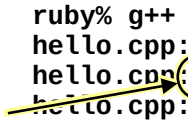
Engineering 101

HOW TO Debug Code

Debugging Syntax Errors (compile-time errors):

A *syntax error* is a mistake in the use of a programming language, or *grammar*, detected during compiling.

1. Find the line number of the very first compiler error message. In the example below the very first error is on line 7.



```
ruby% g++ hello.cpp -o runhello -s
hello.cpp: In function `int main()':
hello.cpp:7: error: `end1' undeclared (first use this function)
hello.cpp:7: error: (Each undeclared identifier is reported only once
   for each function it appears in.)
hello.cpp:9: error: no match for 'operator>>' in 'std::cout >>
   " World!"'
hello.cpp:10: error: `endl' undeclared (first use this function)
hello.cpp:10: error: parse error before numeric constant
```

2. Find the error on or before that line in your code and fix just that one error with help from the sometimes cryptic error message.
3. Save and recompile before fixing other errors because later errors may be results of earlier errors.

Debugging Semantic Errors (run-time errors):

A *semantic error* is a mistake in the meaning, or *logic*, of the program, detected during program execution, after compiling.

Printing Debugging Messages Using "**`cerr`**":

You can temporarily print messages to the screen by inserting "**`cerr`**" statements throughout your code. The "**`cerr`**" command is like "**`cout`**" except that "**`cerr`**" prints immediately whereas "**`cout`**" sometimes prints delayed.

*Pinpointing an Error Using "**`cerr`**":*

When your program crashes or quits prematurely with error messages like "**`Segmentation Fault`**" (usually caused by out-of-range array indices) or "**`Floating Exception`**" (usually caused by division by zero), you can pinpoint the error location from the last printed "**`cerr`**" message inserted throughout the code like these:

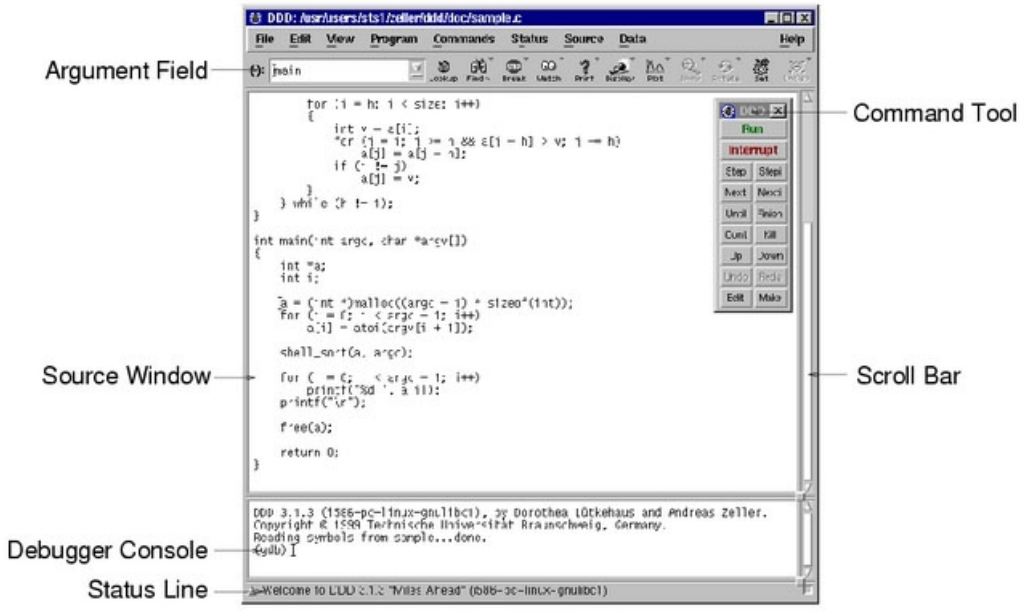
```
cerr << "point 1" << endl;          // or
cerr << "Got here 4" << endl;       // or
cerr << "Inside addNum() function" << endl;
```

*Displaying Values Using "**cerr**":*

When you program finishes properly but produces erroneous output, you can print out values of variables to keep track at which point the values become wrong in your code. Some example print statements are:

```
cerr << "Num1 is " << num1 << endl;                    // or
cerr << "mass[" << i << "] = " << mass[i] << endl;     // or
cerr << "Inside addNum() function: x = " x << " and y = " << y << endl;
```

Using the DDD (Data Display Debugger) Graphical Debugging Program:



Initial DDD Window

http://www.gnu.org/software/ddd/

DDD is a graphical debugger which allows you to step through your program line by line while displaying the values of variables of interest. To use DDD, follow these steps:

1. Open your code with in a text editor like xemacs:

   ruby% xemacs *filename*.**cpp &**

2. Compile your code with the "**-g**" debugging flag and without the "**-s**" stripping flag:

   ruby% g++ *filename*.**cpp -o** *executablename* **–g**

3. Run DDD with the executable as its argument with the ampersand (**&**):

   ruby% ddd *executablename* **&**

   - Your code will appear in the main source window.

4. Set breakpoints where you wish to begin stepping line by line through your code.

   a. Click on the blank space on the left of a line where you want to place a breakpoint.
   b. Click on the "**Break**" button in the toolbar up top.

OR

   a. Right click on the blank space on the left of a line and select "**Set Breakpoint**."

5. Run your program to the first breakpoint by clicking "**Run**" on the floating command tool window.

   - Input from "**`cin`**" statements is entered in the bottom debugger console window.
   - Output from "**`cout`**" statements is displayed in the bottom debugger console window.

6. Display values of variables of interest.

   a. Click on a variable to highlight it.
   b. Click the "**Display**" button in the toolbar.

OR

   a. Right click on a variable and then select "**Display** *variablename*".

   - The display window will open above the main source window.
   - You may organize the boxed values by dragging them around the display window.
   - Placing the mouse pointer over a variable will display the value above the variable.

7. Step through your code by clicking on the "**Next**" button (to execute one line of code) or by clicking on the "**Step**" button (to step into a function), both in the floating command tool window.

   - The green arrow on the left of a line shows the location of the next line to execute.
   - Observe that the values displayed in the display window may change.
   - Continue running the program to the next breakpoint by clicking "**Cont**."
   - Stop the program at anytime by clicking "**Kill**" and rerun by clicking "**Run**."

8. Once you find your mistake, fix your code in a separate text editor.

9. Recompile your code once again with the "**`-g`**" debugging flag.

10. Rerun your program in DDD by clicking "**Run**" in the floating command tool window.

    - DDD will state in the debugger console window that the program has changed.

Using Additional DDD Resources:

The DDD online manual can be found at:
`http://www.gnu.org/manual/ddd/`

A sample DDD session can be found at:
`http://www.gnu.org/manual/ddd/html_mono/ddd.html#Sample%20Session`