C1

Introduction to C Programming: Toolchain

This exercise introduces you to C programming. It has three aims. Firstly, to introduce you to the resources available to read up on features of the C language. Secondly, to introduce you to the use of a command line shell to interact with the computer through a terminal. And thirdly, to familiarize you with the tool chain consisting of a text editor and a C-compiler.

```c
/* hello.c */
/* A famous test for the C build process. */

#include <stdio.h>

#define SUCCESS 0

int main() {
    printf("Hello world!\n");
    return SUCCESS;
}
```

Schedule

Preparation Time	:	1 hour
Lab Time	:	3 hours

Items required

Tools	:	
Components	:	
Equipment	:	
Software	:	`gcc`, `notepad++` or equivalent text editor

Version: October 5, 2020

Klaus-Peter Zauner
Electronics and Computer Science
University of Southampton

Before starting the laboratory you should read through this document and complete the preparatory tasks detailed in section 2.

> **Academic Integrity** – *If you wish you may undertake the preparation jointly with other students. If you do so it is important that you acknowledge this fact in your logbook. Similarly, you will probably want to use sources from the internet to help answer some of the questions. Again, record any sources in your logbook.*

You will undertake the exercise working individually. During the exercise you should use your logbook to record your observations, which you can refer to in the future – perhaps to write a formal report on the exercise, or to remind you about the procedures. As such it should be legible, and observations should be clearly referenced to the appropriate part of the exercise. As a guide the ✎ symbol has been used to indicate a mandatory entry in your logbook. However, you should always record additional observations whenever something unexpected occurs, or when you discover something of interest.

For each Task you should create a new directory so that you have a working version of every program at the end of the lab. Remember to place comments in your code.

You will be marked using the standard laboratory marking scheme; at the beginning of the exercise one of the laboratory demonstrators will provide you with answers to the preparatory work and at the end of the exercise you will be marked on your progress, understanding. After the lab the demonstrator will mark your preparation and logbook.

Notation

This document uses the following conventions:

✎	An entry should be made in your logbook
	Care should be exercised
`command input`	Command to be entered at the command line

1 Introduction

This lab introduces you to programming in C.

1.1 Outcomes

At the end of the exercise you should be able to:

- Write simple C programs.
- Compile and run C programs from the command line.
- Interpret error messages from the compiler.

2 Preparation

Take a look at the slides from lecture 2. For this exercise you will need to know a bit about the 'Hello World" program (see Hello World) and about compilation (see Compilation).

Consider the following three variations of the "Hello World" code from the lecture 2 slides:

LISTING 1: The closing */ in line 1 is missing:

```
1 /* hello.c
2 /* A famous test for the C build process. */
3
4 #include <stdio.h>
5
6 int main() {
7     printf("Hello world!");
8     return 0;
9 }
```

LISTING 2: The name of the library in line 4 has been changed:

```
1 /* hello.c */
2 /* A famous test for the C build process. */
3
4 #include <mystdio.h>
5
6 int main() {
7     printf("Hello world!");
8     return 0;
9 }
```

LISTING 3: The name of the function called in line 7 has been changed:

```
1 /* hello.c */
2 /* A famous test for the C build process. */
3
4 #include <stdio.h>
5
6 int main() {
7     myprintf("Hello world!");
8     return 0;
9 }
```

Copy the table below into your logbook and mark in it what you expect to happen when these variants are compiled. You are not expected to get these right, but you should be able to justify why you answered in the way you did.

	Listing 1	Listing 2	Listing 3
No error message			
An error from the preprocessor			
An error from the compiler			
An error from the linker			

Read up on how to use variables in C, either in the textbook pp. 19–20 or the C Wikibook section on variables[1]. Then write down a declaration for a variable. You should know what its *type* is, what its *name* is, and what its *value* is.

3 Laboratory Work

3.1 Part 1

To set up your work environment, take the following steps:

1. Open the Text Editor
2. Open a terminal with the compiler

   (a) Create a “Folder” named “elec1201” and within it a subfolder “c1”. Do not use any spaces in your folder names or your file names!

   (b) Resize the terminal window so that it takes up about half of your screen. The editor can share the other half with the file browser.

Copy `hello.c` from https://secure.ecs.soton.ac.uk/notes/ellabs/1/c1/hello.c to your folder and open it with the text editor.

[1] https://en.wikibooks.org/wiki/C_Programming/Variables

LISTING 4: hello.c

```c
/* hello.c */
/* A famous test for the C build process. */

#include <stdio.h>

#define SUCCESS 0

int main() {
    /* A new-line control character (\n)    */
    /* is used here, in case the terminal   */
    /* collects complete lines before       */
    /* showing them.                        */
    printf("Hello world!\n");
    return SUCCESS;
}
```

In the compiler terminal you can type `ls` (which stands for "list") to show the files in your folder. It should now show the `hello.c` file.

Note, that the terminal remembers previous commands and you can use the up-arrow cursor key to bring up a previously entered command line. You can also use the Tab key to let the command line attempt to complete what your typing. (Try this out, it is very convenient.)

To compile the program, enter the following command in the terminal:

```
gcc hello.c -o hello
```

This will run the gcc compiler[2], tell it to compile the file `hello.c` and to write the output of the compilation to a file named `hello`.

On the command line, the first word is always the command to be called, subsequent words are command line arguments that will be supplied as input to the command, or command line options that change how the command will act. The latter start with a dash (-). So here `hello.c` is an argument, and `-o` ("output") is an option, that is followed by an argument for the option, the `hello`.

Why was there a warning above (2c) not to place spaces in file names or folder names?

Execute the above command. Then use the `ls` command to see what you find in your folder.

Be very careful that you never specify the filename of your source code as the name where the compiler output should be written to—the compiler will otherwise overwrite it without warnings or questions.

While you develop a program it is a good idea to make frequent backup copies. You can use the `cp` ("copy") command to make bakups like this:

[2]The gcc stands for GNU Compiler Collection, because it provides front-ends for several languages and back-ends for many processor architectures.

```
cp hello.c hello_bak001.c
```

The compiler has produced an executable file from the source code (on Windows the ending .exe is added to indicate that it is executable).

You can now run this program, by typing its file name in the terminal[3]:

```
hello
```

It should print its message on the terminal.

Note, how your hello.c program now acts like a new command line command named `hello`. In fact many of the command line commands are nothing else than C programs with the corresponding name—one can easily add new commands to the terminal shell by writing the required C programs.

Now modify the `hello.c` program equivalent to the three listings in the preparation section above. Compile after each change and observe the messages form the compiler. Note in your logbook what errors are raised and write some comments on how these error messages correspond to the changes you made.

How do you recognize which part of the build phase (preprocessor, compiler, linker) raised the error?

Do your observations agree with your expectations from the preparation?

Modify the `hello.c` program by removing the closing comment symbols `*/` from the second line rather than the first line and compile it. How does this change the response of the compiler? Can you explain why the compiler responds so differently?

3.2 Part 2

Copy `helloyou.c` from https://secure.ecs.soton.ac.uk/notes/ellabs/1/c1/helloyou.c to your folder and open it with the text editor.

LISTING 5: helloyou.c

```
1  /* helloyou.c */
2  /* Hello with text string. */
3
4  #include <stdio.h>
5
6  #define SUCCESS 0
7
8  char name[]="human";
9
10 int main() {
11     printf("Hello %s!\n", name);
```

[3] On Mac OSX or Linux you would need to type `./hello` to indicate that you want to call a program from the local directory.

```
12      return SUCCESS;
13  }
```

In the program you find a variable declaration with initialization; in this case a text string that is an array of characters. Observe how the value of the variable is included in the output of the program.

Use the text editor and add your own variable declaration from the preparation to the code. Then change the `printf()` statement, such that it prints the value of your variable. Depending on the type of variable you have written down in the preparation you will need a suitable format specifier in the first argument to the `printf()` function. In the example given, the format specifier `%s` specifies that a string argument will follow.

Type	Format specifier	Output as
int	%d	decimal value
int	%x	hexadecimal value
float or double	%f	fixpoint notation
float or double	%e	exponential notation
char	%c	

After you have successfully printed out the value of your variable, investigate what happens if the variable type and the format specifier do not match. Keep notes in your logbook of the results of your experiments.

3.3 Part 3

In this section you will learn how to use input from the user in a program.

Copy `hello-input.c` from https://secure.ecs.soton.ac.uk/notes/ellabs/1/c1/hello-input.c to your folder and open it with the text editor.

LISTING 6: hello-input.c

```
1   /* hello-input.c */
2   /* Ask for input. */
3
4   #include <stdio.h>
5
6   #define SUCCESS 0
7   #define NAME_BUFFSIZE 100
8
9
10  int main() {
11      char name[NAME_BUFFSIZE];
12
13      printf("What is your name?  ");
14      scanf("%s", name);
15      printf("Hello %s!\n", name);
16      return SUCCESS;
```

```
17 }
```

Compile the program and test it.

In line 11 memory space is reserved for the user input. What do you think will happen if the input is larger than the reserved space?

Change line 7 to make the buffer smaller and test what happens with a long input.

The possibility that user input can overflow the allocated memory and thus write in an uncontrolled way to adjacent memory is exploited in many security attacks (“buffer-overflow attack”). Code like the one in this example should never be used where the user may have malicious aims.

4 Optional Additional Work

In the example code we have seen constants defined with preprocessor commands (e.g. SUCCESS). The compiler also defines constants for the programmer, one of which is `__LINE__`, it contains the current line number.

Define (with the preprocessor) a statement called MARK that inserts a (printf()) call that prints the current line number. Test it and write it in your logbook.

Now see whether you can make this MARK silent, except if the label DEBUG is defined. You will find the required information to achieve this in https://gcc.gnu.org/onlinedocs/gcc-3.0.1/cpp_4.html.

Write the code you have developed in your logbook—it will be useful for debugging your programs.