Object-oriented Programming
for Automation & Robotics

Carsten Gutwenger

LS 11 Algorithm Engineering

Lecture 12 • Winter 2011/12 • Jan 17

technische universität
dortmund

department of
computer science

How to debug programs with Visual Studio?

Preliminary remarks:

- This lecture targets Visual Studio 2010
  (not 2008 as installed on the pool computers)
- Screenshots are from the German version (sorry!)
- Most discussed features and guidelines also apply to other development environments (IDEs)
- Reference (MSDN):
  http://msdn.microsoft.com/en-us/library/sc65sadd.aspx

General procedure for developing programs

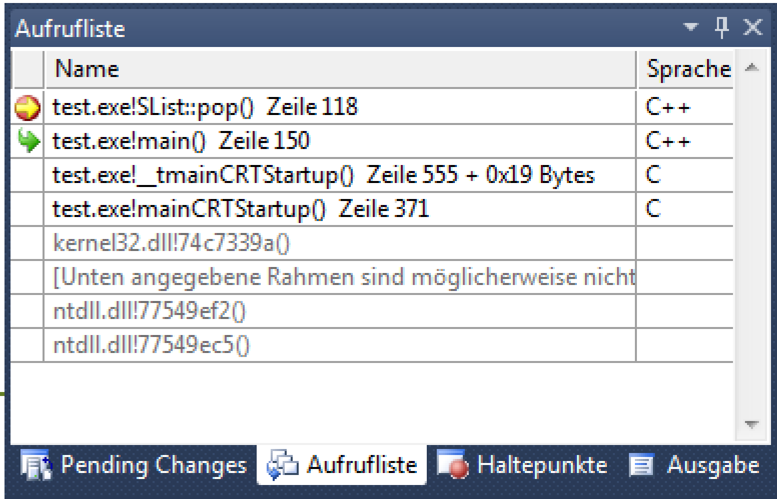
1. Build your program in **Debug** configuration.
2. Fix all compiler errors.
3. Do not ignore compiler warnings!
   Warnings usually point you to potential problems in your code; try to fix all warnings.
4. Test your program for correctness.
   If errors / crashes / wrong results occur → **Debugging**
   If you have to modify your program, go back to 1.
5. Build your program in **Release** configuration.
   (In the rare case of compiler errors or warnings, fix them too.)
6. Test your program with respect to runtime (and correctness).

Programs built in **Debug** configuration contain additional information used by the debugger.

Programs built in **Release** configuration are highly optimized for speed.

Main Features of the Debugger

- **Break execution**
  - when an error occurs
  - on request (anytime during execution)
  - at **breakpoints**
- **Stepping through the code**
  - line by line
  - step into function calls
  - resume execution until the current function returns
  - run to cursor (resume execution until the program reaches the cursor location)
- **Viewing Data**
  - show the value of variables
  - evaluate (simple) expressions
  - navigating through the program's **call stack**



	Name	Sprache
➡	test.exe!SList::pop() Zeile 118	C++
➡	test.exe!main() Zeile 150	C++
	test.exe!__tmainCRTStartup() Zeile 555 + 0x19 Bytes	C
	test.exe!mainCRTStartup() Zeile 371	C
	kernel32.dll!74c7339a()	
	[Unten angegebene Rahmen sind möglicherweise nicht	
	ntdll.dll!77549ef2()	
	ntdll.dll!77549ec5()	

How to: Start Debugging

1. Build your program in Debug configuration.
2. On the **Debug** menu, choose **Start Debugging** (or press **F5**).

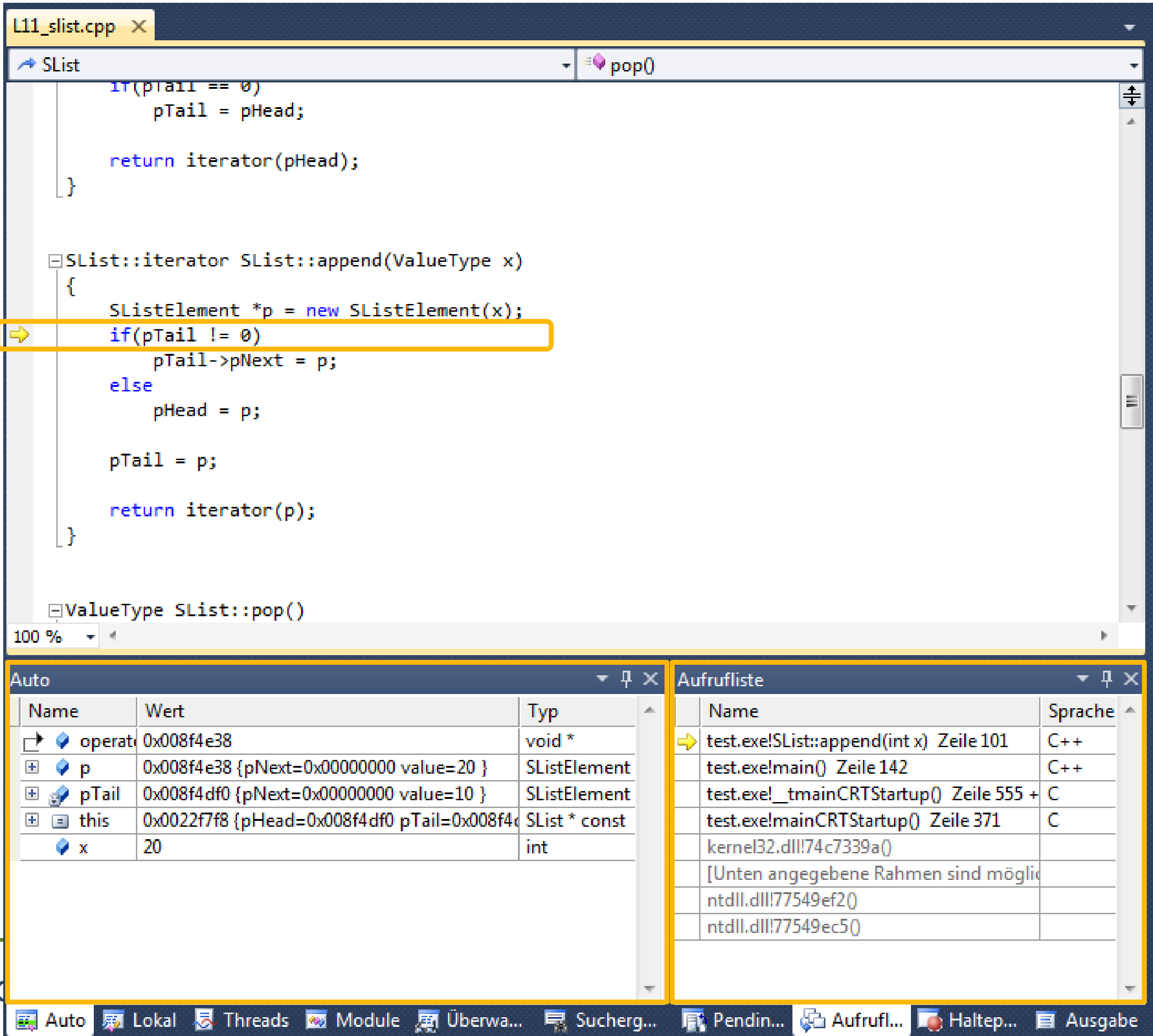
The programs runs until

- you choose **Stop Debugging** (**SHIFT+F5**) on the **Debug** menu
  → program is aborted
- you choose **Break All** on the **Debug** menu
  → program just stops in debugger
- a breakpoint is reached
  → program just stops in debugger
- a runtime error (exception) occurs
  → a dialog box appears which allows you to jump into the debugger
- the program is finished

In the Debugger…

When your program stops in the debugger, you can

- see the current line being executed
- execute the program step by step
- navigate through the call stack
- display variables and evaluate expressions



```
    if(pTail == 0)
        pTail = pHead;

    return iterator(pHead);
}


SList::iterator SList::append(ValueType x)
{
    SListElement *p = new SListElement(x);
    if(pTail != 0)
        pTail->pNext = p;
    else
        pHead = p;

    pTail = p;

    return iterator(p);
}


ValueType SList::pop()
```

Auto

Name	Wert	Typ
operat	0x008f4e38	void *
p	0x008f4e38 {pNext=0x00000000 value=20 }	SListElement
pTail	0x008f4df0 {pNext=0x00000000 value=10 }	SListElement
this	0x0022f7f8 {pHead=0x008f4df0 pTail=0x008f4	SList * const
x	20	int

Aufrufliste

Name	Sprache
test.exe!SList::append(int x) Zeile 101	C++
test.exe!main() Zeile 142	C++
test.exe!__tmainCRTStartup() Zeile 555 +	C
test.exe!mainCRTStartup() Zeile 371	C
kernel32.dll!74c7339a()	
[Unten angegebene Rahmen sind mögli	
ntdll.dll!77549ef2()	
ntdll.dll!77549ec5()	

Stepping through the Code

The **Debug** menu and toolbar provide the following commands:

- **Step Into** (**F11**)
  - Executes the next line of code.
  - If this line contains a function call, execution halts again at the beginning of that function.
- **Step Over** (**F10**)
  - Same as Step Into except for function calls (executes the entire function, then stops at first line outside the function).
- **Step Out** (**SHIFT+F11**)
  - Resumes execution until the current function returns.
  - Breaks at the return point in the calling function.

Stepping through the Code

The **Debug** menu and toolbar provide the following commands:

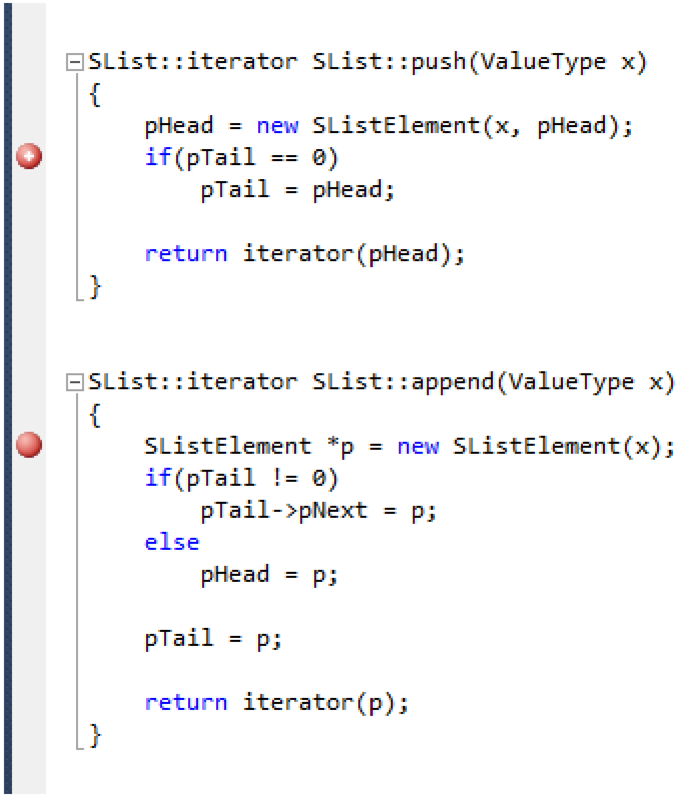
- **Run To Cursor** (**CTRL+F10**)
  - resumes execution until a specified line is reached
  - right-click a line and choose **Run to Cursor**; or
    move the cursor to the line and press **CTRL+F10**
  - if any breakpoint is hit before the line is reached, execution will stop at the breakpoint
- **Continue** (**F5**)
  - resumes execution
- **Stop Debugging** (**SHIFT+F5**)
  - aborts program

Breakpoints

- Breakpoints allow to stop execution, when a particular line of code is reached.
- **Conditional breakpoints**
  - based on an expression like: `pTail == 0`
    (here `pTail` is a variable in the program)
    execution is only stopped at the breakpoint if the condition evaluates to true
  - based on the current hit count (how many times the breakpoint was hit)
    execution only stops when the current hit count
    - equals a specified value
    - is ≥ a specified value
    - is a multiple of a specified value

How to: Set Breakpoints

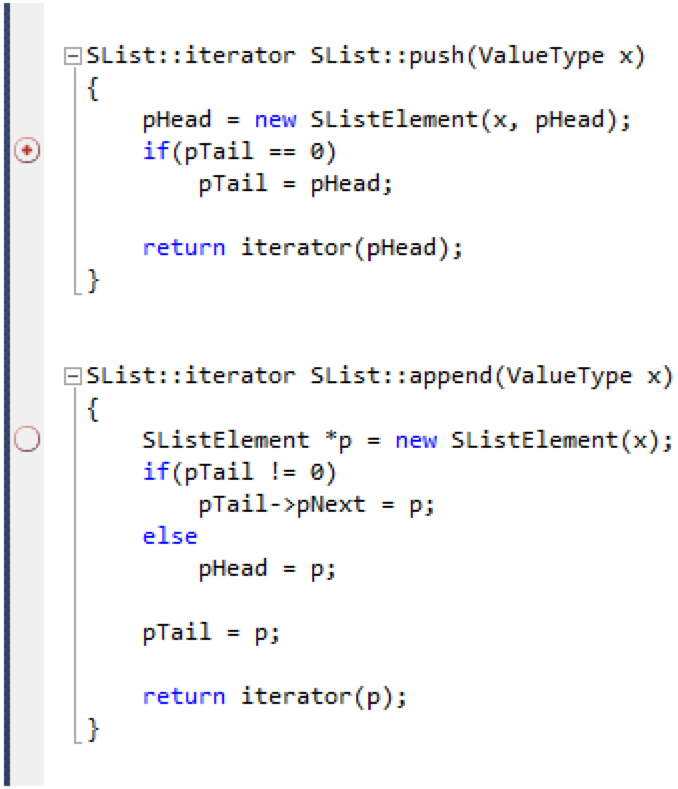
- Set a breakpoint:
  - click in the grey left column
  - right-click on a line and choose **Breakpoint → Insert Breakpoint**
  - choose **Toggle Breakpoint** (**F9**) from the **Debug** menu
- Delete a breakpoint:
  - click on the breakpoint symbol
  - right-click a breakpoint and choose **Delete** from the shortcut menu
  - choose **Toggle Breakpoint** (**F9**) from the **Debug** menu
- Delete all breakpoints:
  - choose **Delete All Breakpoints** (**CTRL+SHIFT+F9**) from the **Debug** menu



```
SList::iterator SList::push(ValueType x)
{
    pHead = new SListElement(x, pHead);
    if(pTail == 0)
        pTail = pHead;

    return iterator(pHead);
}

SList::iterator SList::append(ValueType x)
{
    SListElement *p = new SListElement(x);
    if(pTail != 0)
        pTail->pNext = p;
    else
        pHead = p;

    pTail = p;

    return iterator(p);
}
```

How to: Enable / Disable Breakpoints

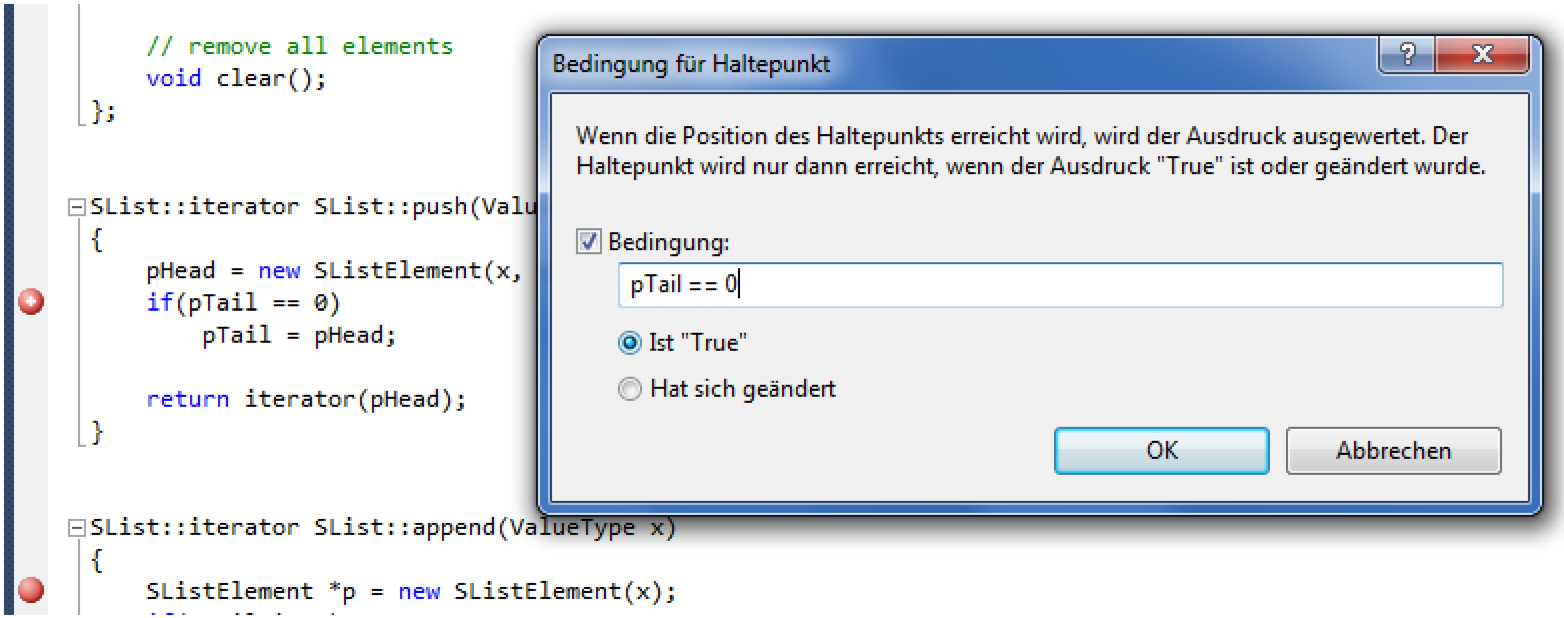
- Sometimes, you just want to disable breakpoints temporarily
- Disable a breakpoint:
  - right-click a breakpoint and choose **Disable Breakpoint** (**CTRL+F9**) from the shortcut menu
- Enable a breakpoint:
  - right-click a breakpoint and choose **Enable Breakpoint** (**CTRL+F9**) from the shortcut menu



```
SList::iterator SList::push(ValueType x)
{
    pHead = new SListElement(x, pHead);
    if(pTail == 0)
        pTail = pHead;

    return iterator(pHead);
}

SList::iterator SList::append(ValueType x)
{
    SListElement *p = new SListElement(x);
    if(pTail != 0)
        pTail->pNext = p;
    else
        pHead = p;

    pTail = p;

    return iterator(p);
}
```

- Enable or disable all breakpoints:
  - Chose **Enable** (**Disable**) **All Breakpoints** from the **Debug** menu

How to: Specify a Breakpoint Condition



```
    // remove all elements
    void clear();
};

SList::iterator SList::push(Valu
{
    pHead = new SListElement(x,
    if(pTail == 0)
        pTail = pHead;

    return iterator(pHead);
}

SList::iterator SList::append(ValueType x)
{
    SListElement *p = new SListElement(x);
```

Bedingung für Haltepunkt

Wenn die Position des Haltepunkts erreicht wird, wird der Ausdruck ausgewertet. Der Haltepunkt wird nur dann erreicht, wenn der Ausdruck "True" ist oder geändert wurde.

☑ Bedingung:

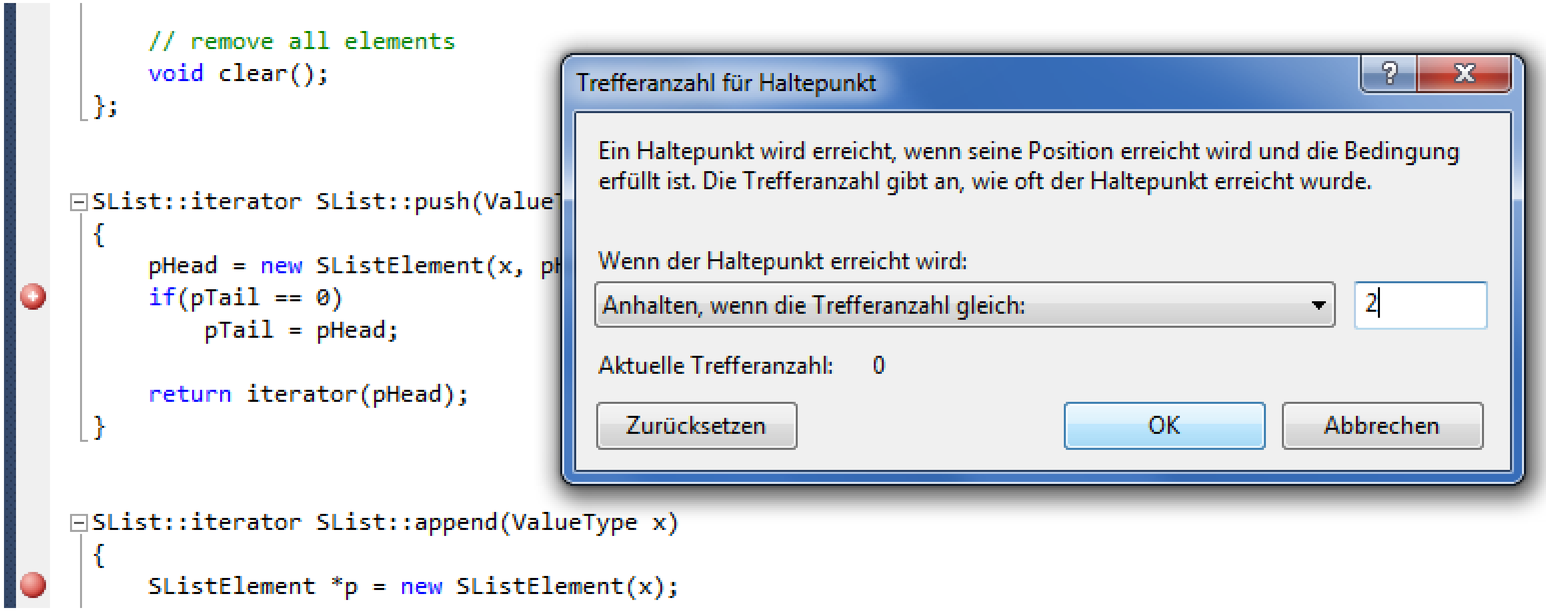
pTail == 0

● Ist "True"

○ Hat sich geändert

- Right-click a breakpoint and choose **Condition** from the shortcut menu.
  - In the dialog box, enter a valid expression.
    This expression may contain all variables visible at the breakpoint location.
  - Choose **Is true** if you want to break if the conditions is satisfied, or **Has changed** if you want to break when the condition has changed.

How to: Specify a Hit Count



```
    // remove all elements
    void clear();
};


SList::iterator SList::push(ValueT
{
    pHead = new SListElement(x, pH
    if(pTail == 0)
        pTail = pHead;

    return iterator(pHead);
}


SList::iterator SList::append(ValueType x)
{
    SListElement *p = new SListElement(x);
```

Trefferanzahl für Haltepunkt

Ein Haltepunkt wird erreicht, wenn seine Position erreicht wird und die Bedingung erfüllt ist. Die Trefferanzahl gibt an, wie oft der Haltepunkt erreicht wurde.

Wenn der Haltepunkt erreicht wird: Anhalten, wenn die Trefferanzahl gleich: 2

Aktuelle Trefferanzahl: 0

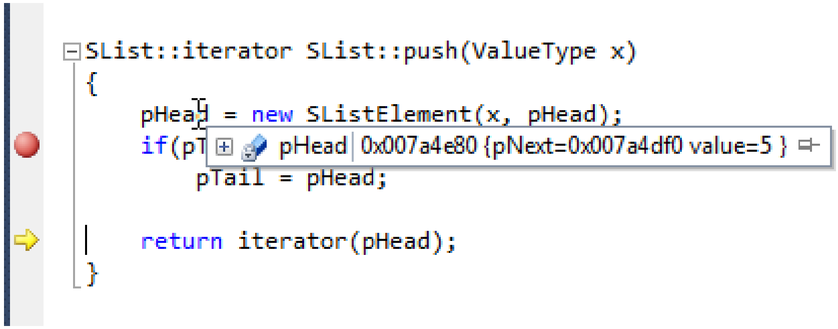
- Right-click a breakpoint and choose **Hit Count** from the shortcut menu.
  - In the dialog box, select the behavior from the **When the breakpoint is hit** list.
  - Enter an integer value in the text box (only visible if not **Break always** is selected)

Viewing Data

Various features allow you to view data during debugging:

- **DataTips**
  - tooltips that appear when you move the mouse pointer over a variable
  - very powerful since Visual Studio 2010
- **Variable Windows**
  - **Autos** Window
    - shows variables used in the current and preceding line of code, as well as return values of functions
  - **Locals** Window
    - shows variables local to the current context and scope
  - **Watch** Window
    - allows you to add variables and expressions you want to watch
- **QuickWatch** dialog box
  - a dialog box that works similar as the Watch window

How to: DataTips

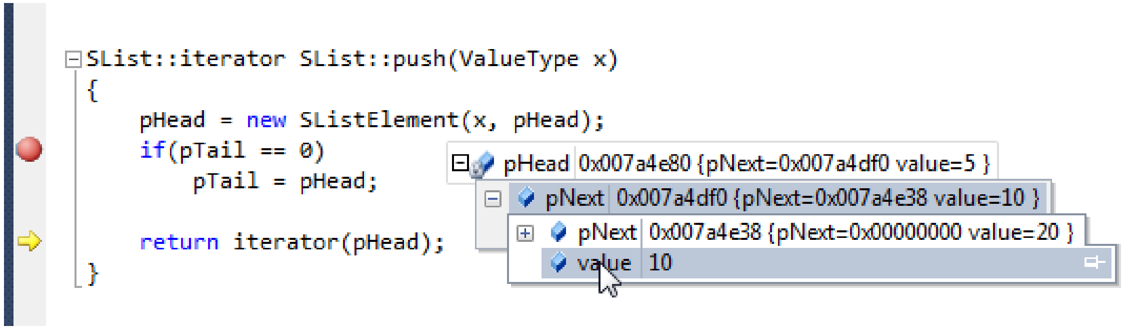


```
SList::iterator SList::push(ValueType x)
{
    pHead = new SListElement(x, pHead);
    if(pT
        pTail = pHead;

    return iterator(pHead);
}
```

```
pHead 0x007a4e80 {pNext=0x007a4df0 value=5 }
```

- Displaying a DataTip
  - move the mouse cursor over a variable symbol in the current scope
  - the DataTip disappears when you remove the mouse pointer
  - to pin the DataTip, click the **Pin to source** icon
- A pinned DataTip
  - can be dragged around in the source window
  - click the **Unpin from source** icon to make it float (then you can move it over other windows, too)
  - close a pinned DataTip by clicking the **Close** icon
- Close all DataTips
  - Choose **Clear All DataTips** from the **Debug** menu

How to: DataTips



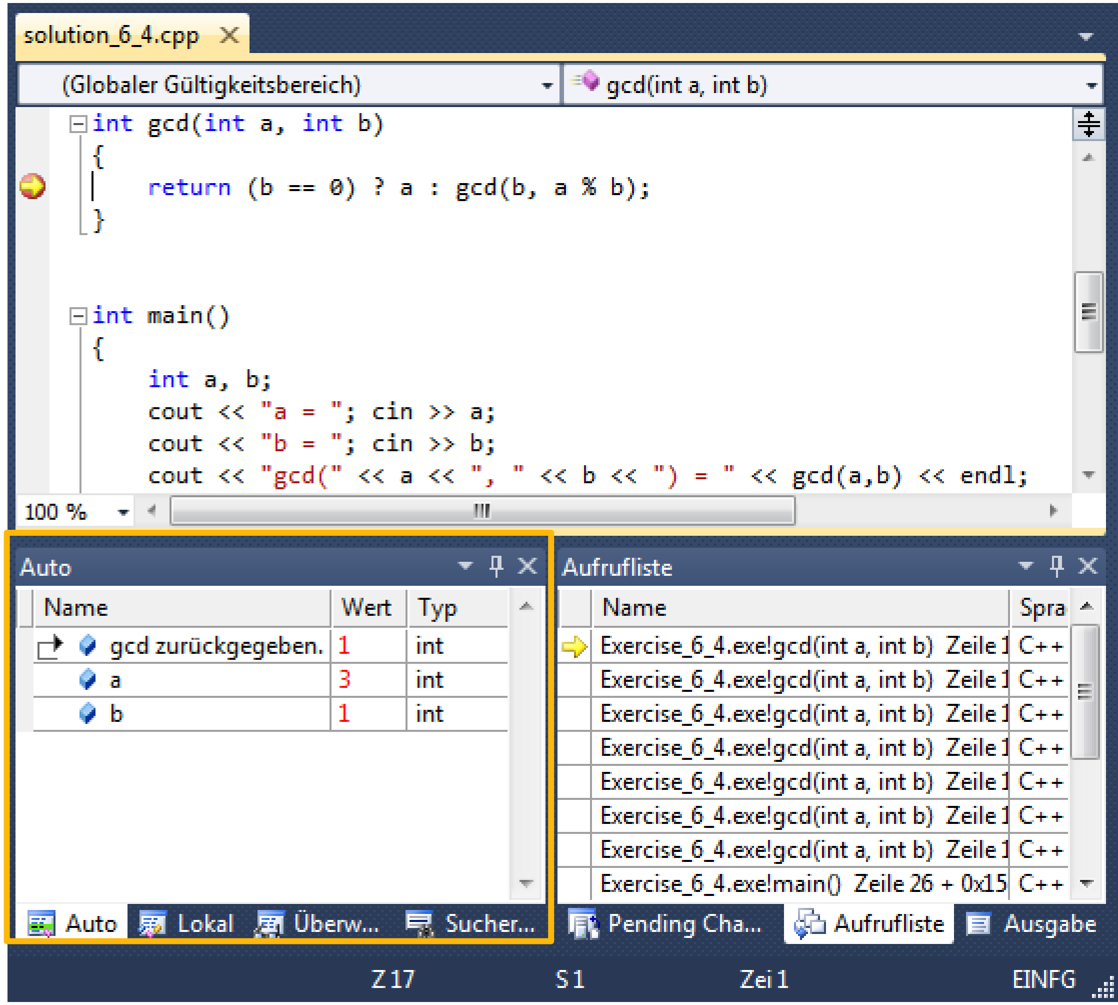
```
SList::iterator SList::push(ValueType x)
{
    pHead = new SListElement(x, pHead);
    if(pTail == 0)
        pTail = pHead;

    return iterator(pHead);
}
```

```
⊟ pHead   0x007a4e80 {pNext=0x007a4df0 value=5 }
  ⊟ pNext   0x007a4df0 {pNext=0x007a4e38 value=10 }
    ⊞ pNext   0x007a4e38 {pNext=0x00000000 value=20 }
      value   10
```

- DataTips also allow you to
  - expand variables (e.g. structures, pointers to structures)
    (use the **+** sign before the variable name)
  - edit the values of variables
    (click on the value and type a new value)
  - add expressions to pinned DataTips
    (right-click on the DataTip and choose **Add Expression** from the shortcut menu)

Variable Windows: Autos

- **Autos Window**
  - shows name, value, and type for currently interesting variables
  - you can expand variables, e.g. (pointers to) structures
  - you can edit the values of variables (double-click the value)
  - you can switch to another stack frame by double-clicking the corresponding row in the call stack window



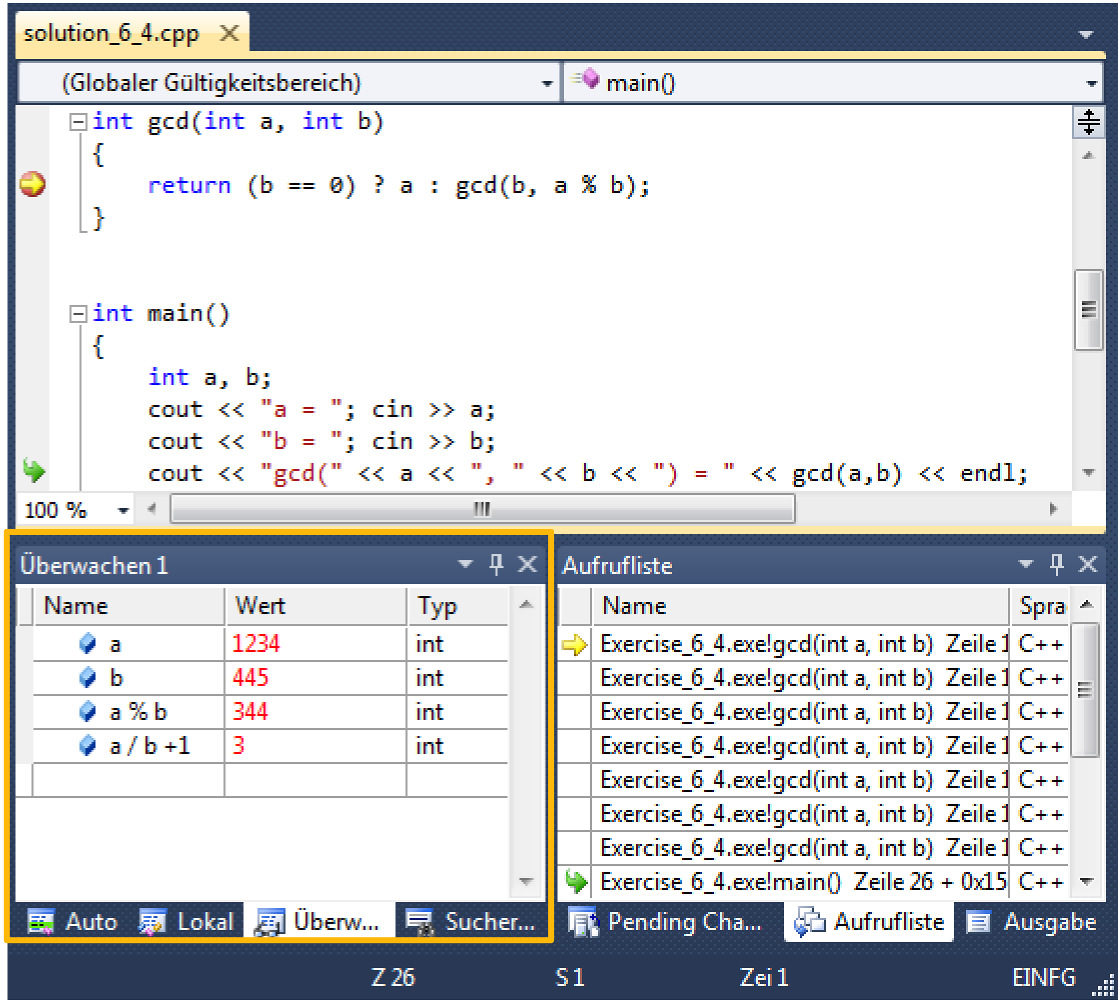
```
int gcd(int a, int b)
{
    return (b == 0) ? a : gcd(b, a % b);
}


int main()
{
    int a, b;
    cout << "a = "; cin >> a;
    cout << "b = "; cin >> b;
    cout << "gcd(" << a << ", " << b << ") = " << gcd(a,b) << endl;
```

Name	Wert	Typ
gcd zurückgegeben.	1	int
a	3	int
b	1	int

Name	Spra
Exercise_6_4.exe!gcd(int a, int b) Zeile 1	C++
Exercise_6_4.exe!gcd(int a, int b) Zeile 1	C++
Exercise_6_4.exe!gcd(int a, int b) Zeile 1	C++
Exercise_6_4.exe!gcd(int a, int b) Zeile 1	C++
Exercise_6_4.exe!gcd(int a, int b) Zeile 1	C++
Exercise_6_4.exe!gcd(int a, int b) Zeile 1	C++
Exercise_6_4.exe!gcd(int a, int b) Zeile 1	C++
Exercise_6_4.exe!main() Zeile 26 + 0x15	C++

Variable Windows: Watch

- **Watch Window**
  - shows name, value, and type for variables and expressions
  - you can add an expression by clicking into the last row
  - you can edit an expression by double-clicking on it
  - you can remove an expression by selecting the row and pressing **DEL**
  - otherwise behaves similar as the Autos window



```
int gcd(int a, int b)
{
    return (b == 0) ? a : gcd(b, a % b);
}


int main()
{
    int a, b;
    cout << "a = "; cin >> a;
    cout << "b = "; cin >> b;
    cout << "gcd(" << a << ", " << b << ") = " << gcd(a,b) << endl;
```

Überwachen 1

Name	Wert	Typ
a	1234	int
b	445	int
a % b	344	int
a / b +1	3	int

Aufrufliste

Name	Spra
Exercise_6_4.exe!gcd(int a, int b) Zeile 1	C++
Exercise_6_4.exe!gcd(int a, int b) Zeile 1	C++
Exercise_6_4.exe!gcd(int a, int b) Zeile 1	C++
Exercise_6_4.exe!gcd(int a, int b) Zeile 1	C++
Exercise_6_4.exe!gcd(int a, int b) Zeile 1	C++
Exercise_6_4.exe!gcd(int a, int b) Zeile 1	C++
Exercise_6_4.exe!gcd(int a, int b) Zeile 1	C++
Exercise_6_4.exe!main() Zeile 26 + 0x15	C++