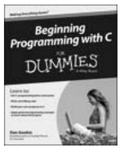
HOW TO USE THE CODE::BLOCKS DEBUGGER WITH C PROGRAMMING


RELATED BOOK
Beginning Programming with C For Dummies

By **Dan Gookin**

Code::Blocks integrates the GNU debugger, which is one of the most popular debuggers available for programming with C. As long as you create a project by including debugging information, you can use the debugger from within Code::Blocks to peer into your code and, hopefully, discern its ills and ails.

HOW TO SET UP THE DEBUGGER

To debug a project, you need to set its target — the resulting program — to have debugging information included. The debugger uses that information to help you locate flaws in your code and generally to see how things work. This process works when you create a debugging target build for your code. Follow these steps:

1. Start a new project in Code::Blocks.
   Choose File→New→Project.

2. Choose Console Application and click Go.

3. Choose C and click Next.

4. Type the project title.

5. Click the Next button.
   So far, these first few steps are the same as for creating any C language console program in Code::Blocks.

6. Place a check mark by the Create “Debug” Configuration.

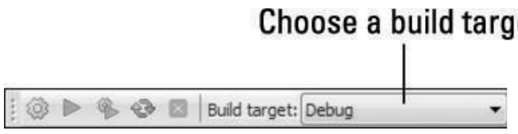
The Debug setting allows a program to be created with special debugging information included.

7 Ensure that the item Create “Release” Configuration is also selected.

8 Click the Finish button.

The new project appears in Code::Blocks.

When you activate debugging for a project, as well as keeping the release configuration (refer to Step 7), you can use the Compiler toolbar to choose which version of the code is created. Use the View→Toolbars→Compiler command to show or hide that toolbar.



When debugging, ensure that the Debug command is chosen as the build target. You cannot debug the code unless the debugging information is included in the final program.

REMEMBER

To create the final program when you’re finished debugging, choose the Release command from the Build Target menu. Though you could release a debugging version of your program, that information makes the final program larger. It also includes your source code so that anyone else can “debug” your program and see how it works.

HOW TO WORK THE DEBUGGER

The debugger operates by examining your code as it runs, showing you what’s happening, both internally to the program as well as the output. If you’ve created a new Code::Blocks program with debugging information, and you have code to debug, you’re ready to start.

This code is purposefully riddled with bugs.

DEBUG ME!

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
  char loop;
  puts("Presenting the alphabet:");
  for(loop='A';loop<='Z';loop++);
    putchar(loop);
  return 0;
}
```

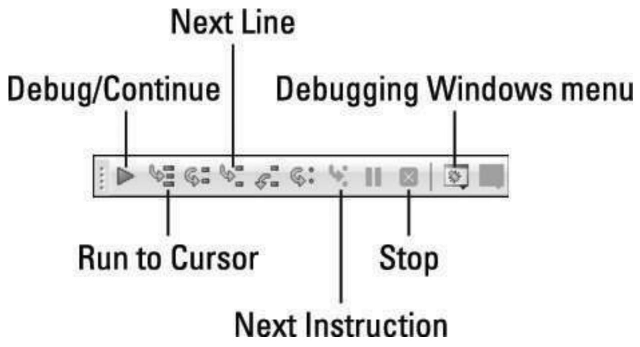
Exercise 1: Create a new project in Code::Blocks, one that has a Debug target build. Copy the source code from Debug Me! into the main.c file. Ensure that you copy the text exactly, including a mistake you may see at the end of Line 9. Build and run.

Because the Code::Blocks editor is smart, as are other programming editors, you may catch the erroneous semicolon at the end of Line 9 because the following line didn’t automatically indent. That’s a big clue, but it’s also something you may not notice, especially if you have 200 lines of code to look at. Regardless, the program’s output tells you something amiss. Here’s what you should see:

```
Presenting the alphabet:
[
```

The alphabet doesn’t show up, of course. Not only that, what’s the [character for? Time to debug!

Use the Debugger toolbar in Code::Blocks to help you wade into your code to see what’s fouled up. To show or hide that toolbar, choose View→Toolbars→Debugger.



Follow these steps to work through your code to see what's wrong:

1 Click the cursor in your code right before the puts() statement.

That would be at Line 8.

2 Click the Run to Cursor button on the Debugging toolbar.

The program runs, but only up to the cursor's location. The output window appears, and debugging information shows up in the logging panel at the bottom of the Code::Blocks window.

3 Click the Next Line button.

The puts() statement executes; its output appears.

4 Click the Next Line button again.

The for loop does its thing; no output.

5 Click the Next Line button again.

The putchar() function displays a random character on the screen.

Hopefully, at this point you look closer at your code and find the stray semicolon at the end of Line 9. You don't need to exit or stop the debugger to fix it.

6 Remove the semicolon at the end of Line 9.

7 Click the Stop button to halt the debugger.

Now you try to see whether you've fixed the problem, by stepping through the code again:

8 Click the mouse pointer to place the cursor right before the for statement at Line 9.

9 Save and rebuild your code.

10 Click the Run to Cursor button.

11 Click the Next Line button twice.

An *A* appears as output. Good.

12 Keep clicking the Next Line button to work through the for loop.

Or, if you're satisfied that the code has been debugged:

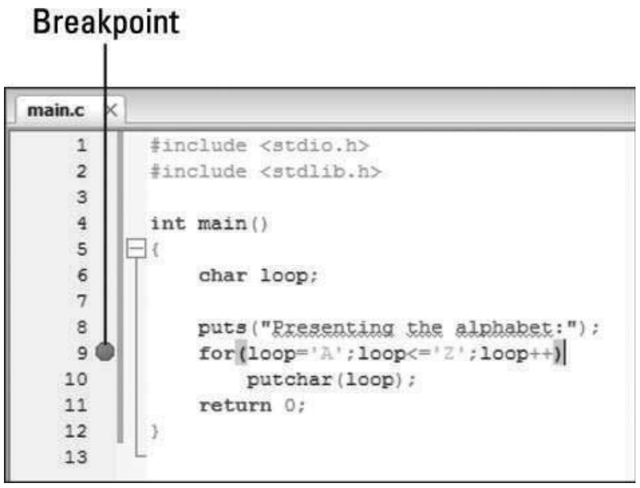
13 Click the Stop button.

The program runs fine after you fix the stray semicolon. The random output character was due to the putchar() function at Line 10 being executed without the loop variable initialized. The character you see is whatever random garbage exists at the variable's location in memory.

HOW TO SET A BREAKPOINT TO DEBUG C PROGRAMMING

No one wants to step through 200 lines of source code to find a bug. Odds are that you have a good idea where the bug is, either by the program's output or because it ran just five minutes ago, before you edited one particular section. If so, you know where you want to peek into operations. It's at that place in your code that you set a debugging breakpoint.

A *breakpoint* is like a stop sign in your text. In fact, that's the exact icon used by Code::Blocks. To set a breakpoint, click the mouse between the line number and the green line (or yellow line, if you haven't saved yet). The Breakpoint icon appears.



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

int main()
{
    char loop;

    puts("Presenting the alphabet:");
    for(loop='A';loop<='Z';loop++)
        putchar(loop);
    return 0;
}
```

To run your code to the breakpoint, click the Debug/Continue button on the Debugging toolbar. The program works as you've intended, but then comes to a screeching halt at the breakpoint. From then on, you can step through the code or click the Debug/Continue button again to run the program until the next breakpoint — or to the current breakpoint when it's in a loop.

C FOR DUMMIES, 2ND EDITION

MORE ABOUT THIS BOOK

Author: Dan Gookin