DIbugger: User Manual

Benedikt Wagner Chiara Staudenmaier Etienne Brunner
Joana Plewnia Pascal Zwick Ulla Scheler

Betreuer: Mihai Herda, Michael Kirsten

March 6, 2018

Contents

Contents

Chapter 1

Overview

1.1 Quick Start

The following chapters provide a detailed explanation of the functionality the DIbugger provides. If reading these pages proves too time-consuming for you or if you feel the exciting urge to use the DIbugger right away, here is the quick start guide. In it, we will refer to the user interface which is described in the next section.
Let's begin. We are assuming that you already opened the debugger and that you have two program texts you want to compare.

1. **Add your program texts**
   Make sure your code follows the requirements of the DIbugger language. (The preview text in the left program window provides an example.) Then, just write or paste your program texts into the text windows.

2. **Set some breakpoints**
   If you want the debugger to pause at certain lines of code, double-click on the left of the line. You can delete the breakpoints by double-clicking on them.

3. **Add a watch-expression**
   In the watch-expression panel, add a boolean expression you want to test. For example a comparison between two variables in your programs. To identify a variable in a program, prefix it with the program name and a point, e.g. "A.x" for a variable x in program A.

4. **Add a conditional breakpoint**
   In the conditional breakpoint panel, add a boolean expression. After starting the debugging process, the DIbugger will stop the execution, if the boolean expression evaluates to true.

5. **Start debugging**
   Press the "Play"-Button.

6. **Make steps**
   Have fun! Step over, step back, make one or more steps in one program only or in all programs at once.

7. **Understand the output**
   Amongst other things, the DIbugger shows you the values of all variables and the values of your watch-expressions at the current moment of execution.

8. **Stop debugging**
   Press the stop button.

9. **Make your changes and start the process all over again.**

10. **Save your work.**
    You can do this via the File-Menu.

1.2 The User Interface

This section provides a short overview over the user interface. The functions of the buttons and menu entries itself are explained within the next chapters.

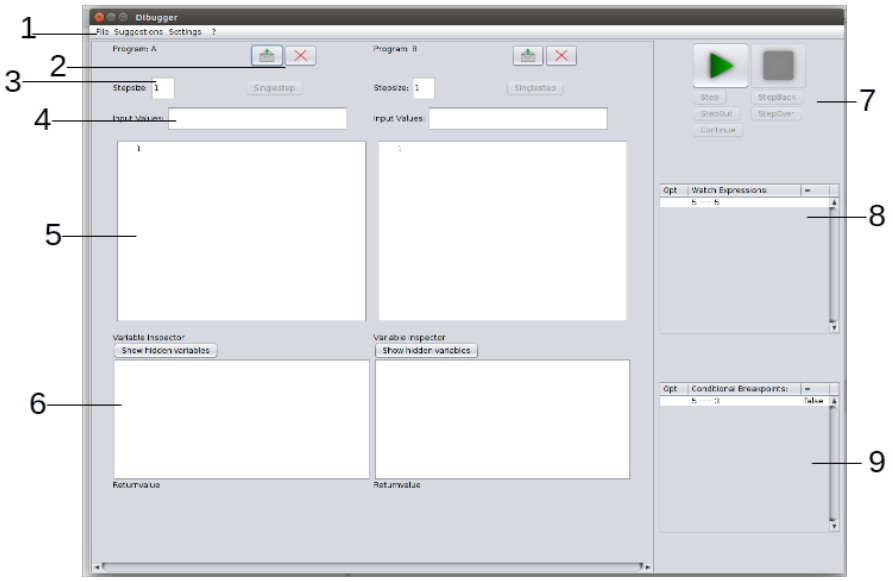

Figure 1.1: The User Interface

1 Menu

2 Load text into program window / delete programwindow

3 Stepsize of an individual program

4 text box fot the input variables

5 text box for the program code

6 values of variables in current debugger step

7 Play-Button, Stop-Button, steps

8 Watch Expression Panel

9 Conditional Breakpoint Panel

Chapter 2

Configuring the DIbugger

2.1 Configuring the Language

You can change the language of the DIbugger with:
Menu > Settings > Change language

2.2 Configuring the Maximum Number of Function Calls

The maximum number of function calls determines how many function calls are allowed within one program (including the obligatory main()-method). This number can be used to impose a limit on the depth of recursion within your program. You can change the maximum number of function calls with:
Menu > Settings > Change maximum function calls

2.3 Configuring the Maximum Number of Iterations

The maximum number of iterations determines how many iterations of a while-loop are allowed within one program, thus guaranteeing that a loop cannot run forever. You can change the maximum number of iterations with:
Menu > Settings > Change maximum iterations

Chapter 3

Loading and Saving

3.1 Adding a program

The file menu allows you to add a program to a new program window. You are asked to choose the file you want to open within the new program window. If you cancel the file choosing action, you are presented with an empty new program window. The program you want to load has to be a .txt-file. You can do this with:
Menu > File > Add a Program

3.2 Adding text to a program window

There are multiple options to add text to a program window: You could copy-paste it, drag and drop the text or use the evelope button above the program panel which opens a file-chooser dialog.

3.3 Saving DIbugger-Configurations

Saving a DIbugger-Configuration means saving not only the code in your program windows but your Watch-Expressions, Breakpoints and Conditional Breakpoints, too. You can do this with:
Menu > File > Save config

3.4 Saving Code

There is no dedicated menu entry to save your program code as your code is automatically saved when you save your DIbugger-Configuration. You can do this with:
Menu > File > Save config

3.5 Loading DIbugger-Configurations

You can load a DIBugger-Configuration (that is all your program windows, Breakpoints, Conditional Breakpoints and Watch-Expressions) with:
Menu > File > Load config

3.6 Removing a program

You can only remove one of your programs as long as there are more than two programs. In other words: There have to be two program windows at all times. (As otherwise, a relational DIbugger would not make sense.) You can remove a program window by using the "X"-Button above the upper right corner of the program you want to remove.

Chapter 4

Editing Program Code

4.1 Resetting a Program Code Window

You can reset your programcode the same way as you remove with: Menu > Settings > Change language

4.2 Required Format of the Code

The format of the Wlang code is basically in C syntax, although there are some differences as explained below.

Writing functions and procedures You can write a routine in Wlang with the syntax `<type or void> <routinename>(<list of arguments>) <content of the routine>`. The arguments of the routine have to be separated with a ",", as usual. This is the same syntax as in the programming language C. For every program it is necessary, that there is a main-Routine with the name main. This is the entry point of the execution. All other routines have to be declared and implemented above the main method. In other words: The main method has to be the last method in your program.

Declaring arrays Array declaration has the syntax `<type> <dimensions> <arrayname>`.

Calling functions You can call a function in a assignment. Within this assignment the functioncall is the only thing on the right side e.g. `x = foo(y);`

4.3 Sample Program

As an easy example, see the implementation of the factorial of an integer in two different ways. With iterations:

```
//factorial programmed in an iterative manner
int main(int n){
    int i = 1;
    int sum = 1;
    while(i<=n) {
         sum = sum * i;
         i = i + 1;
    }
    return sum;
}
```

Or with recursion:

```
//other functions must be declared before the main
int fac(int k) {
    //Calculate the factorial of k recursively.
    if (k <= 1)
         return 1;
    // int res = fac(k-1) in a single command is not applicable
    int res;
    res = fac(k-1); //this is the correct way to call functions.
    res = res * k;
    return res;
}

//every program needs a main method
int main(int k) {
    int res;
    res = fac(k);
    return res;
}
```

4.4 Common Mistakes

- You always need a main method. Without a correct main method a syntax error pop-up will occur.
- The main method has to be the last method of your program.
- You cannot write a routine inside another routine. This leads to a syntax error, too.
- Make sure that all array declarations are in the form described in this paragraph. It is not allowed to write the dimensions after the name of the array.
- Make sure that all input values (in the text field above your program) are seperated with a semikolon.

- It is not allowed to call a function and use its return value for calculations immediately, e.g. `x = bar(4)+3;`. The function call must stand alone on the right side of the assignment.
- Similarly, you cannot return the result of a calculation immediately, e.g. `return bar(4)+3;`. Instead you could write something like: `int i = bar(4)+3; return i;`

4.5 Auto-Generating Input

To get suggestions for input variables, you can use the menu as follows: Menu > Suggestions > Suggest Input Values

Chapter 5

Simple Breakpoints

Breakpoints in the DIbugger work just like breakpoints in any other debugger: They mark the lines where the execution of the program code stops when you are debugging. The first start of the debugger will automically run until it meets its first (conditional) breakpoint. If you then press the "Continue"-Button in the debug-mode, each program runs until it meets its next breakpoint (or conditional breakpoint).

5.1 Adding a Breakpoint

You can add a breakpoint by double-clicking into the space on the left of the program line where you want to break.

5.2 Removing a Breakpoint

You can remove a breakpoint by double-clicking on it.

Chapter 6

Watch-Expressions and Conditional Breakpoints

6.1 Syntax of Watch-Expressions

A Watch-Expression allows you to compute information about your program code as you execute it. For example, if you have a program A which contains a variable "x" which should behave the same way as a variable "y" in program B, you could add a new Watch-Expression as follows:

```
A.x == B.y
```

Or:

```
A.x + B.y
```

While debugging your program (that is after starting the debug mode by pressing the Play-Button), the Watch-Expression will always tell you its current value.

The following examples illustrate the types of expressions allowed:

```
A.x != B.y
true
5.0
(A.x-B.y)*3
```

Writing the name of a variable will return its value at the current point of execution.

```
A.x
```

You could also use some fixed number to compute information with:

```
A.x == 5
B.y != 114.2
A.x - 5
B.y * 13
```

The syntax of Watch-Expressions does not allow comparisons of three programs. To ensure the equivalence of a variable across three programs, you would have to use two expressions, e.g.:

```
A.x == B.x
B.x == C.x
```

6.2 Adding a Watch-Expression

You can add a new Watch-Expression in the Watch-Expression Panel on the right side of the program. To do this, click into the middle cell (the section under "Watch Expression:") of the last Watch-Expression in the Watch-Expression Panel. After the program start, this will be the already existing examplary Watch-Expression. Clicking there will add a new Watch-Expression that you can modify in the next steps.

6.3 Modifying a Watch-Expression

To change an existing Watch-Expression, double-click on its middle cell (the section under "Watch Expression:") and type in your changes.

6.4 Deleting a Watch-Expression

To delete an existing Watch-Expression, click into the empty space on the left of your Watch-Expression. This will open a menu where you can select DELETE, then press the OK-Button.

6.5 Adding and Changing the Scope of a Watch-Expression

The scope of a Watch-Expression consists of the lines of code, wherein the Expression is evaluated. Thus, if you are only interested into comparing variables in a certain section of two programs, you could use a scope to do so.
By default, the scope is set to all lines of the program. To add or change a scope, click into the empty space on the left of your Watch-Expression. This will open a menu where you can select ADJUST SCOPE, then press the OK-Button and exit via the "X"-Button in the corner of the dialog.

6.6 Syntax of Conditional Breakpoints

Each Conditional Breakpoint consists of a boolean term the syntax of which is equivalent to the syntax of Watch-Expressions. (If the Conditional Breakpoint does not consist of a boolean term, it is always evaluated to false.) Their usage differs, though: If the boolean term of the Conditional Breakpoint is evaluated

to true, the Conditional Breakpoint stops the execution of the program in the current line. A Conditional Breakpoint thus does exactly what its name suggests: It breaks on a condition. In contrast to a traditional breakpoint, it is therefore not bound to any fixed line.
But beware: **The Conditional Breakpoint only breaks if it is evaluated to true during the steps that you make. In other words: There might be a point in your program where the condition is true - although the Conditional Breakpoint did not break.**

6.7 Adding a Conditional Breakpoint

Adding Conditional Breakpoints works the same way as explained for Watch-Expressions in 6.2.

6.8 Modifying a Conditional Breakpoint

Modifying Conditional Breakpoints works the same way as explained for Watch-Expressions in 6.3.

6.9 Deleting a Conditional Breakpoint

Deleting Conditional Breakpoints works the same way as explained for Watch-Expressions in 6.4.

6.10 Adding and Changing the Scope of a Conditional Breakpoint

Adding and changing scopes of Conditional Breakpoints works the same way as explained for Watch-Expressions in 6.5.

6.11 Using Suggestions for Watch-Expressions and Conditional Breakpoints

Suggestion for Watch-Expressions can be generated with:
Menu > Suggestions > Suggest watchexpression

Accordingly, suggestions for Conditional Breakpoints can be generated with:
Menu > Suggestions > Suggest conditional breakpoint

After the suggestion window pops up, press “Ok” to generate a suggestion. Copy and paste it to your Watch-Expression or Conditional Breakpoint Panel if you intend to use it.

Chapter 7

Debugging Program Code

After two or more programs are added you can start the Debug-Mode with the green play button in the upper right corner of the DIbugger. More general: to control the debugging process, use the buttons on the right. It is always possible to stop the Debug-Mode and return to Edit-Mode by clicking the red square stop button beside the play button.

7.1 Making Steps

There are different buttons for different kinds of steps.

Step The button labeled with "step" leads to an execution of all programs either according to the number of commands given in the specified stepsize, or until the occurrence of the next breakpoint if it occurs before the specified number of steps is fulfilled. Assume for example the stepsize of program A is 3 and the stepsize of program B is 2. Then, there are three commands executed in program A and two commands in program B. Encountering a function call, the execution is always stepping into the function.

StepBack The button labeled with "stepBack" causes the DIbugger to rewind one command in each program.

StepOut The button labeled with "stepOut" causes the DIbugger to jump out of the current function in every program.

StepOver The button labeled with "stepOver" causes the DIbugger to jump over the next command, if it is a function call.

Continue The button labeled with "continue" causes the DIbugger to run each program until either a conditional breakpoint evaluates to true or a normal breakpoint occurs.

Singlestep There is one button up above each program with the label "singlestep". This button will run exactly one command only in the program it belongs to.

7.2 Understanding the Output

You can inspect the variables occuring in a program in the variable inspector, which is settled under the programs code. With your right mouse button you can hide a variable.
These values are shown only after you started the debug mode.
A program, whose main-routine has a return type will produce an output. This return value will be printed right under the variable inspector.