Developing and debugging a C Program under Linux

Teodor Rus

August 21, 2011

1 Steps of a C program development

There are four steps in the process of a C program development:

1. Write your program on a ".c"a terminated file. This is how C compiler recognizes that in input file may contain a C program. Example of file containing a C program would be `myFirstC.c`.

2. Remove all syntax-errors from your program. For that you need to compile your program using the command `gcc myFirstC.c`. All syntax-errors discovered bt the C compiler in the program written in the file `myFirstC.c` are reported as message on the screen of your terminal. The form of these messages is:
   `file.c:line# error:` text of the syntax error in line#
   You need to read these errors and change the line-numbers of the file `file.c` thus reported such that your program conforms to the syntax of C language. There is no tool that may help you with this process. You need to understand C language syntax to carry out this activity.

3. Remove all the warnings C compiler sends you concerning your program. Warnings are similar to syntax errors and the compiler sends them to you following the same mechanism as syntax errors. But warnings are not syntax errors. Warnings are compiler's limitation in treating some legal conditions. For example, if the `main()` function in your C program contains no `return()` (which is not required) the C compiler may tell you that control reaches end of that function. Warnings are reported by the compiler similar with the syntax errors and intermixed with them using messages of the form: `file.c:line# warning:` text of the warning in line#
   A program that generates warnings may well execute correctly but the C compiler cannot guarantee that this will happen. Therefore, in order to ensure that your program is syntactically correct you need to eliminate all warnings the C compiler finds in your program. But the C compiler does not automatically generates all warnings that may exists in your program. Hence, in order to force the C compiler to report all warnings it discovers in your program and then to eliminate them you must call the C compiler with the option `-Wall` on. For example, to force

the C compiler to report all warnings it discovers in your first C program written in the file `myFirstC.c` you must call the compiler with the command `gcc -Wall myFirstC.c`. Again, there is no tool that may help you with the process of removing the warnings from your C program. To remove the warnings reported by the C compiler you need to understand the meaning of the C language construct the compiler complains about and to fix its syntax accordingly.

4. Make sure that the computation you want to implement is actually implemented by your program. For that you need to execute the code generated by the compiler on battery-of-testes that cover all cases of control-flow in your program. This activity is called *debugging* your program. Every programming language is supposed to provide a tool called `debugger` that allows you to run your program stepwise, statement-by-statement if necessary, while examining the values of the variables involved in the statements you are executing. To use this tool you need to have a program on which compiler reports no syntax errors or warnings. The `debugger` relies on the information collected by the compiler during code-generation in order to do its job. The compiler collects this information if the user asks it to do so. The user asks the compiler to collect debugging information using an appropriate option on the compilation command. The C language program development under Linux system use *Gnu compilers*. To tell the Gnu C compiler to collect debugging information you must use the `-g` option on the compiling command. That is, you must call the Gnu compiler by the command `gcc -g file.c`. Then you must execute your program by starting first the debugger using the command `gbd` and then by running the debugger on the executable generated by the compiler, as we shall see further.

In conclusion, to develop a C program follows the steps:

1. Compile the program with the command `gcc -Wall file.c`. If syntactic errors or warnings are reported, you must fix them. You need to repeat this step until no syntax errors or warnings are reported by the compiler.

2. Compile the program with the command `gcc -g file.c` (note the use of the option "-g" which tells the compiler to collect debugging information.

3. Run the program under the debugger as explained below.

2   Using the GDB debugger

The purpose of a debugger, such as GDB, is to allow you to see what is going on "inside" your program while it executes or what was your program doing at the moment when it crashed.

GDB can do four main kinds of things (plus other secondary things in support of these four) to help you catch semantic errors (also called bugs) in in your program:

1. Start your program, specifying anything that might affect its behavior.

2. Make your program stop on specified conditions.

3. Examine what has happened, when your program has stopped.

4. Change things in your program, so you can experiment with correcting the effects of one bug and go on to learn about another.

You can use GDB to debug programs written in C, C++, and Modula-2. Fortran support will be added when a GNU Fortran compiler is ready.

Program compilation: to use GDB you need to compile your program by the command

gcc -g program.c, which creates the executable a.out, or
gcc -g -o program program.c, which create the executable program.

This allows the compiler to collect debugging information

3   GDB commands

GDB is invoked with the shell command `gdb`. Once started, it reads commands from the terminal until you tell it to exit with the GDB command `quit`. You can get online help from gdb itself by using the command `help`.

4   Running the program under GDB

You can start GDB with no arguments or options by tying the command `gdb`; but the most usual way to start GDB is with one argument or two, specifying an executable program as the argument.

For example:
gdb program /* Example: gdb ./a.out */
tels the GDB to run your executable file `program`. You can also start with both an executable `program` and a `core` file using the command: `gdb program core`.

In addition, you can also specify a process ID as a second argument, if you want to debug a running process. For example, `gdb program 1234` would attach GDB to process 1234 (unless you also have a file named 1234; GDB does check for a core file first).

5   Frequently used commands

Here are some of the most frequently needed GDB commands:

- `break [file:]line` sets a break-point at the statement recorded on the line `line` in the file `file`, if specified.

  Example: break assign1.c:28 or break 28 would set a break-point at the statement recorded on the line 28 of the file assign1.c.

- `break [file:]func` sets a break-point at the function `func` in the file `file` if specified.

  Example: break assign1.c:func or `break func` sets a break-point at the entry in the function `func` in the program `assign1.c`.

- `run [arg-list]` starts your program (with arg-list, if specified).

  Example: `run text1 text2` start your program with the arguments `text1` and `text2`.

- `bt` backtracks your program when it stops, that is, display the program stack when the program stops.

- `print expr` evaluate the expression `expr` and print the result.

  Example: `print i` would print the value of the variable `i`.

- `continue` or c tells the debugger to continue the running of your program (after stopping, e.g. at a break-point).

- `next` tells the debugger to execute next program line (after stopping). Note, it steps over any function call in the line.

- `edit [file:]function` tells the debugger to look at the program line where it is presently stopped.

- `list [file:]function` tells the debugger to type the text of the program in the vicinity of where it is presently stopped.

- `step` tells the debugger to execute next program line (after stopping). Note, it steps into any function call in the line.

- `help [name]` tells the debugger to show information about GDB command `name`, or general information about using GDB.

- `quit` tells the debugger to exit. That is, one exits from GDB by `quit`.