Operating Systems Lab - Week 1: exercise - with answers

This lab is about getting started with the environment for this course and practising the basic concepts of C. You will write the code of a program that prints a fixed number of times the welcome string `hello world!`.

1 Getting started

1.1 Connect to the teaching server `linux.cim.rhul.ac.uk`

For pedagogical reasons, we ask you to write, compile, and run your programs on the Linux environment provided by the department, i.e. the RHUL Computer Science *teaching server*. You can connect to the teaching server with the available command line `ssh` client, e.g. `puTTY`, and use a command-line editor, e.g. emacs, vim, or nano. Otherwise, you can use the `NoMachine` graphical interface. In this case, you can open the terminal and the text editor on separate windows.

Depending on your OS, use the following instructions to connect to `linux.cim.rhul.ac.uk`:

Unix Open the terminal and run

```
ssh yyyyxxx@linux.cim.rhul.ac.uk
```

where `yyyyxxx` is your college username, and enter your password to access the teaching server.

Windows Launch the Windows SSH client `puTTY` [1], enter `linux.cim.rhul.ac.uk` in the empty field *Host Name (or IP address)* and click on *Open*. The client opens a new window where you are required to enter your college user name `yyyyxxx` and password.

1.2 Create a new directory

To see the content and navigate in your home directory use the UNIX commands: `ls`, `cd`, `..` Create a new directory, called `CS2850Labs`, by running

```
mkdir CS2850Labs
```

We suggest you use `CS2850Labs` to save and run all programs of this course. You can create a sub-directory of `CS2850Labs` called `weekI`, $\texttt{I} = 1, \ldots, 11$, with

```
mkdir weekI
```

Use `ls` to show the content of the current directory and `pwd` to print its path.

1.3 Create and edit a text file with a command line editor

Choose one of the following command-line editors: emacs, nano, or vim. From the terminal, you can create an empty file, `helloWorld.c`, and open it with the text at once by running

```
vim helloWorld.c
```

To enter characters, go to *insert mode* by typing `i`. Use `ESC` to go back to *command mode* and `:wq` (in command mode) to save an exit. Write something in the file and check that everything was saved correctly using

```
more helloWorld.c
```

[1] `puTTY` should be installed on all department's machines. If you work on your own Windows machine you can download it at download puTTY and install it as explained.

2 Your first C program

2.1 Write the C code

Open `helloWorld.c` again, replace your name with the following C code

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

save, and exit.

2.2 Compile and run your C code

Compile `helloWorld.c` by running

```
gcc -Wall -Werror -Wpedantic helloWorld.c
```

To see all compilation options, type `man gcc` in the terminal and scroll the page with the up and down arrows. To read more about the meaning of the flags `-Wall`, `-Werror`, and `-Wpedantic` have a look at the gcc online manual on your web browser.
If you now print on the screen the content of `week1`, you should find a new file, `a.out`, which is the executable of `helloWorld.c`. Try to open it with vim. What do you observe? Why does the content of the file look so strange? In command mode, type

```
:%!xxd -b
```

to see the binary in the right format.
To see what `helloWorld.c` does, execute the binary file, `a.out`, by running

```
./a.out
```

Check that your output is exactly as follows

```
hello, world
```

2.3 More *hello, world*

Add the following lines to your code (just below the first call to `printf`)

```
printf("hello");
printf(",");
printf(" ");
printf("world\n");
printf("hello, world\n hello, world\n  hello, world!\n");
```

and check that the output is

```
hello, world
hello, world
hello, world
 hello, world
  hello, world!
```

including the strange indentation and the exclamation mark. What happens if you swap the exclamation mark and the last *newline* symbol, `\n`?

Answer: You get

```
cim-ts-node-01$ ./a.out
hello, world
hello, world
hello, world
 hello,  world
  hello, world
!cim-ts-node-01$
```

2.4 Debugging

The free system `valgrind` contains powerful debugging tools for Linux programs. The Valgrind suite is already installed on `linux.cim.rhul.ac.uk` and we suggest you use it to detect possible bugs in the programs you write for these labs. To see what may be wrong with your program, run the following

```
valgrind ./a.out
```

and have a look at the messages printed on the terminal. For the moment, this may look unnecessary and the messages you get are quite trivial. But running such sanity checks will become more and more important in the following weeks. One of the hardest parts of learning C is to understand how to manage the memory allocated by a program and looking at the `valgrind` messages may save you hours of debugging work.

3 Control flow

The control structures `for` and `while` allows you to repeat an operation a given number of times. Their usage and syntax in C are similar to what you know from other programming languages, but we suggest you have a look at this C online manual for all the details.

3.1 Create loops with `for`

Copy the code given in Section 2.1 into a new file called `forHelloWorld.c`. Add the following *macro-substitution* instruction on Line 1

```
#define N 10
```

and write a `for`-loop to make the program print the string `hello, world` `N` times. See Section 4.11.2 of The C Programming Langaugefor more details about macro-substitution statements. A `for`-loop in C is specified by three quantities

- the *iterator*, which needs to be declared as an integer `int i` and initialised inside the `for`-loop arguments list
- the *stopping condition*, e.g. `i < 4`, which stops the iteration when *false*
- the *iteration step*, e.g. `i = i + 1`, which defines the increment of the iterator at each iteration

For example, a `for`-loop defined by

```
int i;
for (i = 3; i <= 6; i = i + 2) {
   doSomething(...);
}
```

will call the function `doSomething` 2 times.

Answer:

```
#include <stdio.h>
#define N 10
int main() {
  int i;
```

```
  for (i=0 i < N; i = i + 1)
    printf("hello, world\n");
}
```

3.2 `while`-loop (optional)

Have a look at the following program

```
#include <stdio.h>
#define N 10
int main() {
  int i;
  int sum = 0;
  for (i = 0; i < N; i = i + 2) {
    sum = sum + i;
    printf("%d + ", i);
  }
  sum = sum + i;
  printf("%d = %d\n", i, sum);
}
```

Can you predict what is the output on the terminal without compiling and running the program? The following C code uses a `while`-loop and an `if`-statement to produce an analogous output

```
#include <stdio.h>
#define N 10
int main() {
  int i = 0;
  int sum = 0;
  while (i < N) {
    if (i % 2 == 0) {
      sum = sum + i;
      printf("%d + ", i);
    }
    i++;
  }
  sum = sum + i;
  printf("%d = %d\n", i, sum);
}
```

Write a new version of both programs, so that the output of both becomes

```
1 + 3 + 5 + 7 + 9 = 25
```

Use `Valgrind` to see if your program runs correctly and the heap usage of your program.

Answer: With a `for`-loop

```
#include <stdio.h>
#define N 10
int main() {
  int i;
  int sum = 0;
  for (i = 1; i < N -1; i = i + 2) {
    sum = sum + i;
    printf("%d + ", i);
  }
  sum = sum + i;
  printf("%d = %d\n", i, sum);
}
```

and with a `while`-loop:

```c
#include <stdio.h>
#define N 10
int main() {
  int i = 0;
  int sum = 0;
  while (i < N — 1) {
    if (i%2==1) {
      sum = sum + i;
      printf("%d + ", i);
    }
    i++;
  }
  sum = sum + i;
  printf("%d = %d\n", i, sum);
}
```