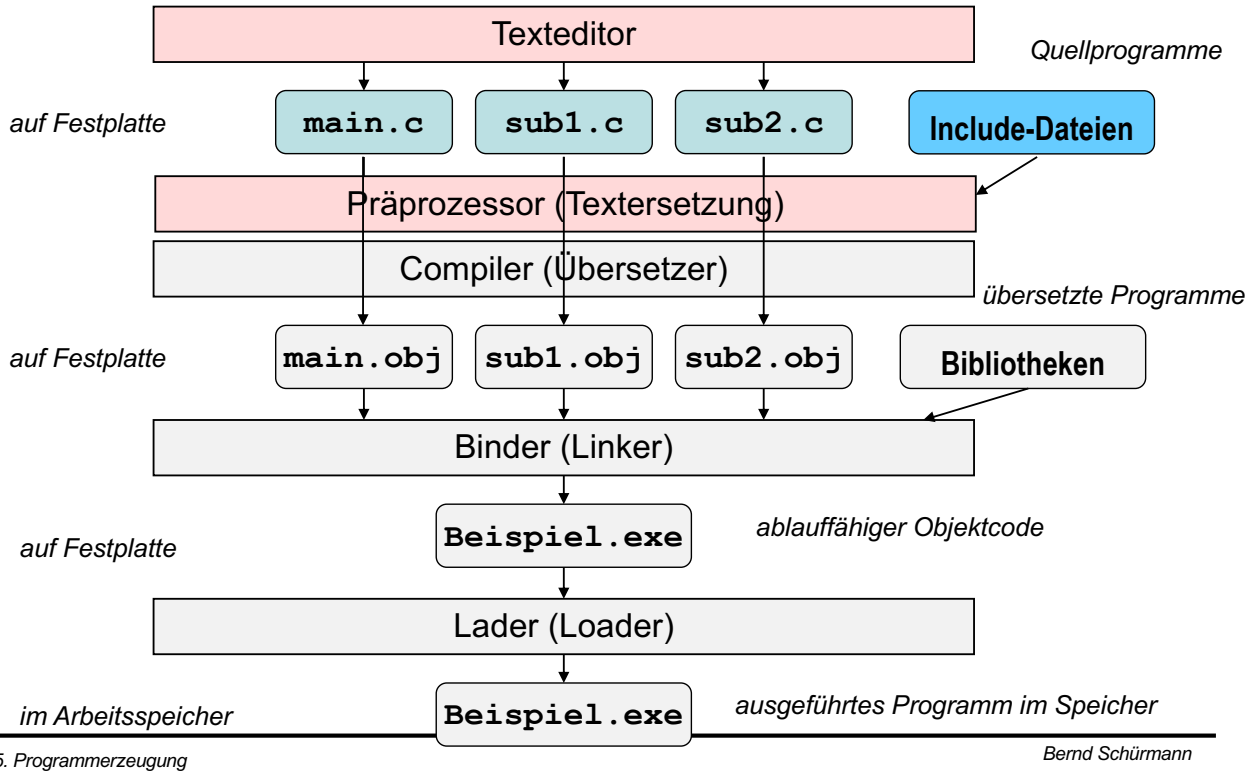
Programmieren in C

für Elektrotechniker

Kapitel 5: Programmerzeugung

- **Präprozessor**
- Programmübersetzung
- Fehlersuche
- Debugger
- Entwicklungsumgebungen

Übersetzung und Ausführung



Präprozessor

- **Aufgaben**
  - Einfügen von Dateien (`#include ...`)
  - Ersetzen von Text (`#define ...`)
  - Bedingte Übersetzung (`#if ...`)

- **Syntax**
  - `#Direktive Text <return>` Interpretation durch den Präprozessor bis zum Zeilenende.
  - `#Direktive Text1\`
`Text2<return>` Interpretation durch den Präprozessor über zwei (oder mehrere) Zeilen.

- **Einfügen von Dateien**

  `#include <filename>`

  - Präprozessor sucht in den typischen Include-Verzeichnissen nach der Datei `filename`, z.B. in `/usr/include`.

  `#include "filename"`

  - Präprozessor sucht zunächst im aktuellen Verzeichnis und dann in den typischen Include-Verzeichnissen nach der Datei `filename.`

- Nach dem Einkopieren des Inhalts der Datei `filename` wird der Präprozessor den einkopierten Text analysieren.

Ersetzen von Text

- Typische Anwendungen: - symbolische Konstanten
  - Makros

```
#define HUND DOG
```

- Ersetzt alle Vorkommen **HUND** durch **DOG**.
- Keine Ersetzung von - **HUNDERT**
  - **" Dies ist ein HUND. "**

Programmieren in C Präprozessor

Ersetzen von Text

- Typische Anwendungen: - symbolische Konstanten
  - Makros

```
#define Bezeichner (Par_1, …, Par_n) Ersatztext
```

- Setzt den **Ersatztext** an den Stellen **Bezeichner** ein, wobei die aktuellen Parameter in den **Ersatztext** eingesetzt werden.
- Beispiel:

```
#define evalmac(z) z*=(z+1)/2 /* Gauß-Formel */
....
int a = 5;
evalmac(a);
....
```

➔

```
....
int a = 5;
a*=(a+1)/2;
....
```

Achtung:
Makros sehen aus wie Funktionen (→ Aufruf und Parameterersetzung zur Laufzeit), es handelt sich aber um eine Textersetzung zur Übersetzungszeit.

▪ Bedingte Übersetzung

```
#if konstanter_Ausdruck        #ifdef Symbol
#elif konstanter_Ausdruck      #if defined(Symbol)
#else                          #ifndef Symbol
#endif
```

- Anwendungen z.B. - Anpassung des Programms an das Rechnersystem
  - Einbinden von Test-Code
- Beispiel

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

#define TESTVERSION

int main (void) {

    #if INT_MAX > 32767
       int i;
    #else
       long i;
    #endif

    #ifdef TESTVERSION
    printf ("Test: sizeof (int) = %d\n", sizeof (int));
    printf ("Test: sizeof (long) = %d\n", sizeof (long));
    #endif
... }
```


▪ Vordefinierte Standard-Makros

__LINE__	Liefert Zeilennummer, in der **__LINE__** steht.
__FILE__	Liefert den Dateinamen, in dem **__FILE__** steht.
__DATE__	Liefert das aktuelle Datum im US_Format als Zeichenkette.
__TIME__	Liefert die aktuelle Uhrzeit im Format hh:mm:ss als Zeichenkette.

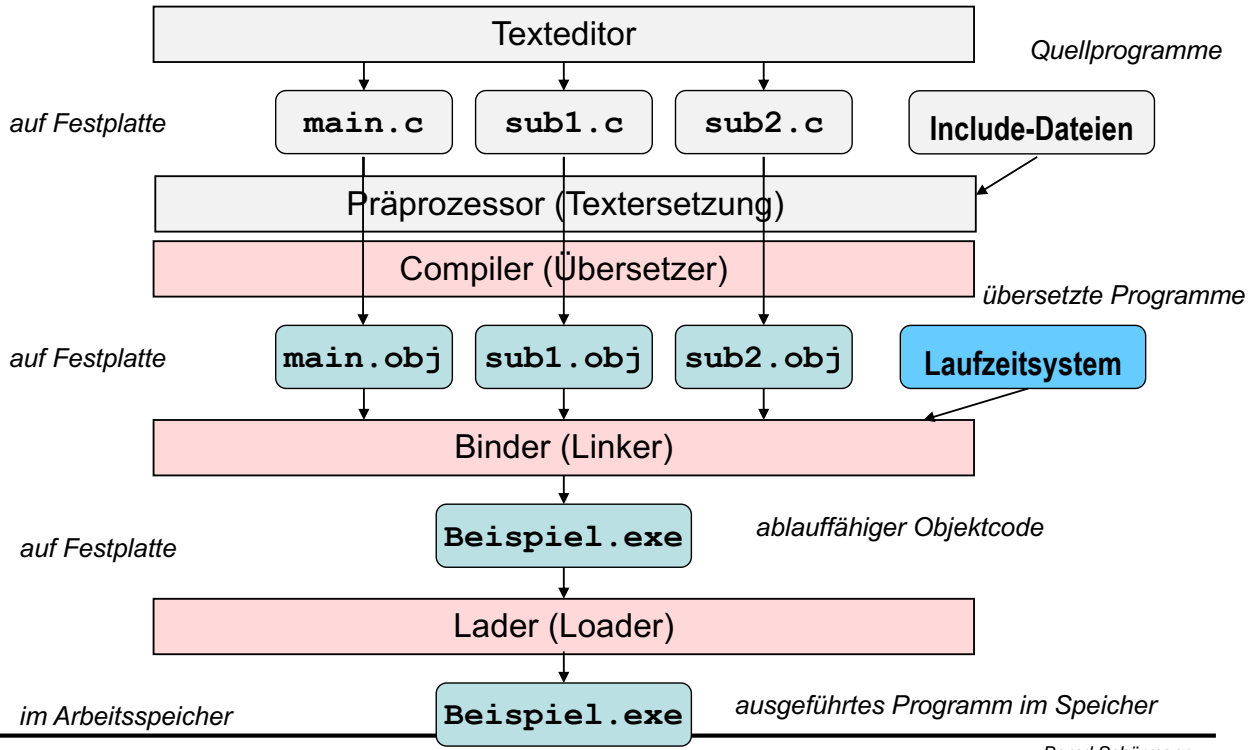
Programmieren in C

für Elektrotechniker

Kapitel 5: Programmerzeugung

- Präprozessor
- **Programmübersetzung**
- Fehlersuche
- Debugger
- Entwicklungsumgebungen

Übersetzung und Ausführung



Compiler (Übersetzer)

- Quellcode → Maschinencode (Assembler, *vgl. EIT-Vorlesung*)
- Syntax einer Programmiersprache durch Metasprache definiert

Beispiel: Ausschnitt aus C-Syntax in Backus-Naur-Form

```
<stmt> ::= <labeled-stmt>  | <expr-stmt> | <compound-stmt>
         | <select-stmt> | <iter-stmt>  | <jump-stmt>
<labeled-stmt> ::= ….
<expr-stmt> ::= {<expr>}? ;
<select-stmt> ::= if ( <expr> ) <stmt>
                | if ( <expression> ) <stmt> else <stmt>
                | switch ( <expression> ) <stmt>
<iter-stmt> ::= while ( <expression> ) <stmt>
              | do <stmt> while ( <expr> ) ;
              | for ( {<expr>}? ; {<expr>}? ; <expr>}? )
                <stmt>
<expr> ::= <assignment-expr>
         | <expr> , <assignment-expr>
```

→ *nicht näher in dieser Vorlesung*

Übersetzung in 5 Schritten

- Lexikalische Analyse

  → Programm: Scanner

  Erkennung der „Grundeinheiten“ einer Programmiersprache (→ Token):
  Wörter, Zahlen, Trennzeichen etc.

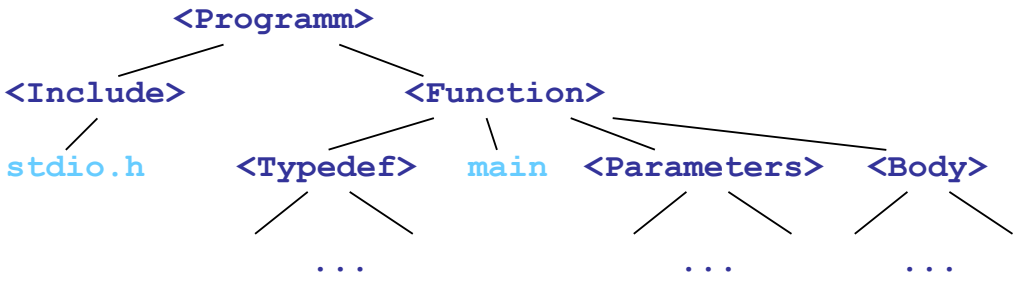
```
/* File: hello_world.c */
#include <stdio.h>

int main (void)
{
     printf ("Hello, world!\n");
     return 0;
}
```

```
T1 = /* File: ...c */
T2 = #include
T3 = <
T4 = stdio.h
T5 = >
T6 = int
T7 = main
T8 = (
T9 = void
...
T20 = }
```

Übersetzung in 5 Schritten

- Syntaxanalyse
  → Programm: Parser
  Syntax (Grammatik): Regelwerk, das beschreibt, welche Folge von Tokens zulässig ist.
  → vgl. Deutsch: `Satz ::= <Subj> <Prädikat> <Obj> .`

  Parser: Teil des Compilers, der das Programm auf korrekte Syntax prüft und in seine syntaktische Struktur zerlegt.



Übersetzung in 5 Schritten

- Semantische Analyse
  - analysiert die Bedeutung der Programmteile
  - erzeugt abstrakten Zwischencode

  → keine vollständige semantische Analyse,
  sondern Analyse der Bedeutung von Namen
  (→ Sichtbarkeits-, Gültigkeits-, Typregeln – s.u.)

  - statische Semantik: vom Compiler überprüft
  - dynamische Semantik: erst zur Laufzeit prüfbar *(→ Laufzeitsystem – s.u.)*

Übersetzung in 5 Schritten

- Optimierungen

  u.a.
  - Speicherplatz
  - Rechenzeit

  → *andere Vorlesungen*

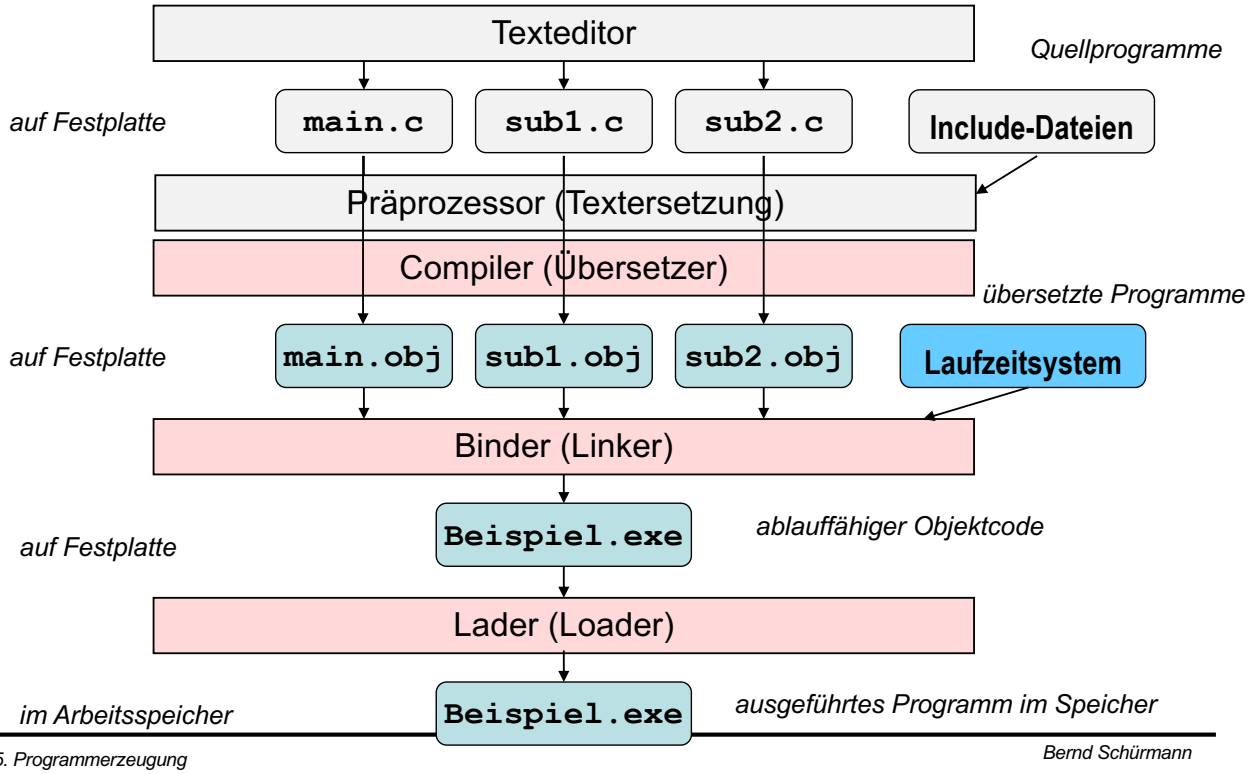
Übersetzung in 5 Schritten

- Codeerzeugung

  Zwischencode → Maschinencode

  → Einsetzen von Adressen

Übersetzung und Ausführung



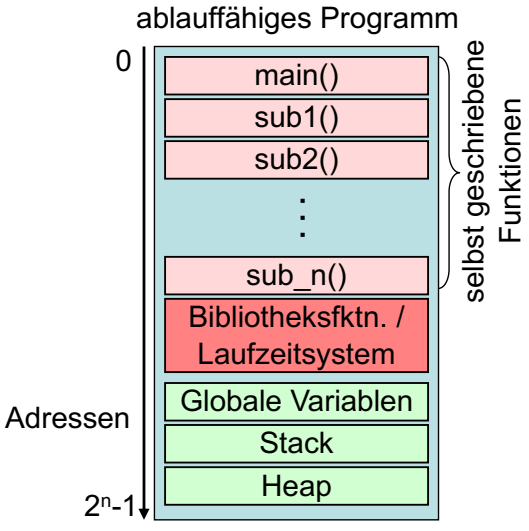
Laufzeitsystem

- Komponenten, die zur Ausführung eines Programms notwendig sind, z.B.
  - Betriebssystem
  - Prüfung der dynamischen Semantik
  - Fehlerbehandlung
  - Interpreter (nicht bei C)

- **Binder (Linker)**

  Fügt Objektcode einzelner Module (Übersetzungseinheiten) zusammen.
  - Auflösung von Querreferenzen.
  - Hinzufügen von Bibliotheken.

  Ergebnis: einheitlicher Adressraum für gesamtes Programm.



Laufzeitsystem

- Komponenten, die zur Ausführung eines Programms notwendig sind, z.B.
  - Betriebssystem
  - Prüfung der dynamischen Semantik
  - Fehlerbehandlung
  - Interpreter (nicht bei C)

- **Lader (Loader)**
  - Programm wird in Arbeitsspeicher geladen.
  - Programm wird gestartet
    (evtl. mit Eingabe von Argumenten – s.u.).

Programmieren in C
für Elektrotechniker

Kapitel 5: Programmerzeugung

- Präprozessor
- Übersetzer
- **Fehlersuche**
- Debugger
- Entwicklungsumgebungen

Fehlersuche in Programmen

- **Goldene Regeln für Programmierer**
  - *Alles, was schiefgehen kann, wird auch schiefgehen.*
  - *Alles, was nicht schiefgehen kann, wird voraussichtlich doch schiefgehen.*

  → Programme systematisch testen!
  In der Praxis:
  - Debugging-Werkzeuge
  - Testumgebungen ⟶ *nicht in der Vorlesung*

- **Fehlerklassen**
  - Syntaxfehler: vom Compiler entdeckt
  - Laufzeitfehler: viel schwerer zu entdecken

Fehlersuche in Programmen

- **Grundlegende Guidelines**
  - **Programme gut strukturieren**
    - Kleine Funktionen, die gut zu strukturieren sind.
    - Aufrufhierarchie funktionsweise von unten nach oben testen.
  - **Funktionen testen**
    - Funktionen einzeln mit Hilfe von Testprogrammen überprüfen.
  - **Testausgaben einsetzen**

    → später wieder löschen!
  - **Prüfen des allgemeinen Falls, aber auch aller Sonderfälle** *(Eingabewerte 0,” ”, …)*
    - Bei Verzweigungen möglichst Werte nahe der Grenze wählen
    - Beispiel:
      ```
      if (u <= 30) {
      . . .
      }
      ```
      → testen mit 29.9999, 30.0, 30.0001
    - Achtung: Viele Fehler treten erst nach langer Zeit auf
      (wenn kritische Datenkonstellationen zufällig auftreten).

Fehlersuche in Programmen

- **Ausgabe von temporären Werten**
  - **Testausgaben**

    Temporäre Ausgaben im Programm (→ `printf(…)`),
    die im „Produktivsystem“ wieder gelöscht/auskommentiert werden.
  - **Nutzung des Präprozessors (s.o.)**

    Über die Direktive `if` … Code nur während der Testzeit in das Programm einbinden.

    • Beispiel:

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

int main (void) {

#ifdef TEST
printf („Test: Wert der Variable x an Zeile %d: %d\n",
                        __LINE__, x);
#endif
... }
```


Programmieren in C

für Elektrotechniker

Kapitel 5: Programmerzeugung

- Präprozessor
- Übersetzer
- Fehlersuche
- **Debugger**
- Entwicklungsumgebungen

Debugger

Programm, das andere Programme ausführt (→ Interpreter) und den Benutzer die Kontrolle über das ausgeführte Programm gibt.
➔ Größter Freund eines Programmierers. ☺

- **Typische Aufgaben und Aktionen**
  - Breakpoints: Programmstellen, an denen die Programmausführung anhält.
  - Einzelschritte: Programm (-teile) werden Anweisung für Anweisung durchlaufen.
  - Anzeige aktueller Variablenwerte.
  - Setzen von Variablenwerten.
  - Anzeige des Systemzustands.

Debugger

- **Beispiel: GNU debugger (gdb)**

  → grafische Oberfläche unter dev-C++.
  - C-Programm muss mit „-g“-Option übersetzt worden sein.

    ```
    gcc -g hello_world.c -o hello_world
    ```
  - Wichtige Befehle

`b N`	Breakpoint in Zeile **`N`**.
`b fn`	Breakpoint zu Beginn der Funktion **`fn`**.
`d M`	Lösche breakpoint Nr. **`M`**.
`info bp`	Listet alle breakpoints.
`r`	**`run`** – Programmstart bis breakpoint oder Fehler.
`c`	**`continue`** – analog zu r - ab breakpoint.
`f`	Programmausführung, bis Funktion beendet wurde.
`s / n`	**`step / next`** – Ausführung einer Zeile/Anweisung.
`s N`	Ausführung von n Anweisungen.
`p <var>`	Ausgabe des aktuellen Werts der Variablen **`var`**.
`w <var>`	Watch. Stopp und Ausgabe von **`var`** bei Änderung.
`u / d`	Eine Ebene höher/tiefer im Stack.
`q`	**`Quit`**.

- Beispiel: Ein Programm, das eine Variable **`i`** mit **`5`** belegen und diesen Wert in der Funktion **`sub`** ausgeben soll.
  → *Ist das Programm korrekt?*

```
#include <stdio.h>
int sub(int x) {
    printf("x hat den Wert %d\n", x);
}
int main() {
    int i;
    printf("Programmstart.\n");
    sub(i);
    return 0;
}
```

Übersetzung und Ausführung des Programms:

```
% cc -g -o temp temp.c
% ./temp
Programmstart.
x hat den Wert 4231
```

Fehler: Es wird nicht der erwartete Wert **`5`** ausgegeben.


```
% gdb temp
GDB is free ...
(gdb) break main
Breakpoint 1 at 0x160f: file temp.c, line 5.
(gdb) run
Starting program: /home/james/tmp/temp
Breakpoint 1, main () at temp.c:5
(gdb) n
Programmstart.
(gdb) s
sub (x=4231) at temp.c:2
(gdb)
```

Überspringe Aufruf-Code.
→ breakpoint gesetzt.
Ausführung bis `main()`.

→ gdp stoppt bei `main()`.
Gehe zur nächsten Anweisung.

Nächste Zeile: Funktion `sub()`.

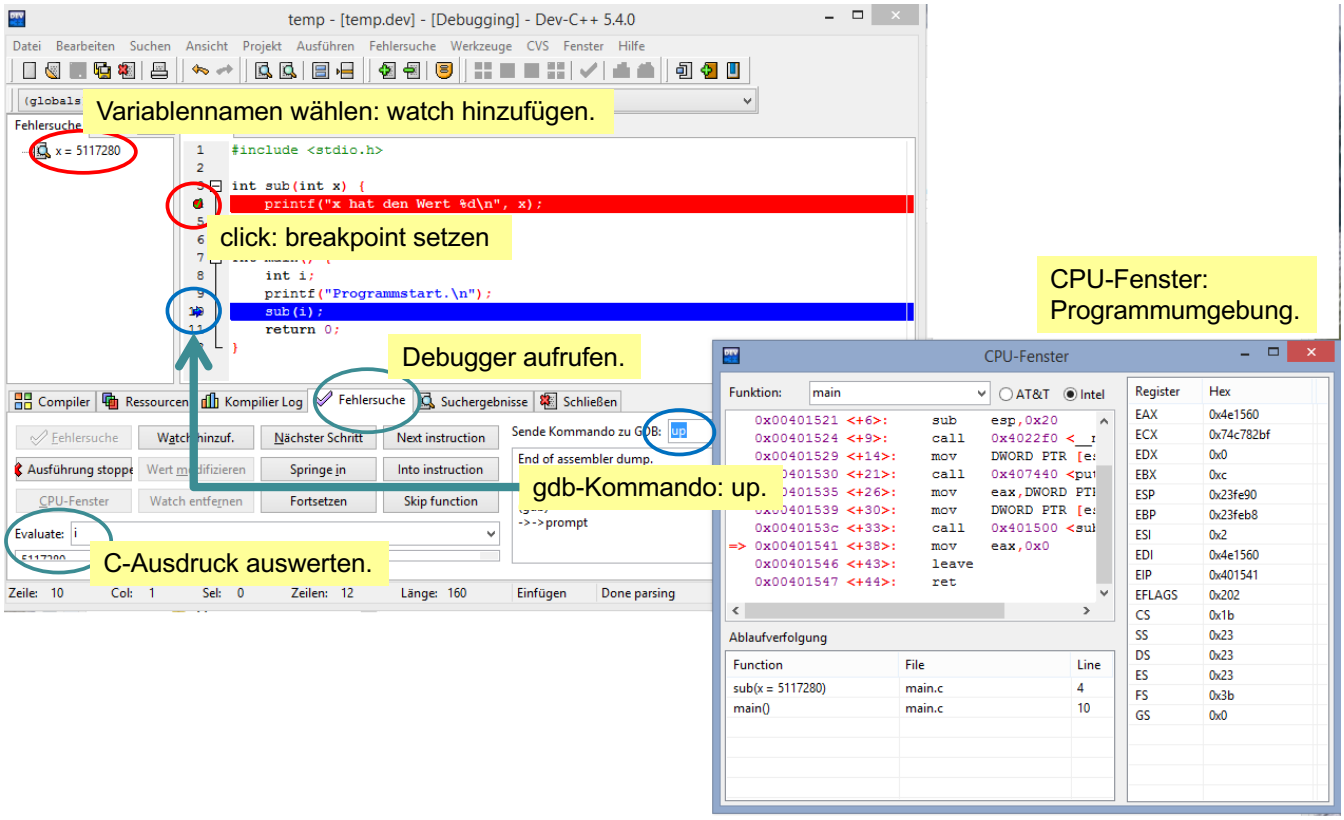
→ **`x`** hat den falschen Wert.

```
(gdb) up
#1  0x1625 in main () at temp.c:8
(gdb) p i
$1 = 4231
```

Eine Ebene höher.

Ausgabe von `i`.

→ Erkannt, dass **`i`** nicht mit **`5`** belegt wurde.



Programmieren in C

Programmieren in C

für Elektrotechniker

Kapitel 5: Programmerzeugung

- Präprozessor
- Übersetzer
- Fehlersuche
- Debugger
- **Entwicklungsumgebungen**

Integrierte Entwicklungsumgebung

(IDE – integrated development environment)

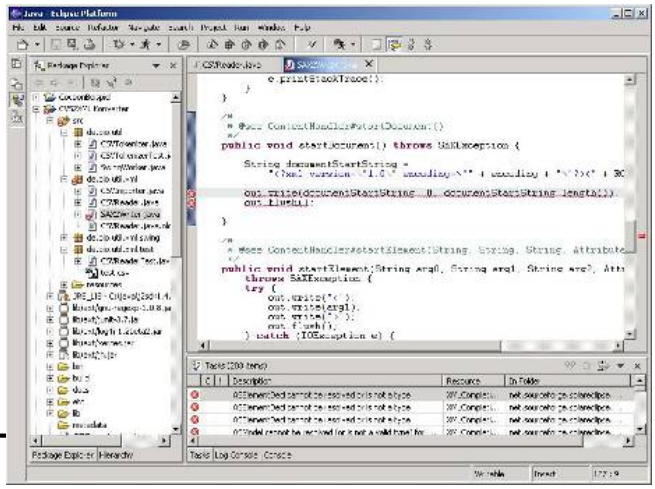
- Programmsystem mit folgenden Komponenten:
  - Editor *(meist sprachsensitiv)*
  - Compiler
  - Binder, Lader
  - Debugger
  - Projektverwaltung *(Verwaltung aller Dateien)*
  - Modellierung *(z.B. UML)*

Programmieren in C Laufzeitsystem

Integrierte Entwicklungsumgebung

(IDE – integrated development environment)

- Beispiel: Eclipse
  - open source, ursprünglich für Java, mittlerweile plugins für viele Sprachen
  - sprachsensitiver Editor
  - leistungsstarke Suche
  - CVS-Unterstützung *(Concurrent Versions System)*
  - Debugger



Integrierte Entwicklungsumgebung

(IDE – integrated development environment)

- Beispiel: MS Visual Studio Express (VSE)
  - kostenlose Microsoft-Entwicklungsumgebung
  - sprachsensitiver Editor (Einfärbung)
  - Debugger
  - optimiert für Web-Entwicklung