

Laboratory Exercises, Part 0

Deadline 2018-09-09

by Jens Eliasson, 2018-09-02

In this lab, you will start to learn software development in C using a modern development environment, Eclipse, as well as with the command line in Linux.

The goal of the lab is to show how a simple C project is created, compiled and executed in either Code::Blocks or Eclipse IDE, or using the command line on Linux. But, before you start, read chapters 1-3 in the 3rd edition of the book [2] or Introduction and chapters 1-2 in the 4th edition of the book [3]. Ask the teachers and lab assistants if you need help. It is also a very good idea to print out this assignment and bring it to the lab (with very I mean mandatory).

In [1], you will find the manuals of all available functions in the C standard library, search for example for *printf()*, *puts()*, *gets()*, *getchar()* and study how to use C functions. These functions are important building blocks when creating programs. Learn how to interpret the documentation over available functions in the C standard library. The *printf()* function can for example be found using this direct link: <http://linux.die.net/man/3/printf>

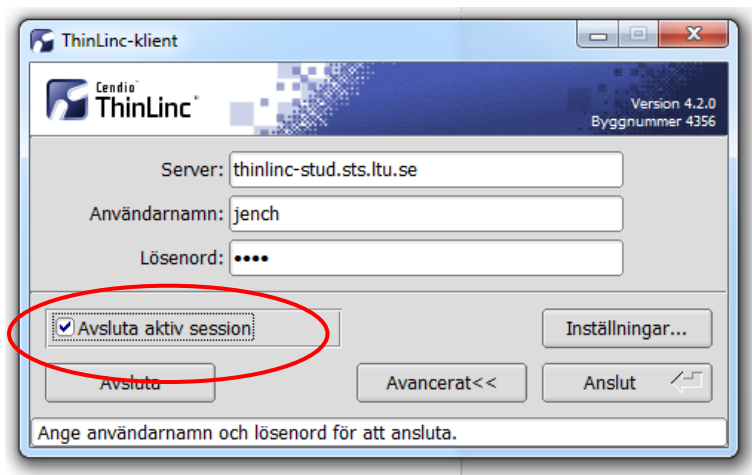
You can replace *printf()* in the link above with the function of your choice.

Exercise 0.1 – Getting started

When in the computer lab:

- Click “Linux Ubuntu (Student)” on the welcome screen
- Login with your LTU username (the default server is thinlinc-stud.sts.ltu.se)

When using ThinLinc on Windows:



- Start ThinLinc Client
- Use **thinlinc-stud.sts.ltu.se** as server. It is also important to check “Avsluta aktiv session”, otherwise you will leave all applications running

when you log out. This will consume resources on the server, making it run very slow for yourself as well as other students.

Command line

The C compiler *gcc* can be invoked using the command line in Linux. To do so, use the *Applications* menu and click “Terminal emulator”. Then type “cd” and press enter to navigate to your home folder.

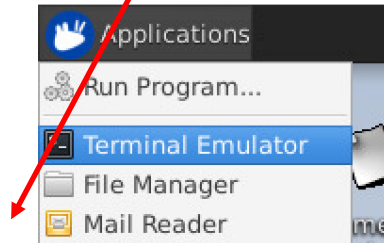


Figure 1: Start the terminal application

Common terminal commands:

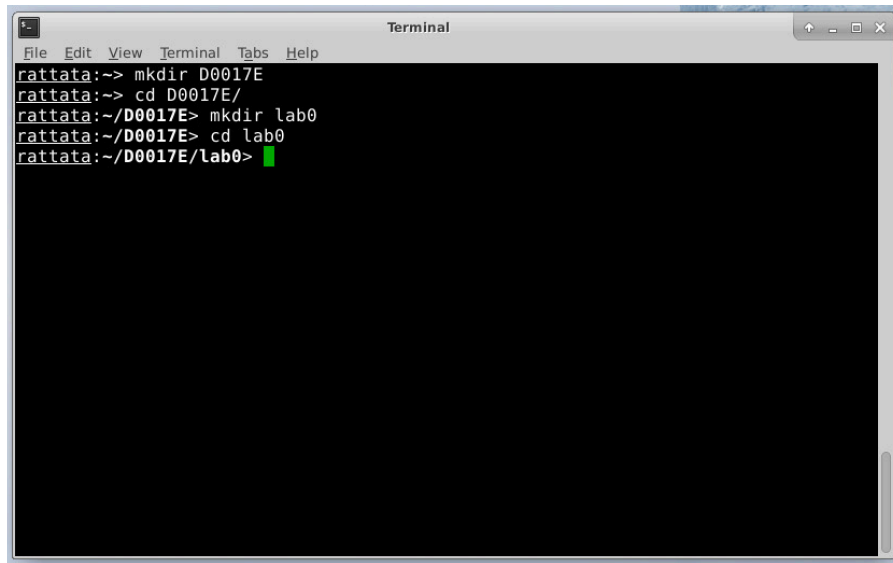
Command name	Action
cd	Changes working directory \$ cd lab0 \$ cd ..
pwd	Prints current working directory \$ pwd
gcc	Invokes the GNU gcc compiler \$ gcc -o appname file1.c file2.c
appname	\$./appname
gedit	Starts a text editor (use & to run in background) \$ gedit lab0.c &
rm	Deletes a file \$ rm brokenfile.c
ls	Lists files and folders \$ ls
make	Tools for automatic building (compilation) of projects \$ make

In lab0.zip there is a Makefile and an example C file. Build in the Terminal using the make command:

```
$ cd Src  
$ make
```

Introduction to programming for engineers – D0017E

Below is an example of usage of the command line.

A screenshot of a terminal window titled "Terminal". The window has a menu bar with "File", "Edit", "View", "Terminal", "Tabs", and "Help". The terminal shows the following commands and their outputs:

```
rattata:~> mkdir D0017E
rattata:~> cd D0017E/
rattata:~/D0017E> mkdir lab0
rattata:~/D0017E> cd lab0
rattata:~/D0017E/Lab0> █
```

Figure 2: Commands' examples

1. Using Code::Blocks Integrated Development Environment (IDE)

You can start Code::Blocks from the Applications -> Development menu. Start by clicking on the "Code::Blocks IDE" icon.

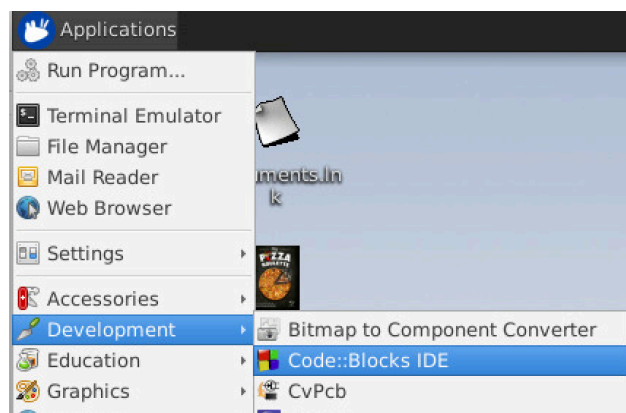


Figure 3: Starting Code::Blocks IDE

1.1 Exercise - Creating a Code::Blocks C project.

Now, it is time to start programming. We will first learn how to create a new C project, compile, and run it. Click "*File -> New -> Project*". Select "*Console application*" and click "*Go*". Click Next, and select C as language on the next screen. Input the name, for example lab0, and location.

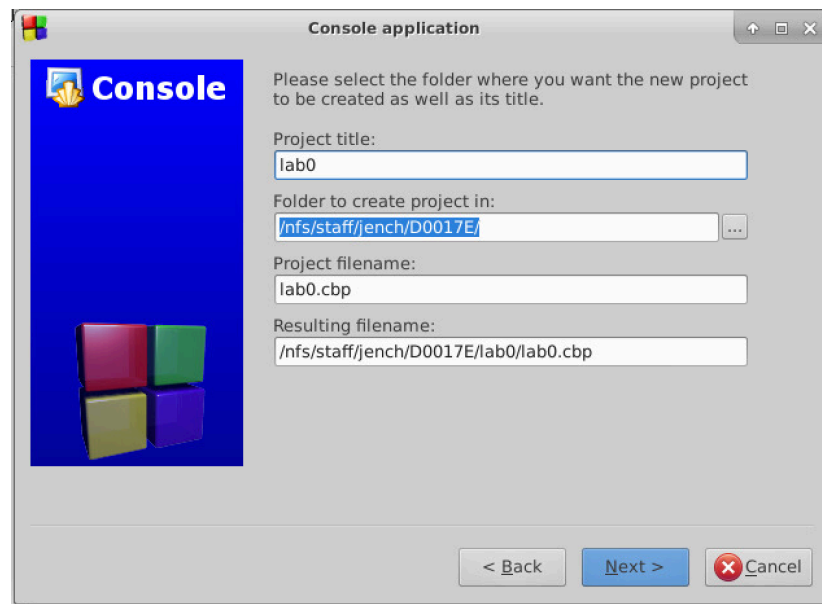


Figure 4: New C project

Fig 5. Shows the project settings to use.

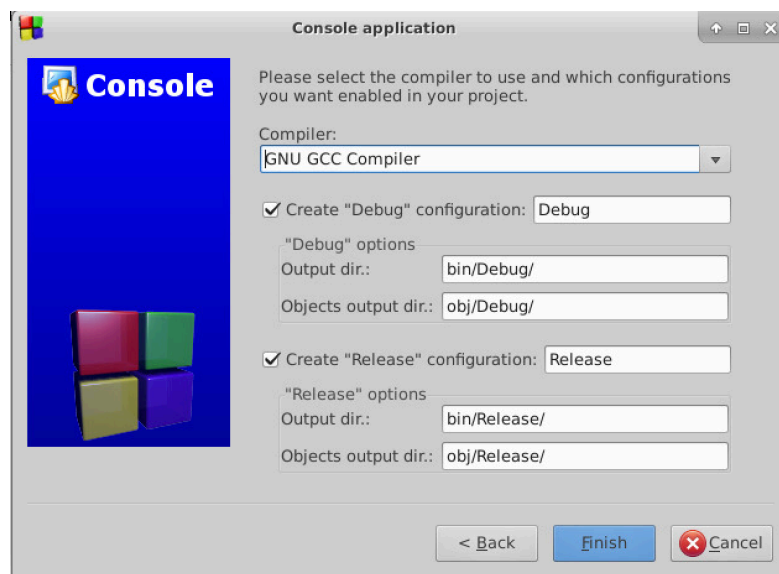


Figure 5: Project settings

Click Finish to create the project. Now you are ready to start. Now go to Figure 10 for a first sample application.

2. Using Eclipse Integrated Development Environment (IDE)

You can start Eclipse from the *Applications -> Development* menu. Start by clicking on the "Eclipse" icon.

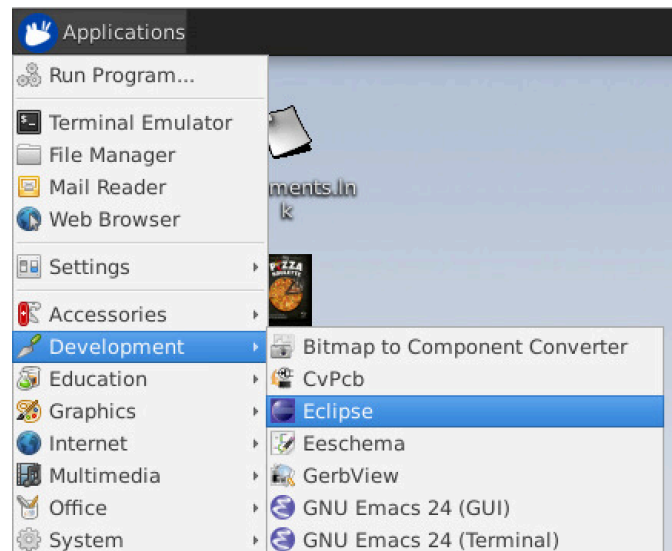


Figure 6: Starting Eclipse IDE

2.1 Exercise - Creating an Eclipse C project.

Now, it is time to start programming. We will first learn how to create a new C project, compile, and run it.

Start by creating a new C project, see Figure 7.

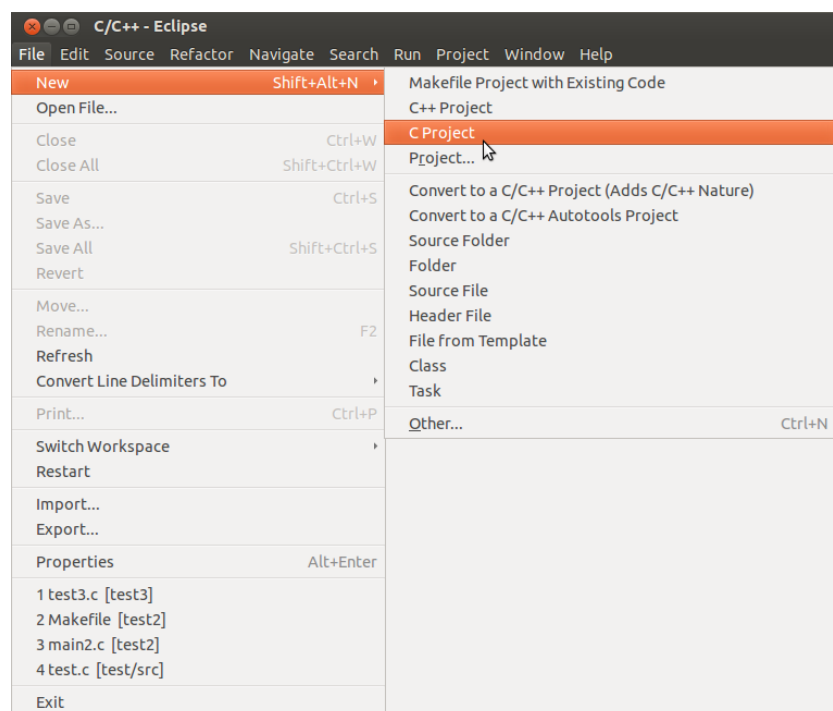


Figure 7: New C project in Eclipse

Chose a suitable name (e.g. lab0, lab1 ...), and the Linux GCC tool chain, see Figure 8. Do **not** use spaces or other odd characters in file or folder names. Click “Finish”.

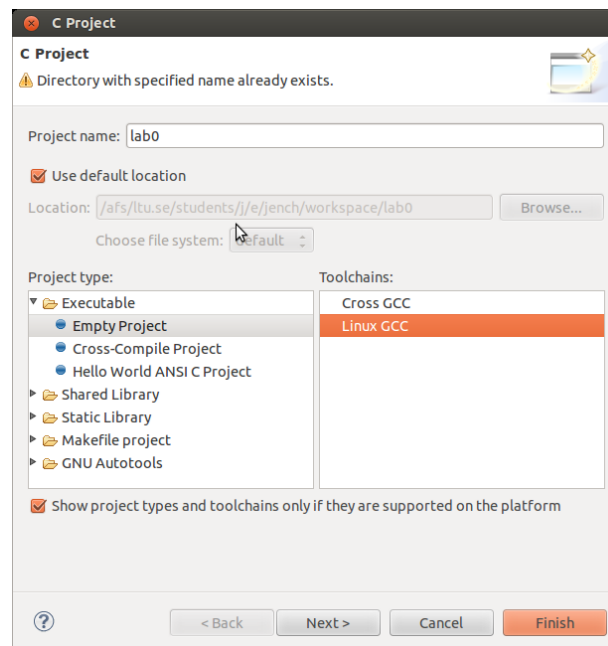


Figure 8: Project name and tool chain selection

Next step is to add new source file(s) to the empty project. See Figure 9 for this. Name the file lab0.c

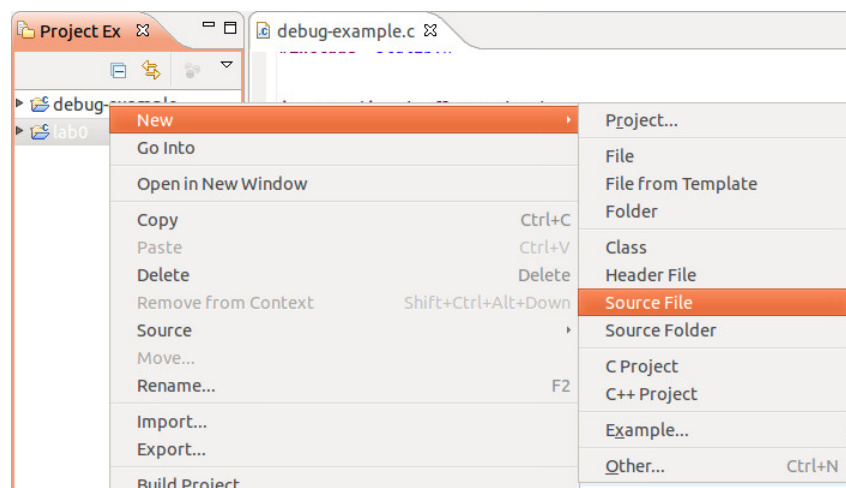


Figure 9: Adding source files

Now input the code shown in Figure 10. This very classical first C program will simply write the text “Hello, world!” on the screen and then exit. All C programs must have one function called `main()`. This is where the program starts. `main()` can either be void, or return an integer. Click on the hammer icon to build (compile) the program. To execute it, click on Run icon (the green circle with a white triangle). The text “Hello, world!” should now print in the “Console” window.

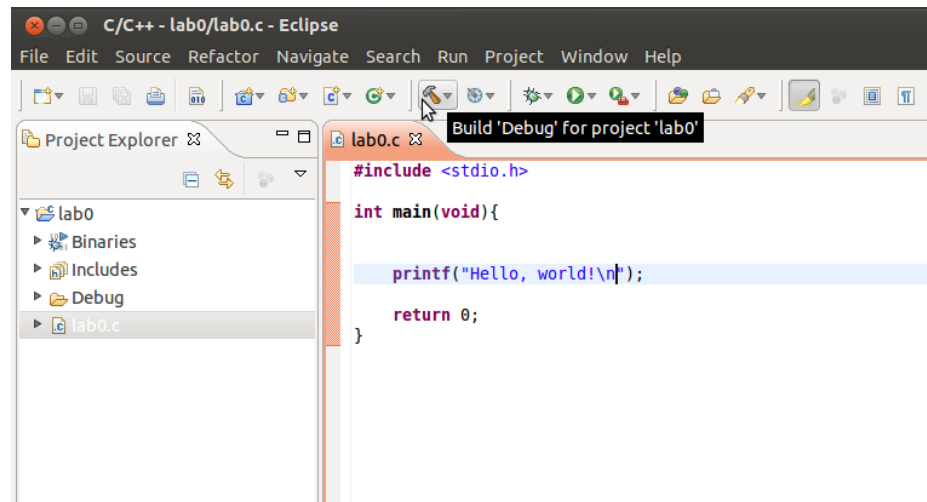


Figure 10: First program

Now, try and remove the semicolon on the last line and rebuild the project. What happens? See Figure 11 for an example. The gcc compiler detects that a semicolon is missing and aborts the build process with an error message. The error message contains information about filename and line number where the error occurred. Restore the semicolon and build and execute the project again.

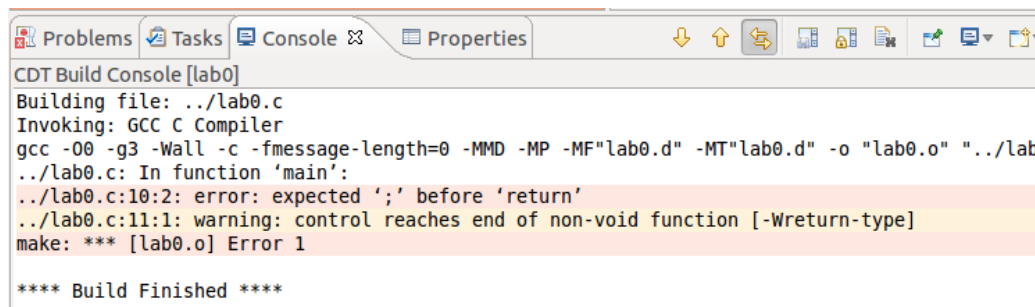


Figure 11: Compilation error

When this works, use the exercise from the book's chapter to write a few simple C programs. Also print integers, strings and float numbers, as shown in the examples below.

Code examples

Modify the hello world program and test the different code examples below. See how the output from the program changes. Try to understand how the output is generated by the code. Try to change the values of the variables and test the program again. Details on how to use *printf()*, variables, etc. will be covered in upcoming lectures and labs.

```
// this is a one-line comment

/* this a multi-line
   comment */
```

Printing an integer:

```
int i = 23;
printf("i=%d\n", i); // the \n combo will result in a new
line on the screen
```

Printing a signed/unsigned integer:

```
unsigned int i = 23;
int j = -23;
printf("i=%u\n", i);
printf("j=%d\n", j);
```

Printing a float:

```
float f = 3.14;
printf("f=%f\n", f);
```

Printing an index in a float array:

```
float farr[2] = {1.0, 2.0};
printf("farr[0]=%f\n", farr[0]);
```

Printing a string:

```
char *str = "This is a string";
printf("The string is: %s\n", str);
```

Printing a character:

```
char c = 'A'; // note the difference between a
string "..." and a character '.'
printf("The character is: %c\n", c);
```

main() examples

```
void main(void) // no arguments, no return value
int main (void) // no arguments, returns int to
                // operating system
int main(int argc, char *argv[]) /* array of strings
                                as argument
                                with length parameter, returns
                                int to operating system */
```

To send arguments to the program, go to Project->Properties->Run/Debug settings. Select lab0 -> Edit -> Arguments. Test to send in an argument, and print it with

```
printf("arg[0]: %s\n", argv[0]); // what happens if
                                // arg[1] is printed?
printf("Number of arguments are: %d\n", argc);
```


Try the following code, which is a basic example of iteration. What happens??

```
int i;
for(i=0; i<5; i=i+1) {
    printf("i=%d\n", i);
}
```

All functions that normally come in the C standard library, such as *printf()*, have a manual on how to use them. See [1] for more functions. Try, for example, to print strings using the *puts()* function.

Mandatory exercises

The following lists of exercises from the book [2] or [3] must be completed and presented to a lab assistant in order to pass this lab. Please note that Chapter 3 in the 3rd edition corresponds to Chapter 2 in the fourth edition. **All exercises will be graded individually but can be solved in groups of two.** Each chapter of the course book, and thus lecture, has its own exercises. The exercise notion of x.y means Chapter x, Exercise y. For example: 3.1 indicates Exercise 1 in Chapter 3. The chapter numbers are based on the 4th edition.

- Oral presentation of (tested) programs – 2.2, 2.4, 3.3, 3.6, 3.8
- Oral presentation of answers (on computer or paper) – 2.3, 2.5, 3.2
- Oral presentation of (tested) programs – 4.2, 4.4, 4.5, 4.8

It is possible to group many exercises in the same .c file. Use the following code example to avoid creating a new Eclipse project for each assignment. It is a good practice to create one project per book chapter and lab.



Figure 12: Grey out unwanted code

Code that is within a `#if 0 ... #endif` block is greyed out and not part of the compiled code. Source code inside a `#if 1 ... #endif` block is part of the compilation. Use this to grey out all completed exercises in the same file.

Lab submission

There is no Canvas submission for this lab, but you are required to show your lab assistant your results. Do not continue with lab1 or other exercises from the book until you have completed all assignments in this lab. Note that not all assignments are verified, but you are still expected to complete all assignments found in the course manual.

References

1. C functions manual, <http://linux.die.net/man/3/>
2. Programming in C, third edition, Stephen G. Kochan
3. Programming in C, fourth edition, Stephen G. Kochan
4. Code::Blocks, www.codeblocks.org/
5. Eclipse IDE, www.eclipse.org/ide/