C++ - Einführung in die Programmiersprache

Fehler abfangen

Warnungen

- Hinweise auf riskanten Code.
- Eine Kompilierung wird nicht verhindert.
- Um größere Fehler zu vermeiden, sollten alle Warnungen aus dem Code entfernt werden.

```
#include <iostream>;

using namespace std;
int main ()
{
    int x = 0;
    double y = 5.34;

    x = (int)y;


    return 0;
}
```

```
rm -f "build/Debug/MinGW-Windows/main.o.d"
g++    -c -g -MMD -MP -MF "build/Debug/MinGW-Windows/main.o.d" -o build/Debug/MinGW-Windows
main.cpp:1:20: warning: extra tokens at end of #include directive
 #include <iostream>;
                    ^
mkdir -p dist/Debug/MinGW-Windows
g++     -o dist/Debug/MinGW-Windows/cppapplication_1 build/Debug/MinGW-Windows/main.o
make.exe[2]: Leaving directory `/d/Users/aue/Documents/Netbean/CppApplication_1'
make.exe[1]: Leaving directory `/d/Users/aue/Documents/Netbean/CppApplication_1'

BUILD SUCCESSFUL (total time: 1s)
```

Softwarefehler

- Programmierfehler entstehen beim Schreiben des Programmcodes.
- Logische Fehler können durch Denkfehler bei der Umsetzung der Aufgabe in ein Programm erzeugt werden. Das Programm wird fehlerfrei ausgeführt, aber das Ergebnis ist nicht korrekt.
- Laufzeitfehler treten während der Ausführung des Programms auf. Zum Beispiel ist eine Datei nicht an dem angegebenen Speicherort.

Programmierfehler

- Syntaxfehler. Die Programmierung entspricht nicht der Syntax der Programmiersprache.
- Typfehler. Ein Ausdruck oder eine Anweisung hat einen falschen Datentyp.
- Während der Kompilierung des Programms werden die Fehler erkannt.

Beispiele für Syntaxfehler

- Tippfehler bei der Eingabe von Variablennamen oder Schlüsselwörtern.
- Syntaxfehler in Schleifen oder bedingten Anweisungen.
- Falsche Parameterübergabe an Funktionen.

Beispiel für Programmierfehler

```
const int maxWert = 10;
int summe = 0;
string zahl = "5";

// Syntaxfehler in der for-Schleife!
for(int count, count <= maxWert; count++)
{
    summe = summe + count    // Syntaxfehler: Anweisungsende!
}

// Typfehler ausgelöst durch die Variable zahl: Datentyp String!
// Variable ergebnis nicht deklariert.
ergebnis = summe * zahl;
```

Leibniz Universität Hannover

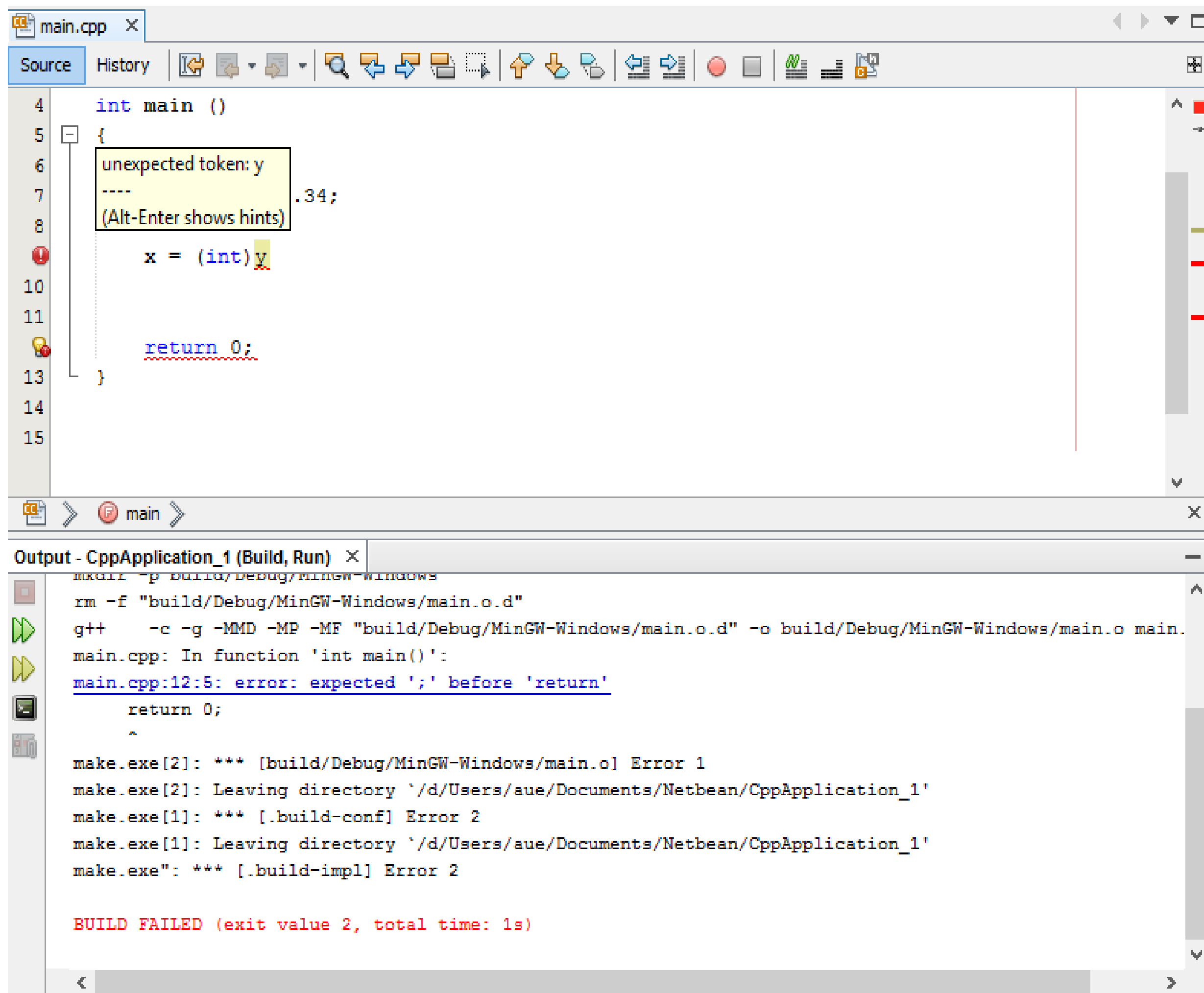
Programm ohne Fehler

```
const int maxWert = 10;
int summe = 0;
int zahl = 5;
int ergebnis = 0;

for(int count; count <= maxWert; count++)
{
   summe = summe + count;
}

ergebnis = summe * zahl;
```

… in NetBeans



Leibniz Universität Hannover

Erläuterung

- Die Syntax- und Typfehler werden mit einem roten Kreis am linken Rand und einem Strich am rechten Rand gekennzeichnet.
- Wenn der Mauszeiger über dem roten Kreis liegt, wird ein Tool-Tip zum Fehler als Hilfe angezeigt.
- Hinweis: Mit Hilfe von *View – Show Line Numbers* können die Zeilennummern eingeblendet werden.

Logische Fehler

- Fehlerhaftes Design der Software.
- Treten bei der Ausführung der Software auf.
- Durch Debuggen und Testen werden logische Fehler aus dem Programm entfernt.

Leibniz Universität Hannover

Beispiel

```
int main (){
    int x = 0;
    double y = 5.34;

    if (x = 0){
      y = 0;
    }
    else{
        x = (int)y;
    }
```

Weitere Beispiele

- Falsche Anzahl von Schleifendurchläufen.
- Falsch formulierte Bedingungen in Anweisungen und Schleifen.
- Falsche oder nicht vorhandene Klammerung von komplexen Ausdrücken.
- Falsch initialisierte oder gar nicht initialisierte Variablen.

Debugging mit Hilfe des Gnu-Compilers

- *g++ -g helloWorld.cpp -o helloWorld* hängt eine Symboltabelle für die Variablen sowie Zeileninformationen an die Objektdatei und die ausführbare Datei an. Die Größe der Datei verändert sich dadurch.
- Die Option -s würde diese Informationen wieder entfernen.

Leibniz Universität Hannover

Debugger in NetBeans

testString - NetBeans IDE 8.0.2

```
 8  int main()
 9  {
10      const int maxWert = 10;
11      int summe = 0;
12      int zahl = 5;
13      int ergebnis = 0;
14
15      for(int count; count <= maxWert; count++)
16      {
17          summe = summe + count;
18      }
19
20      ergebnis = summe * zahl;
```

Name	Value
<Enter new watch>	
maxWert	0
summe	0
zahl	4201600
ergebnis	4201694

Leibniz Universität Hannover

Ausgabefenster für den Debug-Modus

- Rechter Mausklick auf das C++-Projekt.
- *Properties* im Kontextmenü. Kategorie *Run*.
- *Console Type* sollte den Eintrag *External Terminal* haben. Andernfalls wird ein Warnung ausgegeben.

Leibniz Universität Hannover

… starten

- *Debug – Debug Project.* Das Programm wird zum Debuggen kompiliert.
- *Debug – Step Into.* Das Programm wird im Einzelschritt-Modus gestartet.
- Der nächste auszuführende Schritt wird farbig hinterlegt.

Schrittweise ausführen

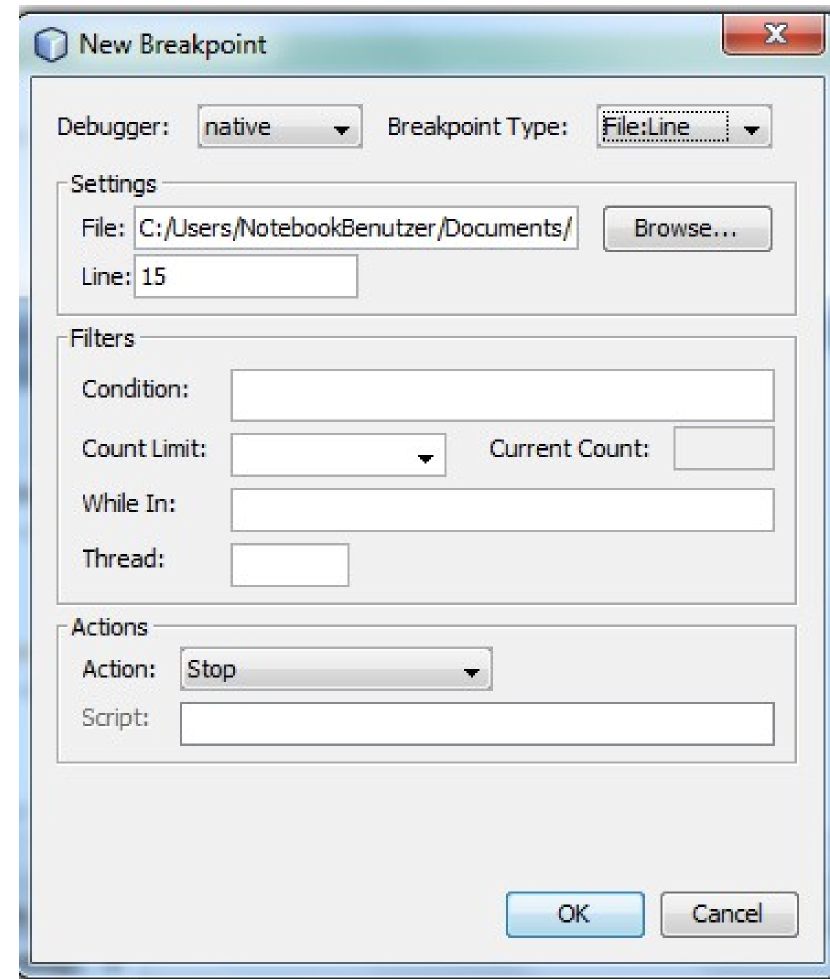
- Hinweis: Die nachfolgenden Befehl können über das Menü *Debug* oder die entsprechende Symbolleiste ausgewählt werden.
- *Step over* führt die aktuelle Anweisung aus und wechselt zur nächsten.
- *Run to Cursor* führt alle Anweisungen im Codefenster bis zu der Zeile, in der die Einfügemarke steht, aus.
- *Continue* führt das Programm ab, der aktuellen Anweisung vollständig aus.

… neu starten und beenden

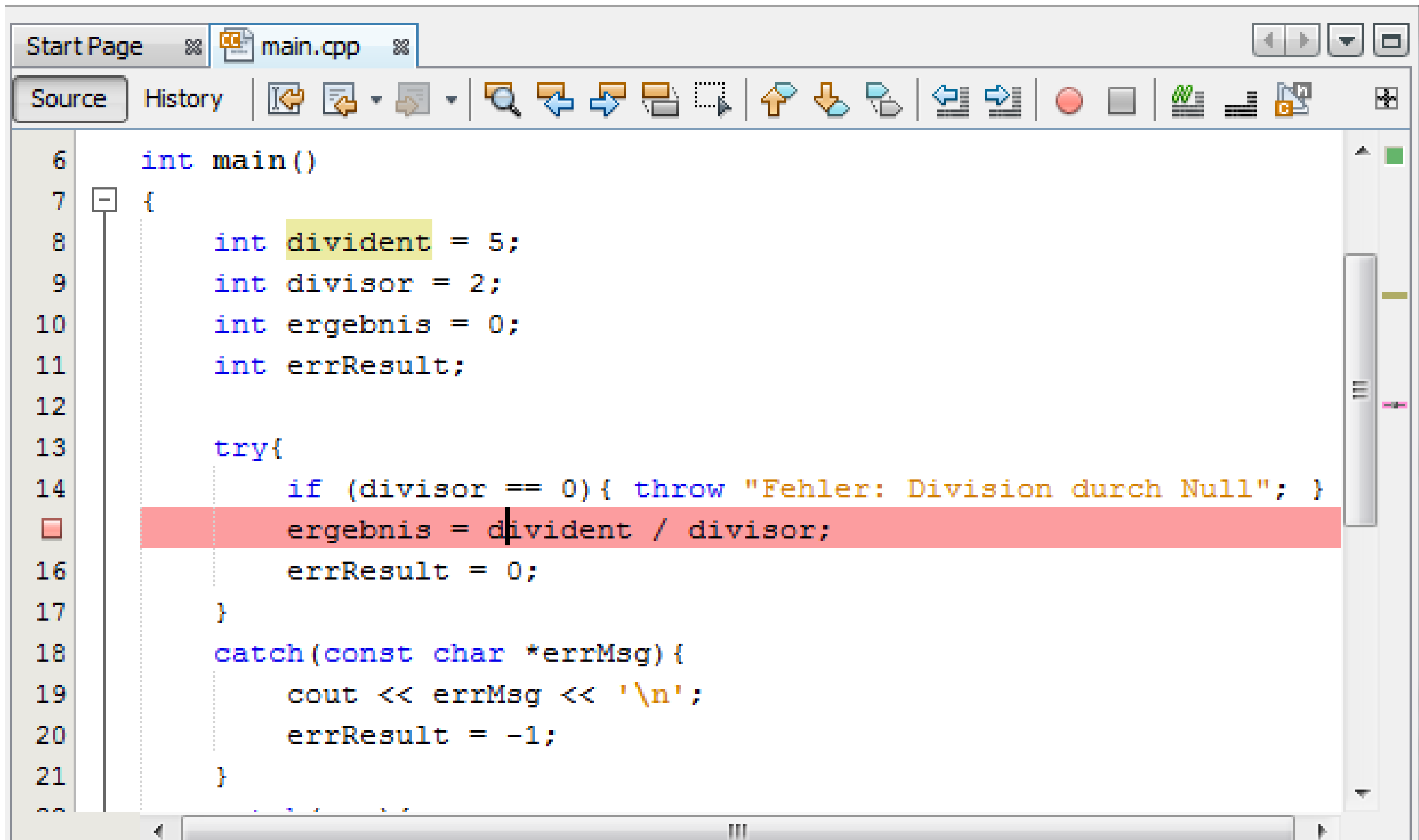
- *Debug – Restart.* Das Programm wird erneut gestartet.
- *Debug – Finish Debugger-Session* beendet den Debug-Modus.

Haltepunkte in einer Zeile

- *Debug – New Breakpoint*.
- Als *Breakpoint Type* wird der Typ *File:Line* genutzt. Der Haltepunkt wird in einer Zeile des Codes gesetzt.
- In dem Textfeld *File* wird der Name der Datei angezeigt, in der der Haltepunkt gesetzt werden soll.
- In dem Textfeld *Line* wird die Zeilennummer angezeigt.
- Standardmäßig wird die aktuelle Zeile in der aktiven Datei angezeigt,

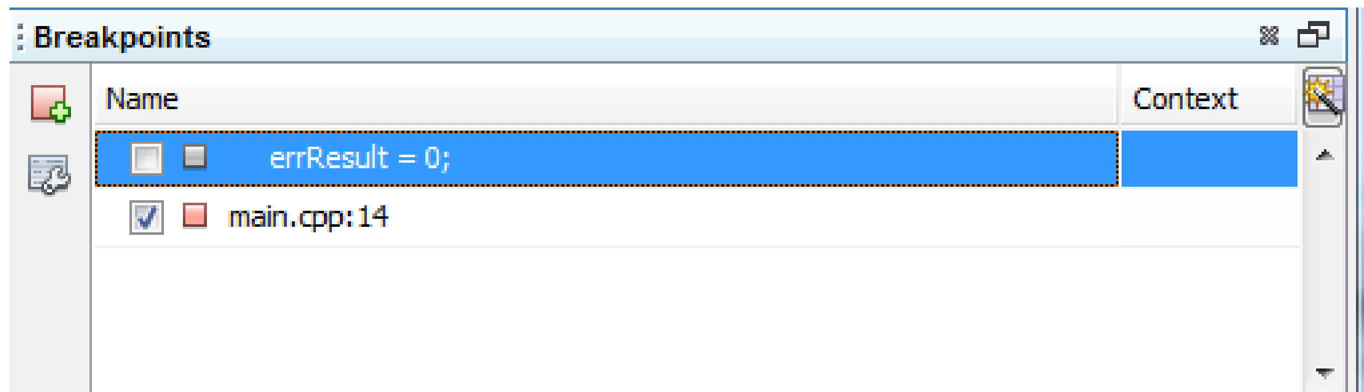


Haltepunkte im Codefenster



Anzeige von allen Haltepunkten

- Mit Hilfe von *Windows – Debugging – Breakpoints* wird ein Fenster, in dem alle Halltepunkte angezeigt werden, geöffnet.
- Mit Hilfe des Kontrollkästchens, links von jeden Breakpoint, kann ein Haltepunkt aktiviert oder deaktiviert werden.

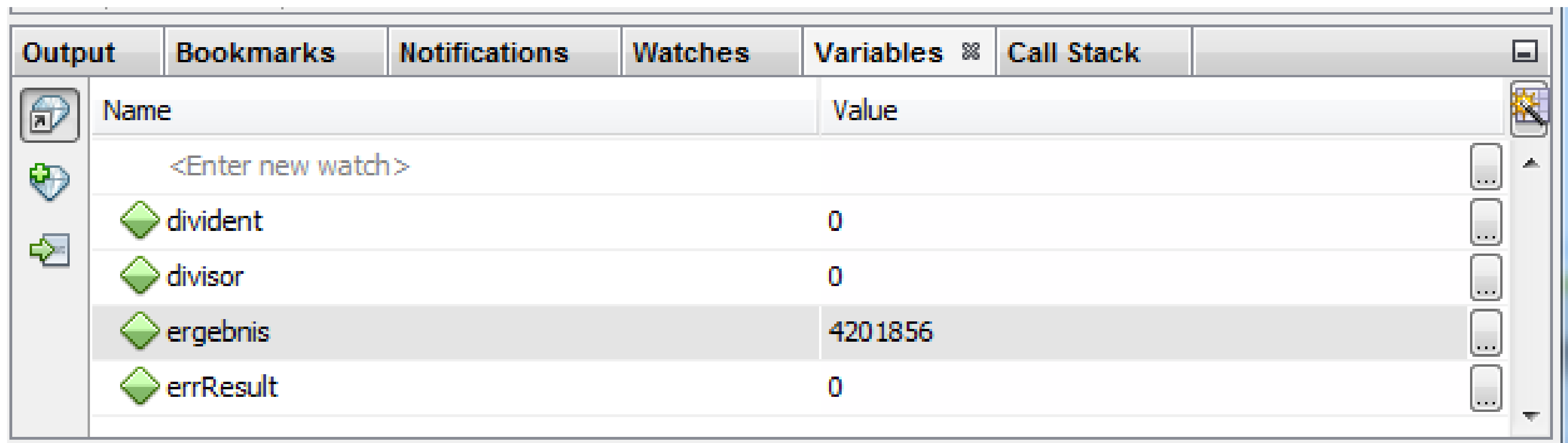


Löschen von Haltepunkten

- im Codefenster: Klick auf die Markierung am linken Rand des Haltepunktes.
- im Dialogfenster *Breakpoints*: Rechter Mausklick auf den zu löschenden Haltepunkte. Auswahl des Menübefehls *Delete* im Kontextmenü. Mit Hilfe des Menübefehls *Delete all* werden alle Haltepunkte gelöscht.

Werte der lokalen Variablen

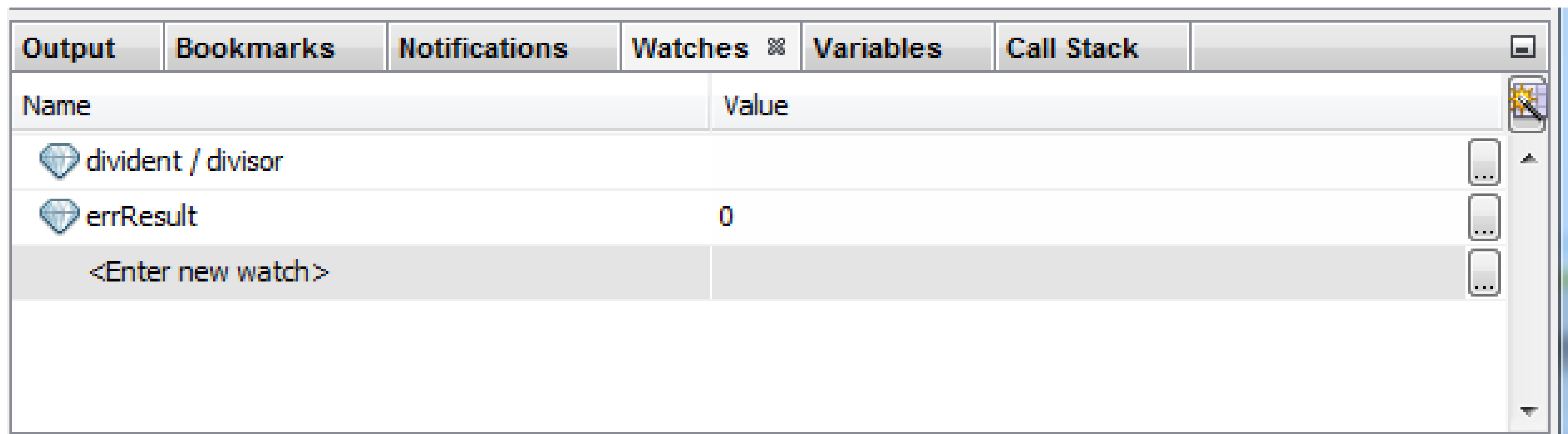
- Mit Hilfe von *Windows – Debugging – Variables* wird ein Fenster, in dem alle lokalen Variablen angezeigt werden, geöffnet.
- Zu jeder Variablen wird der aktuelle Wert angezeigt.



Leibniz Universität Hannover

Überwachung von Ausdrücken

- Mit Hilfe von *Windows – Debugging – Watches* wird das Überwachungsfenster angezeigt.
- In dem Fenster können Ausdrücke, Variablennamen etc. eingegeben werden, die überwacht werden sollen.



Laufzeitfehler

- Ausdrücke oder Anweisungen werden vom Programm nicht korrekt ausgewertet.
- Fehler, die zur Laufzeit des Programms, ein nicht erwünschtes Verhalten des Programms erzeugen.
- Das Programm kann abstürzen.

Exception-Handling in C++

```
int main(){
    try{
        if (divisor == 0){ throw "Fehler: Division durch Null"; }
        ergebnis = divident / divisor;
        errResult = 0;
    }
    catch(...){
        cout << "Unbekannter Fehler" << '\n';
        errResult = -1;
    }
    return errResult;
}
```

Beispiele/_1101_Exception...

Ablauf „Kein Fehler“

```
try{
        ergebnis = divident / divisor;
        errResult = 0;
    }
```

```
    return errResult;
```

Leibniz Universität Hannover

Ablauf „Fehler: Division durch Null“

```
int main(){
    try{
        if (divisor == 0){
            throw "Fehler: Division durch Null";
        }
    }
```

```
    catch(const char *errMsg){
        cout << errMsg << '\n';
        errResult = -1;
    }
```

```
    return errResult;
```

Leibniz Universität Hannover

Ablauf „Unbekannter Fehler“

```
int main(){
    try{
        // Abbruch in der Zeile, die den Fehler verursacht
    }
```

```
    catch(...){
        cout << "Unbekannter Fehler" << '\n';
        errResult = -1;
    }
```

```
    return errResult;
```

Leibniz Universität Hannover

Versuche die Anweisung auszuführen

```
try{
        ergebnis = divident / divisor;
        errResult = 0;
}
```

- Der Anweisungsblock try fasst Anweisungen zusammen, die einen Fehler verursachen können.
- Die Schachtelung von try-Anweisungen ist möglich.

Leibniz Universität Hannover

Werfe einen Fehler ...

```
throw "Fehler: Division durch Null";
```

- throw parameter.
- Löse manuell einen Fehler aus.
- In Abhängigkeit des Parametertyps wird der Fehler mit Hilfe von catch-Anweisungen abgefangen.

Fange den Fehler ab

```
catch(...){
    cout << "Unbekannter Fehler" << '\n';
    errResult = -1;
}
```

- Der Anweisungsblock catch versucht den dazugehörigen Laufzeitfehler zu beheben.
- Exception-Handler.
- Jedem try-Anweisungsblock folgt mindestens eine catch-Anweisungsblock.

Leibniz Universität Hannover

Kopf einer catch-Anweisung

catch	(	...	)
catch	(	const char *errMsg	)
catch	(	Parameter	)

- In Abhängigkeit des Parameters wird der catch-Anweisung einem Fehler zugeordnet.
- Der Parameter ist konstant.

Auswahl der catch-Anweisungsblöcke

- Entsprechend dem Aufruf von überladenden Funktion, wird die erste, zu der Exception passende, catch-Anweisung aufgerufen. Alle anderen möglichen Exception-Handler werden ignoriert. Das heißt, es wird maximal eine catch-Anweisung ausgeführt.
- Falls keine passende catch-Anweisung vorhanden ist, wird das Programm unkontrolliert abgebrochen.

Default-Anweisung

```
catch(...){
    cout << "Unbekannter Fehler" << '\n';
    errResult = -1;
}
```

- Drei Punkte, direkt hintereinander geschrieben, kennzeichnen den Default-Exception-Handler.
- Diese Anweisung steht wie bei einer switch-Anweisung am Ende aller Fälle.
- Es werden alle Fehler abgefangen, die nicht explizit behandelt wurden.

Fange „Zeiger von Char“ ab

```
catch(const char *errMsg){
    cout << errMsg << '\n';
    errResult = -1;
}
```

Beispiele/_1101_Exception...

- In dieser catch-Anweisungen werden Arrays vom Datentyp char abgefangen.
- Mit Hilfe des Zeigers wird ein Verweis auf die Fehlermeldung übergeben.

Leibniz Universität Hannover

Werfe einen Fehler vom Typ „Zeiger von Char“

```
throw "Fehler: Division durch Null";
```

- throw parameter.
- Der Parameter kann durch eine catch-Anweisung für einen String abgefangen werden.
- Jeder String ist ein Array von char. Strings können mit einem Zeiger auf das Array abgefangen werden.

Fange einen konstanten String ab

```
catch(const string errMsg){
    cout << errMsg << '\n';
    errResult = -1;
}
```

Beispiele/_1102_Exception...

- In dieser catch-Anweisungen werden Fehler vom Datentyp String abgefangen.
- Häufig wird der catch-Anweisung die auszugebende Fehlermeldung als String übergeben.

Werfe einen Fehler vom Typ String

```
const string strError = "Fehler: Division durch Null";

try{
    if (divisor == 0){
        throw strError;
```

- Verschiedenen Fehlermeldungen werden als konstante Strings deklariert.
- Die konstante Variable vom Typ String wird in diesem Beispiel geworfen.

Fange ein Fehlercode vom Typ „int“ ab

```
catch(const int errCode){
        switch(errCode){
            case 1:
                errMsg = "Fehler: Division durch Null.";
                break;
            default:
                errMsg = "Nicht bekannter Fehler";
        }

        cout << errMsg << '\n';
        errResult = -1;
}
```

Beispiele/_1103_Exception...

Werfe einen Fehler vom Typ „int“

```
const int errCode = 1;
int errResult;
string errMsg;

try{

    if (divisor == 0){
        throw errCode;
```

- In diesem Beispiel wird ein Error-Code vom Datentyp Integer geworfen.
- Jedes Integer symbolisiert einen Fehler.

Besser: Enumeration für den Fehlercode nutzen

```
enum errCode
{
   DIVISION_WITH_NULL,
   OUT_OF_BORDER,
   NUMBER_TO0_BIG,
   NUMBER_TOO_SMALL,
   BAD_INPUT
};
```

Beispiele/_1104_Exception...

Enumeration

- Benutzerdefinierter Aufzählungstyp.
- Definition von logisch zusammenhängenden Konstanten.

Leibniz Universität Hannover

Kopf einer Enumeration

enum	errCode
enum	Name

- Der Name identifiziert eindeutig eine Enumeration.
- Der Name sollte die, darin enthaltenen Variablen widerspiegeln.

Leibniz Universität Hannover

Konstanten in einer Enumeration

```
enum errCode
{
   DIVISION_WITH_NULL,
   OUT_OF_BORDER,
   NUMBER_TO0_BIG,
   NUMBER_TOO_SMALL,
   BAD_INPUT
};
```

- Innerhalb der geschweiften Klammern werden die benötigten Konstanten definiert.
- Die Konstanten im Rumpf der Enumeration werden durch ein Kommata getrennt.

Name der Konstanten

DIVISION_WITH_NULL		
DIVISION_WITH_NULL	=	110
VARIABLE	=	value

- Der Name identifiziert eindeutig eine Konstante in einer Enumeration.
- Der Name sollte das Element in der Liste widerspiegeln.
- Die Namen werden häufig groß geschrieben.

Wert einer Konstanten

DIVISION_WITH_NULL		
DIVISION_WITH_NULL	=	110
VARIABLE	=	value

- Standardmäßig werden die Variablen in einer Enumeration von 1 bis n durchnummeriert.
- Mit Hilfe des Gleichheitszeichens kann der Variablen eine Wert vom Datentyp „Ganzzahl“ übergeben werden. Die darauf folgende Variable hat den Wert plus eins.

Variable vom Typ „Enumeration“

```
errCode fehler;
```

- Variablen vom Datentyp Enumeration werden genauso wie alle anderen Variablen deklariert.

Leibniz Universität Hannover

Parameter: Enumeration

```
catch(const errCode fehler){
    switch(fehler){
      case DIVISION_WITH_NULL:
        errMsg = "Fehler: Division durch Null.";
        break;
      default:
        errMsg = "Nicht bekannter Fehler";
    }

    cout << errMsg << '\n';
    errResult = -1;
}
```

Beispiele/_1104_Exception...

Werfe einen Fehler vom Typ

```
try{

    if (divisor == 0){
        throw DIVISION_WITH_NULL;
    }
```

- In diesem Beispiel wird der Name einer Konstanten, die in einer Enumeration definiert ist, geworfen.

Leibniz Universität Hannover

Eingabefehler werfen

```
try{
    cout << "Bitte geben Sie eine Divident ein: \n";
    cin >> divident;

    if (cin.fail()) {
        throw BAD_INPUT;
    }

    cin.clear();
    cin.ignore(std::numeric_limits<int>::max(),'\n');
```

Beispiele/_1105_Exception...

Erläuterung

- Die Methode cin.fail() liefert true zurück, wenn die Eingabe fehlerhaft war. Hinweis: Eine Eingabe, die mit Zahlen beginnt, ist nicht fehlerhaft.
- Der Fehlerstatus wird mit Hilfe der Methode cin.clear() zurückgesetzt.
- Mit Hilfe der Funktion cin.ignore(std::numeric_limits<int>::max(),'\n'); werden die, im Buffer vorhandenen Zeichen, verworfen,