HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Praktikum Ingenieurinformatik

Termin 3

Fehlervermeidung, Fehlersuche, Schleifen (while, do-while, for)

1. **Fehlervermeidung, Fehlersuche**
2. **Schleifen**
3. **Debugger**

1.1. Fehlertypen

Fehler bei der Problemanalyse

- Das Problem ist unvollständig oder falsch beschrieben.
- Spezialfälle werden nicht beachtet oder sind nicht beschrieben.

Fehler im Lösungsalgorithmus

Fehler beim Übersetzen

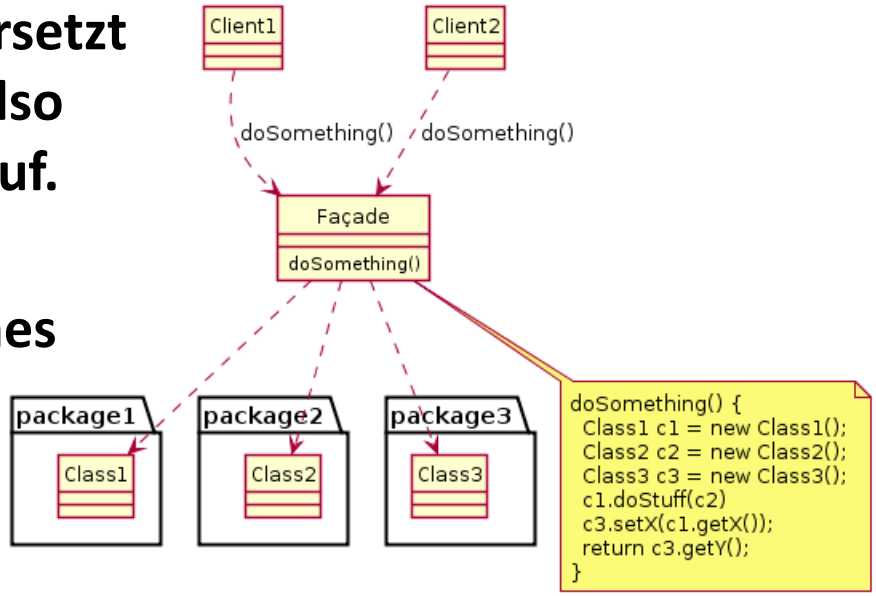
- Das Programm kann nicht übersetzt werden, Compiler zeigt Fehler an.

Laufzeitfehler

- Das Programm kann ohne Fehler übersetzt werden. Nach dem Programmstart (also während der Laufzeit) treten Fehler auf.

Um die ersten beiden Fehlertypen zu vermeiden, wendet man ein methodisches Vorgehen an:

- Strukturierte Programmierung
- UML Unified Modeling Language (Standardverfahren in der Industrie)



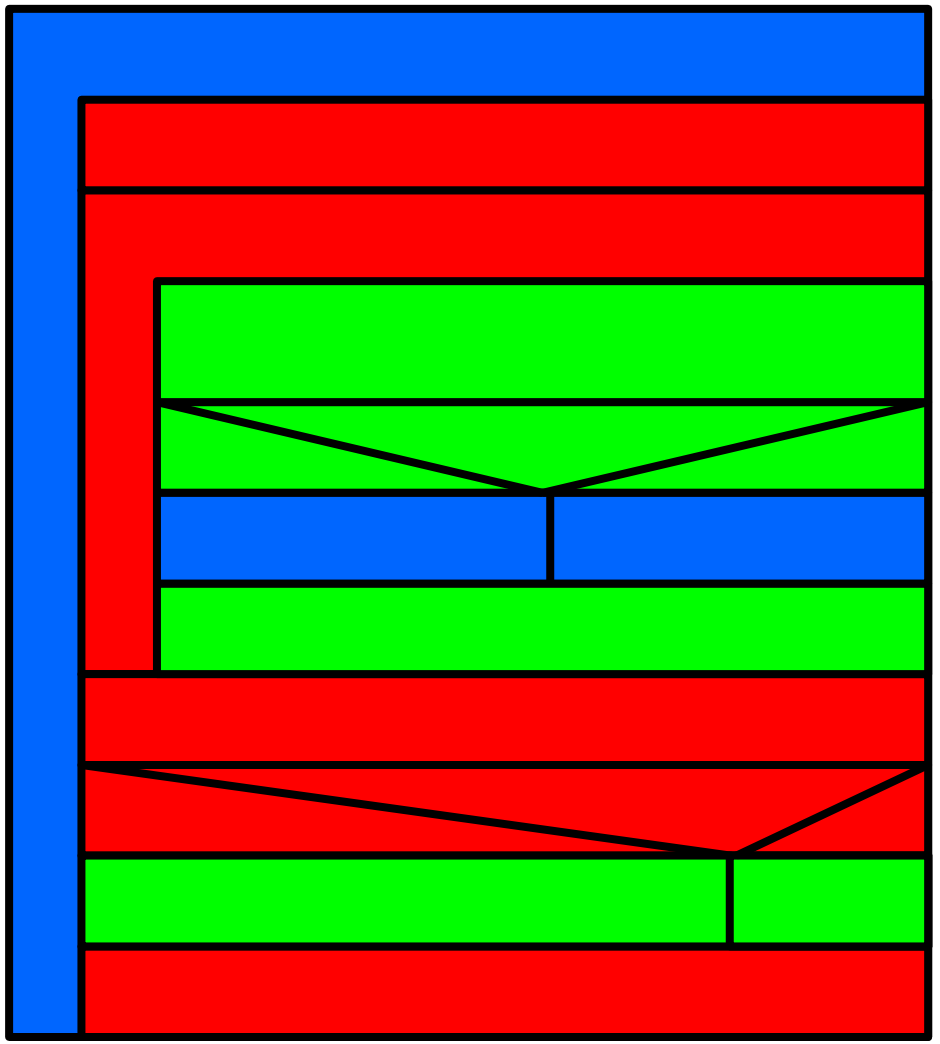
Wikimedia Commons, Example of Façade design pattern in UML.png, CC BY-SA 3.0

1.2. Vermeidung von Fehlern im Programmcode

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Quelltext gut formatieren und vor dem Übersetzen nochmals durchlesen!

```
for(i = 1; i < 100; i = i + 1)
{
    k = k * k;
    while(i < 100)
    {
        k = k + i;
        i = i + 8;
        if(i < 5)
            i = i + 6;
        else
            i = i * 8;
        k = k + 56;
    }
    printf("k = %d\n", k);
    if(i < 7)
    {
        printf("i = %d\n", i);
        i = i + 3;
    }
    k = k - 1;
}
```



1.3. Übung: Quelltext formatieren

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

```
#include <stdio.h>
int main()
{int a, b, i;
for(a = 1; a <= 12; a++)
{switch(a)
{case 2: b = 28; break;
case 4: case 6: case 9: case 11: b = 30;
break;
default: b = 31; break;}
printf("Monat: %d\n", a);
for(i = 1; i <= b; ++i)
{printf("%2d ", i);
if(i % 10 == 0)
printf("\n");}
printf("\n\n");}
return 0;}
```

Formatieren Sie das Programm, indem Sie in jeder Ebene (Verschachtelung) um jeweils vier Spalten einrücken.

Finden Sie heraus, wozu das Programm dient.

Das Programm ist schwer lesbar, weil nichtssagende Variablennamen verwendet werden.

1.4. Übersetzungs- oder Compilerfehler

Der C-Quelltext kann aufgrund von Syntaxfehlern oder aus anderen Gründen (z. B. falsche oder keine Include-Datei angegeben) nicht übersetzt werden. Es wird kein ausführbares Programm erstellt.

Typische Fehler:

- **Semikolon vergessen**
- **Klammer vergessen/überzählig**
- **Hochkomma nicht abgeschlossen**
- **Groß-/Kleinschreibung**
- **Ausdruck falsch formuliert**
- **Schreibfehler**
- **Variable wird verwendet, wurde aber vorher nicht definiert**

Fehlermeldungen des Compilers:

```
c:\temp\vorlesung\quelle.cpp(10):
error C2143: Syntaxfehler: Es fehlt ";" vor "for"
```

- **Der Fehler muss nicht genau dort sein, wo der Compiler ihn vermutet, es könnte sich z. B. auch um einen Folgefehler handeln.**
- **Nach der Fehlerbeseitigung muss der Quelltext erneut gespeichert und übersetzt werden.**

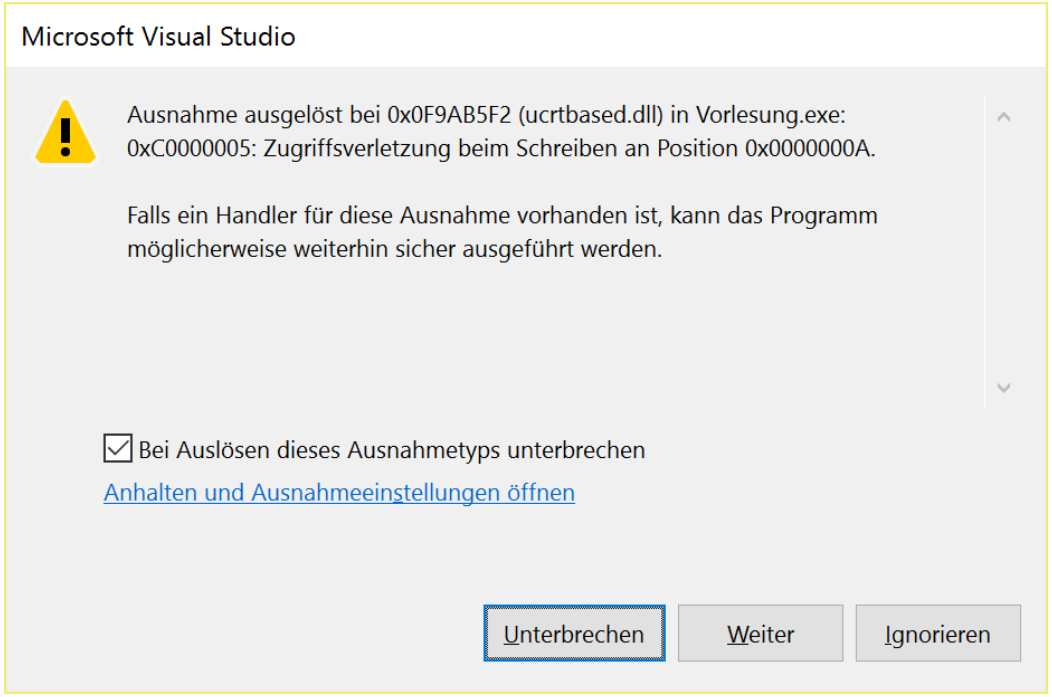
1.5. Laufzeitfehler

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Der C-Quelltext kann ohne Fehler übersetzt werden, es wird ein ausführbares Programm erzeugt. Wenn dann bei der Ausführung des Programms Fehler auftreten, spricht man von Laufzeitfehlern (engl. runtime error).

Typische Fehler:

- **Programm stürzt ab wegen Division durch null oder mit einer „Zugriffsverletzung“ (häufig: & bei scanf fehlt),**
- **Programm stürzt manchmal ab (z. B. nur bei bestimmten Eingaben),**
- **Programm „hängt fest“ (z. B. Endlosschleife),**
- **Programm liefert immer/manchmal falsche Ergebnisse,**
- **Programm liefert ein falsches Ergebnis und man merkt es gar nicht oder es ist nicht offensichtlich (z. B. numerische Lösung einer DGL).**



1.6. Typische Laufzeitfehler

Endlosschleifen:

- ```
  while(i < 5);
    printf("i = %d\n", i);
  ```

(leere Anweisung)

Fehler bei der Ein- und/oder Ausgabe:

- ```
  scanf("%f\n", &i);
  scanf("%d\n", &i);
  ```

Schleife wird „nicht durchlaufen“:

- ```
  for(i = 0; i < 6; i = i + 1);
  {
      printf("i = %d\n", i);
  }
  ```

Tipp für die Prüfung:

Checkliste mit weiteren typischen Fehlern erstellen und mitbringen!

Variable wird ohne vorherige Wert-Zuweisung verwendet:

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

Zuweisung in if-Anweisung:

- ```
  if(i = 5)
      printf("Bedingung ist erfüllt.\n");
  ```

Dezimalpunkt:

- `x = 3,75;` statt `x = 3.75;`
  ```

1.7. Suche nach Laufzeitfehlern

- Quelltext (nochmals) genau studieren
- Alle Eingaben und Zwischenergebnisse zur Kontrolle mit printf ausgeben
- Programm mit Werten testen, bei denen das Ergebnis bekannt ist; falscher Wert kann Hinweis auf die Fehlerursache geben
- Spezial- und Grenzfälle analysieren und besonders testen
- Symbolische Konstanten verwenden, um die Ausgabe von Zwischenergebnissen zu steuern:

```
#define DEBUG 1
    . . .
if(DEBUG == 1) printf("Zeile 7: x = %d\n", x);
    . . .
if(DEBUG == 1) printf("Programm ist in Zeile 25 angekommen.\n");
```

Wenn alle Fehler beseitigt sind:
#define DEBUG 0

- Genauen Programmablauf mit einem Debugger verfolgen

1.8. Übung: Suche nach Laufzeitfehlern (a)


```c
#include <stdio.h>
int main(void)
{
    int z1, z2, z3, summe;
    /* Zahlen einlesen */
    printf("Erste Zahl: ");
    scanf("%d", &z1);
    printf("Zweite Zahl: ");
    scanf("%f", &z2);
    /* Summe berechnen */
    summe = z1 * z2;
    printf("z1 = %d, z2 = %d, ", z1, z2);
    printf("summe = %d\n", summe);
    if(z1 = 17)
        printf("Zahlen sind gleich: z1 = %d\n", z1);
    else
        printf("Zahlen sind ungleich: z1 = %d\n", z1);
    printf("z3 = %d\n", z3);
    return 0;
}
```

Dieses Programm kann übersetzt und gestartet werden. Es enthält dennoch Fehler, die erst zur Laufzeit auffallen. Finden Sie die Fehler!

1.9. Übung: Suche nach Laufzeitfehlern (b)

Schreiben Sie ein Programm, um folgende Summe zu berechnen:
$1^2 + 2^2 + 3^2 + \ldots + 50^2$

```
#include <stdio.h>
int main(void)
{
    int i, quadr, sum;
    i = 0;
    while(i < 50);
    {
        quadr = i * i;
        sum = sum + quadr;
    }
    printf("Summe = %d\n", quadr);
    return 0;
}
```

Das korrekte Ergebnis lautet 42925. Suchen Sie die Fehler im abgebildeten Quelltext!

1.10. Testen von Programmen

Ist ein Programm fertig, sollte man es unbedingt gründlich testen:

- Programm mit Werten testen, für die die Ergebnisse bekannt sind,
- verschiedene Spezialfälle und/oder Grenzwerte ausprobieren, beim Stromrechnungs-Programm einen Verbrauch von 249/250/251 kWh eingeben, negative Eingabewerte ebenfalls testen,
- Verhalten des Programms bei ungültigen Eingabewerten überprüfen,
- genauen Programmablauf mit einem Debugger verfolgen.

Optimierung von Programmen:

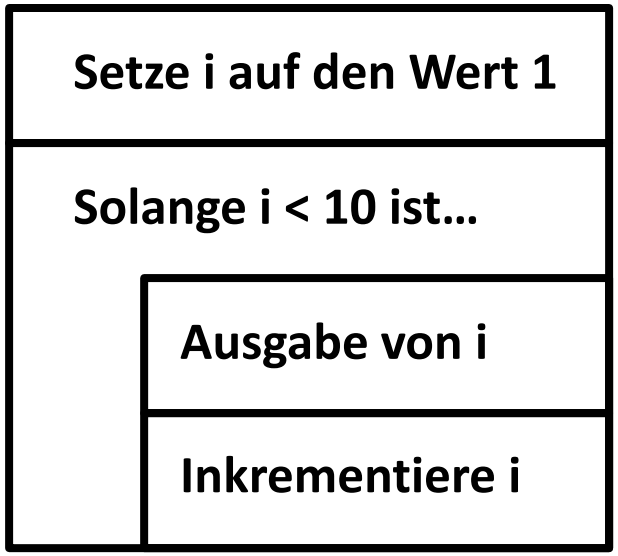
- Rechenzeit (zum Beispiel beim Primzahlprogramm, s. u.),
- Speicherplatzbedarf,
- Aufwand zur Erstellung/Pflege des Programms.

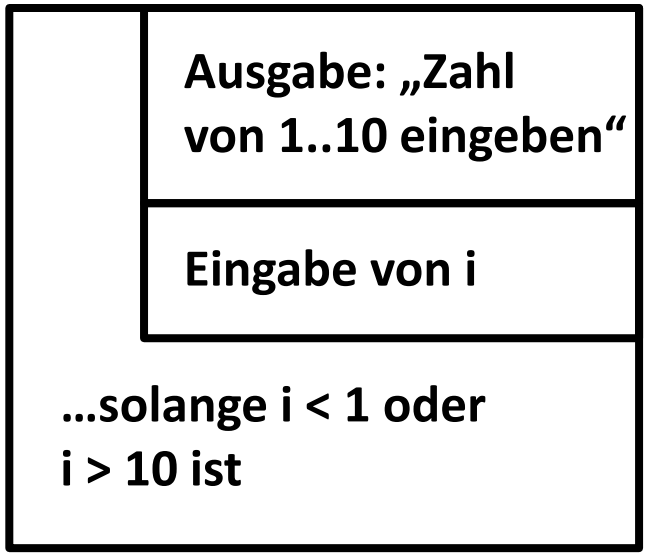
1. Fehlervermeidung, Fehlersuche
2. Schleifen
3. Debugger

2.1. Arten von Schleifen

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Sie haben in der Vorlesung verschiedene Arten von Schleifen kennengelernt. Geben Sie zu den folgenden Struktogrammen den C-Quelltext an!





2.2. Übungsaufgabe (Primzahlen)

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Schreiben Sie ein Programm zur Berechnung aller Primzahlen zwischen 1 und 10000! Eine Primzahl ist eine Zahl, die nur durch 1 und sich selbst ohne Rest teilbar ist. Die kleinste Primzahl ist 2 (die Zahl 1 gilt nicht als Primzahl).

Erstellen Sie zunächst ein Struktogramm!

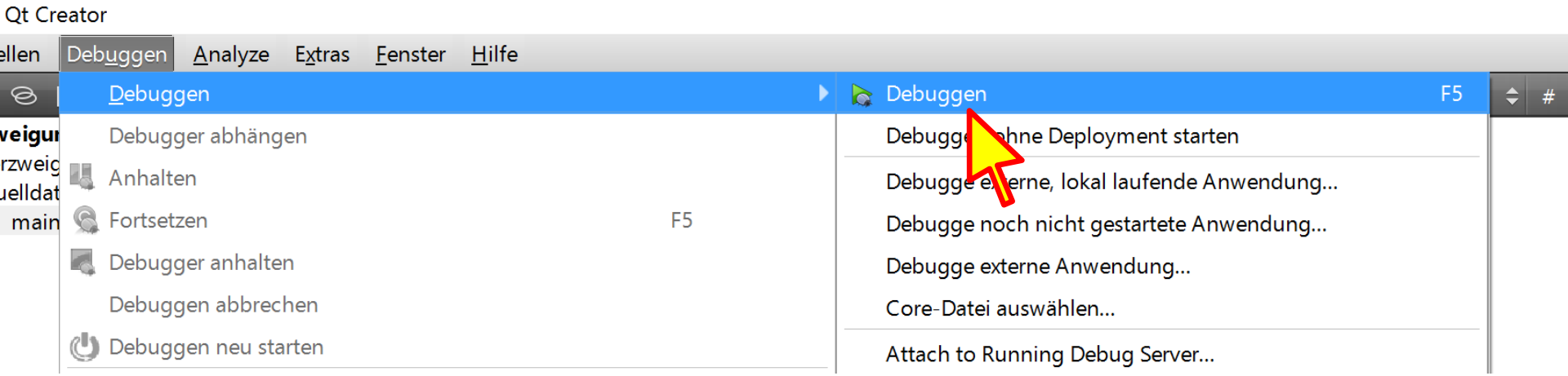
Teilaufgaben:

- Äußere Schleife: Variable "n" läuft von 2 bis 10000
- Innere Schleife: Variable "t" läuft von 2 bis n-1
- Jeweils prüfen, ob n durch t teilbar ist. Falls ja, kann n keine Primzahl sein.
- Falls innere Schleife durchgelaufen ist, ohne einen Teiler zu finden, wird die Primzahl n ausgegeben.

1. **Fehlervermeidung, Fehlersuche**
2. **Schleifen**
3. **Debugger**

3.1. Debugger

Der Debugger (bug = Fehler, Wanze) ist ein Hilfsmittel, um laufende Programme zu analysieren. Programme können an bestimmten Stellen (**Breakpoints**) angehalten oder **zeilenweise abgearbeitet** werden. Aktuelle **Variablenwerte werden angezeigt** und können sogar während der Laufzeit verändert werden.



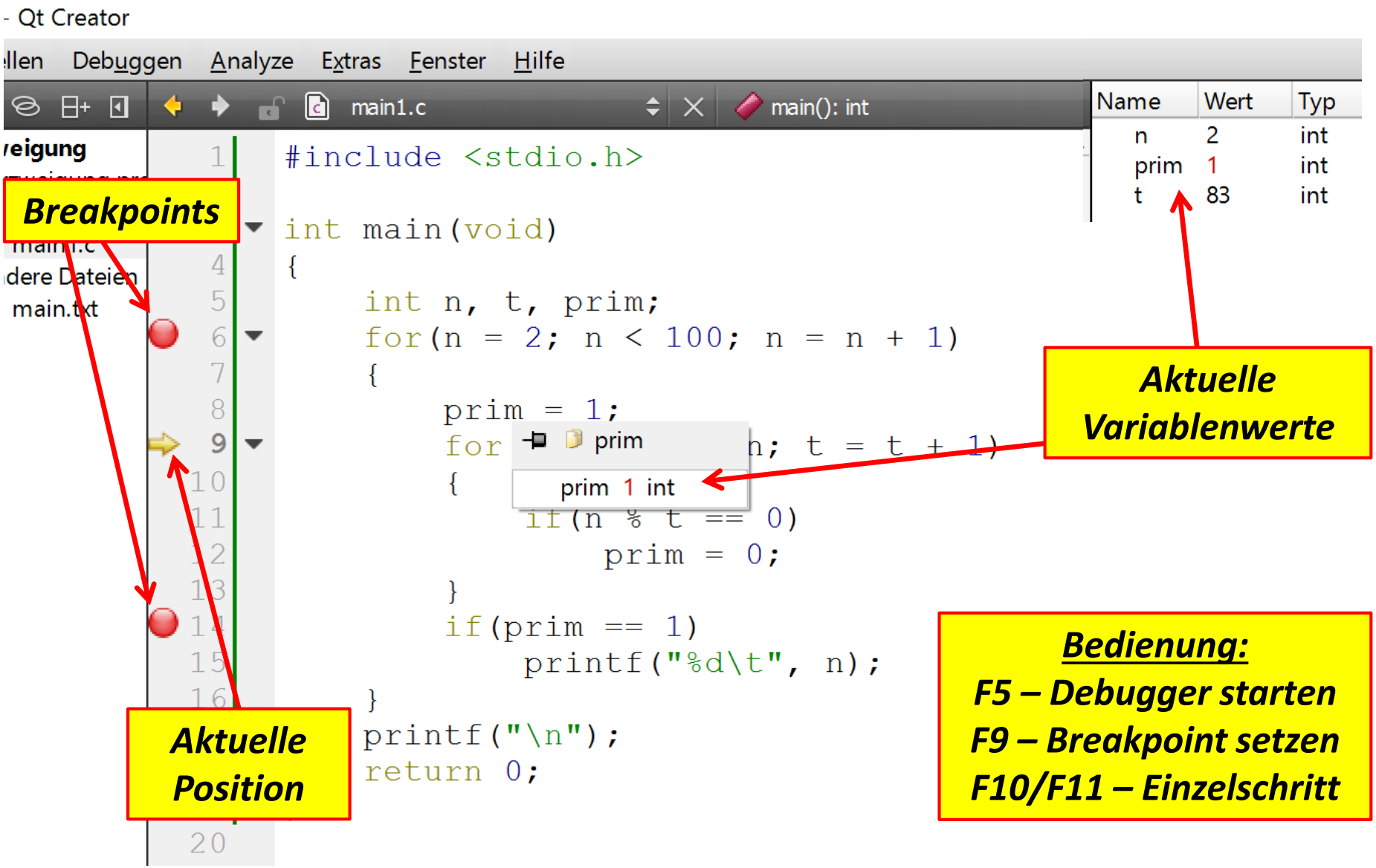
Mit dem Debugger können Sie:

- Fehler im Programm finden,
- sich von der Korrektheit eines Programms überzeugen,
- den Ablauf unbekannter Programme verstehen.

Arbeiten mit dem Debugger, ein „Kochrezept“:

1. Per Mausklick auf den grauen Rand (links neben der Zeilennummer) oder alternativ mit der **Taste F9** wird ein **Breakpoint** an den Beginn des Hauptprogramms oder an den Beginn des zu untersuchenden Bereichs gesetzt.
2. Nun wird über den Menüpunkt **Debuggen → Debuggen** oder alternativ mit der **Taste F5** der Debugger gestartet.
3. Der Programmablauf wird am Breakpoint angehalten, die aktuelle Position im Programm wird durch einen kleinen gelben Pfeil angezeigt.
4. Wenn der Mauszeiger über eine Variable gehalten wird, dann wird der aktuelle Wert dieser Variablen in einem kleinen Fenster angezeigt.
5. Der weitere Programmablauf kann über die **Tasten F10** (bis zur nächsten Zeile ausführen), **F11** (in einen Funktionsaufruf hinein springen) und **F5** (bis zum nächsten Breakpoint ausführen) gesteuert werden.
6. Es können auch mehrere Breakpoints gleichzeitig gesetzt werden.
7. Über den Menüpunkt **Debuggen → Debugger anhalten** wird der Debugger wieder beendet.

3.3. Debugger


- Qt Creator
llen Debuggen Analyze Extras Fenster Hilfe
main1.c
main(): int
Name Wert Typ
n 2 int
prim 1 int
t 83 int
#include <stdio.h>
int main(void)
{
int n, t, prim;
for(n = 2; n < 100; n = n + 1)
{
prim = 1;
for
n; t = t + 1)
{
prim
prim 1 int
if(n % t == 0)
prim = 0;
}
if(prim == 1)
printf("%d\t", n);
}
printf("\n");
return 0;
Breakpoints
Aktuelle Variablenwerte
Aktuelle Position
Bedienung:
F5 – Debugger starten
F9 – Breakpoint setzen
F10/F11 – Einzelschritt

3.4. Übungsaufgabe (Debugger)

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Starten Sie das Primzahlprogramm mit dem Debugger:

- Verfolgen Sie einige Schleifenabläufe Schritt für Schritt.
- Schauen Sie sich die aktuellen Werte der Variablen während des Programmablaufs an.
- Setzen Sie Breakpoints. Zum Beispiel direkt nach dem Ende der inneren Schleife, so dass der Debugger nach jedem Ablauf der inneren Schleife automatisch angehalten wird.
- Wechseln Sie die Art der Schleife (**while, do-while, for**) in Ihrem Programm und testen Sie die geänderte Version erneut mit dem Debugger.