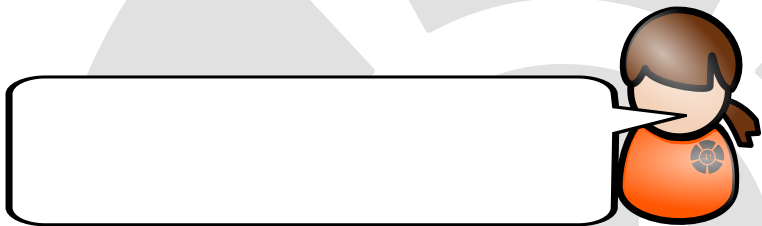
## 4. Schritt: Starten des Programms

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:

```
next  
step  
cont
```



## Das neue Verhalten des Beispiels

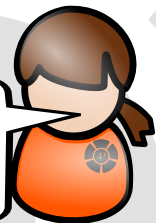
Wir korrigieren das Programm und initialisieren die Variable `j`. Kompilieren neu und erhalten folgendes Verhalten:



## 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

Welche Werte durchläuft `j` denn überhaupt?

Zum "Beobachten" von `j`: Watchpoint setzen. Dazu 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`

4!

## 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:



1, 2, 6

## 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.

Wir erkennen:  
Es fehlt eine Iteration  
der `for`-Schleife!



Wir korrigieren den Programmcode zu:

Listing 50: cbugs/debug.c

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

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`

A large, light gray circular graphic containing the number '4' and an exclamation mark '!' in white, positioned on the right side of the slide.

## 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.freitagrunde.org/Ckurs2009/  
BubbleSortDebug
```

Zum weiteren Selbststudium:

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

A large, light gray circular graphic containing the number '4' and an exclamation mark '!' in white, positioned on the right side of the slide. The background features a faint, larger version of this graphic.

## 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

Fragen?

