C TUTORIAL

Short Description. This tutorial explains the basics of editing, compiling, executing, debugging, and finally testing and evaluating C programs.

Prerequisites. Rudimentary knowledge of Unix line commands, see for example the TU01 "Unix Tutorial" available online on Moodle.

1. Editing

Edit the C program of Fig.1 with your favorite text editor and save it under the name “hello.c”.

```
#include <stdio.h>
main()
{
   printf("hello, world\n");
}
```

Fig. 1. Hello World

Hint: Make sure you have copied the program character by character. Otherwise you will have to debug it.

2. Compilation

A compiler is a program which generates from a source file an executable program, called "binary file". A well known compiler for the C language is gcc, a GNU open source compiler.

1. Open a terminal shell and change your working directory to the one where your source file "hello.c" is located :

   ```
   % cd <relative or full directory path name>
   ```

2. Compile your source file "hello.c" :

   ```
   % gcc –o hello hello.c
   ```

   Note that "gcc" is the call to the compiler, "-o hello" an option indicating to gcc the name of the binary file to be produced, and "hello.c" the source file to be compiled.

3. Execution

Execute your binary file "hello" (note that you have to give the relative path "`./`") :

```
% ./hello
hello, world
%
```

4. Debugging

A compiler can only compile a correct source file. First, the compiler checks the source code, and if there are no syntax errors, it translates it into machine instructions. Otherwise, the compiler displays an error message. The process of finding such errors is called "debugging". Unfortunately, the error messages provided by C compilers are very cryptic for beginners (but soon, even non professionnals find these comments very useful).

Exercise: Edit hello.c and remove the semicolon '`;`' at the end of the fourth line. Save it under the name `hello2.c`. Of course, this piece of code contains now an error. If you compile this file you will obtain something like :

```
% gcc –o hello2 hello2.c
hello2.c: In function 'main':
hello2.c:5: syntax error before '}' token
%
```

The compiler tells you that an error has been detected on line 5, before the closing "}" bracket. Correct the error, recompile the code and execute it again.

4. Test and Evaluation: Interactive and Batch Mode

It is very important to test and evaluate a correctly compiled program. The importance of this task is often underestimated, perhaps because it is often at least as difficult and complex as designing and implementing the application itself ! Fortunately, the Unix line command provides a very powerful tool to realize quite exhaustive and flexible testings, at least for simple programs. And for more complex programs, other powerful tools are available.

```
#include <stdio.h>

/* copy input to output; 2nd version */
main()
{
    int c;

    while ((c = getchar()) != EOF)
        putchar(c);
}
```

Fig. 2. Copy file

Copy the program of Fig. 2, borrowed from [KR 88], page 17, in your favorite editor and save it under the name "copy_file.c". Compile it :

```
% gcc -o copy_file copy_file.c
```

and try out the following commands:

```
% ./copy_file
% ./copy_file < copy_file.txt
% ./copy_file < copy_file.txt > result.txt
```

where "copy_file.txt" is a character file containing whatever text you like, e.g. the text of Fig. 3.

The first of these commands allows you to test the proram **interactively**, the second one in a **hybrid interactive-batch mode,** and the third one in a fully **batch mode**.

```
This is a small test file,
allowing to run the copy_file program in batch mode
```

Fig. 3. Example of an input file for the program of Fig. 2

Play around with other commands inspired by your own fantasy, even crazy ones like :

```
% ./copy_file < copy_file.c | ./copy_file
% ./copy_file < copy_file
```

Hint: Unlikely patterns allow you to test the robustness of a program.

Important Notice : In most Unix shells, the End-of-file character (EOF) is produced by hitting <return>, followed by <Ctrl-D>. If this don't work, try out <return>, followed by <Ctrl-Option(alt)-D>.

Reference

[KR 88] B. W. Kernighan, D. M. Ritchie, *The C Programming Language*, 2nd Ed., Prentice Hall, 1988.

Acknowledgement. Many students and assistants have successfully contributed to this tutorial in the past: Oliver Hitz, Amine Tafat, Alexander Kaufmann, Fulvio Frapolli and Michael Luggen.