Debuggen mit Code::Blocks

Um Fehlern im Programm (sogenannte *bugs* auf Englisch) einfacher auf die Spur zu kommen, kann man das Programm in einem sogenannten Debugger laufen lassen. Ein Debugger ermöglicht, das Programm an festgelegten Stellen zu stoppen, um sich den Wert von lokalen Variablen anzeigen zu lassen, und sich schrittweise durch das Programm zu bewegen.

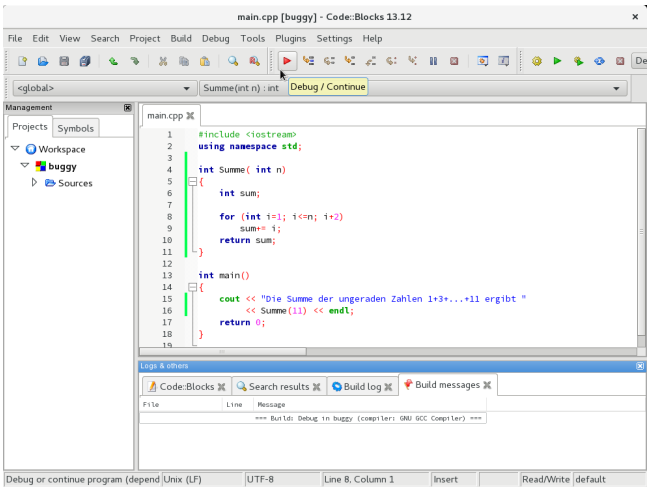
Anhand einer Beispielsitzung lernen wir die grundlegende Arbeitsweise eines Debuggers kennen und erfahren, wie wir den Debugger mithilfe von `Code::Blocks` steuern können. Wir gehen dabei von folgendem Programm aus, das die Summe aller ungeraden natürlichen Zahlen bis einschließlich 11 berechnen soll:

```
1  #include <iostream>
2  using namespace std;
3
4  int Summe( int n)
5  {
6      int sum;
7
8      for (int i=1; i<=n; i+2)
9          sum+= i;
10     return sum;
11 }
12
13 int main()
14 {
15     cout << "Die␣Summe␣der␣ungeraden␣Zahlen␣1+3+...+11␣ergibt␣"
16          << Summe(11) << endl;
17     return 0;
18 }
```

1. Legen Sie ein neues Projekt in `Code::Blocks` an und geben Sie dort als Datei `main.cpp` obigen Programmcode ein. Alternativ können Sie auch die Datei `buggy.cpp` aus dem Lernraum herunterladen und ihren Inhalt in das erstellte Projekt hineinkopieren.

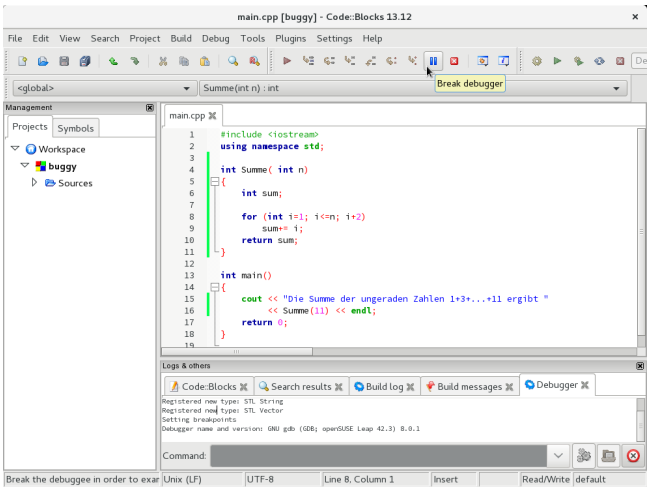
2. Das Programm lässt sich problemlos übersetzen. Bei der Ausführung stellen wir allerdings fest, dass nichts auf dem Bildschirm ausgegeben wird und stattdessen das Programm irgendwo „hängt“. Was passiert hier? Dabei kann uns der Debugger helfen!

3. In der Werkzeugleiste (oben) finden wir einen Button mit einem roten Play-Dreieck, der das Programm im Debugger ausführt („Debug / Continue“).

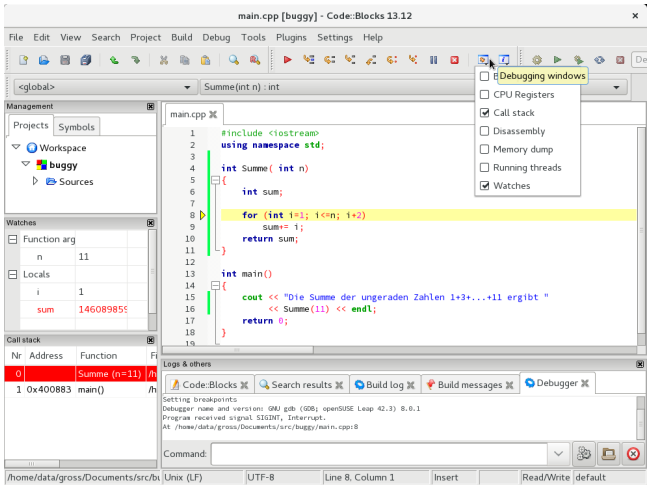


Rechts daneben finden sich weitere Buttons, die wir im Folgenden bei der Benutzung des Debuggers brauchen werden. Alternativ zum roten Play-Button können Sie auch das Menü („Debug > Start / Continue“) oder die Taste F8 benutzen. Wie zuvor beim Ausführen des Programms öffnet sich das Konsolenfenster, ohne dass das Programm irgendetwas tut.

4. Wir unterbrechen das Programm im Debugger mithilfe der blauen Pause-Taste („Break debugger“), um zu sehen, wo es sich gerade befindet und „hängt“. Alternativ können Sie auch das Menü („Debug > Break debugger“) benutzen.



5. Der Cursor springt in Zeile 8 auf die `for`-Schleife. Gleichzeitig zeigt ein gelbes Dreieck neben der Zeilennummer an, wo sich der Debugger gerade befindet.[1] Das Programm hängt also offensichtlich in dieser Schleife.

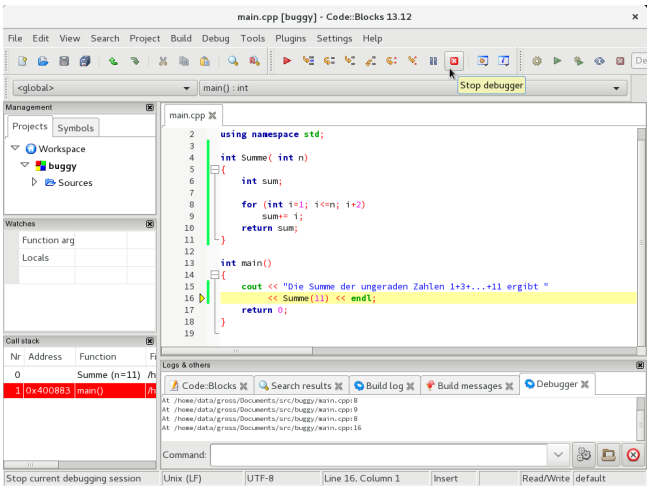


Da das Programm pausiert, können wir uns nun mithilfe des Debuggers einige Informationen anzeigen lassen. In der Reihe der Debug-Buttons finden Sie ganz rechts zwei Buttons mit Fenstersymbolen, eines mit einem Käfer (*bug*) und eines mit einem *i* (für Informationen). Drücken Sie den Käfer-Button und wählen Sie die Fenster „Call stack“ und „Watches“ aus. Alternativ können Sie dazu den Menüeintrag „Debug > Debugging windows“ benutzen. Sie können diese Fenster an eine beliebige Stelle in `Code::Blocks ziehen`, z.B. in die linke Leiste.

[1] Unter Windows funktioniert „Break debugger“ unter Umständen nicht wie gewünscht. In so einem Fall können Sie den Cursor in Zeile 8 bewegen und den Debugger dort mithilfe des Buttons „Run to cursor“ stoppen, wie unter Punkt 8 beschrieben.

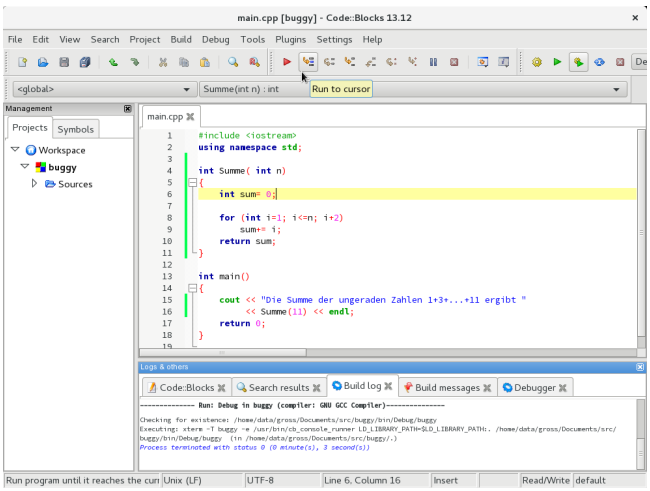
6. Im Fenster „Call stack“ ist zu sehen, dass der Debugger sich gerade in der Funktion `Summe (n=11)` befindet, die von der Funktion `main()` aufgerufen wurde. Im Fenster „Watches“ sind die Argumente und lokalen Variablen der aktiven Funktion `Summe` zu sehen. Uns fällt sofort der merkwürdige Wert von `sum` auf. Woran das wohl liegen mag?

   Wenn Sie im „Call stack“ die Funktion `main()` per Doppelklick auswählen, springt der Debugger an die Stelle in der Funktion `main()`, an der der Aufruf von `Summe (n=11)` erfolgt ist. Gleichzeitig zeigt das Fenster „Watches“ die lokalen Variablen der Funktion `main.cpp` zu diesem Zeitpunkt an. Bei komplexeren Programmen ist der Call stack entsprechend größer und ermöglicht die Orientierung, durch welche Funktionsaufrufe das Programm an die aktuelle Stelle gelangt ist und welchen Status die lokalen Variablen in den einzelnen Funktionskontexten haben.

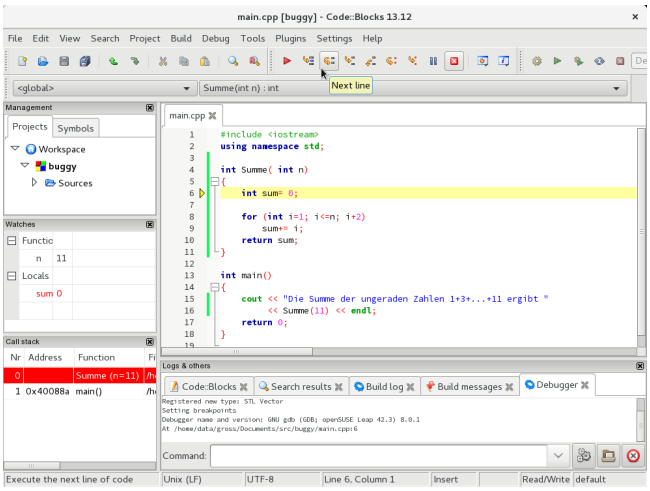


7. Natürlich haben Sie längst erkannt, warum die Variable `sum` einen solch komischen Wert enthält. Um das zu korrigieren, beenden Sie zunächst die Debugging-Session, indem Sie auf den roten Button mit dem Kreuz („Stop debugger“) drücken. Alternativ können Sie auch das Menü („Debug > Stop debugger“) oder die Tastenkombination Shift+F8 benutzen. Dann korrigieren Sie Ihr Programm, indem Sie in Zeile 6 die richtige Initialisierung vornehmen: `int sum= 0;`

8. Wenn wir das Programm erneut kompilieren und ausführen, erkennen wir, dass das Problem noch nicht behoben ist. Wir müssen also nochmals den Debugger bemühen. Diesmal starten wir den Debugger auf eine andere Weise, und zwar mithilfe des Buttons „Run to cursor“.



Das Programm läuft dann im Debugger an und unterbricht, sobald es die Zeile erreicht, auf der der Cursor steht, in unserem Fall also in Zeile 6.



9. Durch wiederholtes Drücken des Buttons „Next line“ (oder alternativ der Taste F7) können wir das Programm Zeile für Zeile durchgehen. Dabei verändern sich im Fenster „Watches“ schrittweise die Variablen. Es wird außerdem klar, dass die `for`-Schleife zwar beliebig oft durchlaufen wird, aber die Zählvariable `i` sich nicht ändert, sondern immer den Wert 1 behält. Das führt hier zu einer Endlosschleife, weil das Update des Zählers in Zeile 8 fehlerhaft ist. Wir korrigieren es, indem wir entweder `i= i+2` oder `i+= 2` schreiben. Nachdem wir den Debugger beendet und den Code erneut kompiliert und ausgeführt haben, verhält sich das Programm nun endlich so wie es soll.

10. Es gibt noch weitere Möglichkeiten, sich schrittweise durch das Programm zu bewegen. Befindet sich in der aktiven Zeile ein Funktionsaufruf, kann der Button „Step into“ dazu benutzt werden, um mit dem Debugger in die betreffende Funktion zu springen. Umgekehrt führt der Button „Step out“ dazu, wieder aus der Funktion in den aufrufenden Kontext zu springen und dort anzuhalten.

11. Eine weitere vielgenutzte Möglichkeit der Programmunterbrechung bieten sogenannte Breakpoints. Erreicht der Programmablauf einen solchen Breakpoint, so pausiert der Debugger jeweils an dieser Stelle. Breakpoints können gesetzt (und wieder gelöscht) werden, indem man rechts neben die Zeilennummer klickt. Alternativ können Sie dafür das das Menü („Debug > Toggle breakpoint“) oder die Taste F5 verwenden.