Introduction to C via Examples

Last update: Cristinel Ababei, January 2012

1. Objective

The objective of this material is to introduce you to several programs in C and discuss how to compile, link, and execute on Windows or Linux. The first example is the simplest hello world example. The second example is designed to expose you to as many C concepts as possible within the simplest program. It assumes no prior knowledge of C. However, you must allocate significant time to read suggested materials, but especially work on as many examples as possible.

2. Working on Windows: Example 1

To get started with learning C, I recommend using a simple and general/generic IDE with a freeC/C++ compiler. For example, Dev-C++ is a nice compiler system for windows. Download and install it on your own computer from here:
http://www.bloodshed.net/devcpp.html

In my case, I installed it in **M:\Dev-Cpp**, where I created a new directory **M:\Dev-Cpp\cristinel** to store all projects I will work on. Inside this new directory, I created yet one more **M:\Dev-Cpp\cristinel\project1**, where I'll store the files for the first program, which is simply the famous "hello world" program. We'll create a new project directory for each new program; this way we keep thing nicely organized in individual directories.

We create a program by creating a new project in Dev-C++. So, start Dev-C++ and create a new project. In the dialog window, select "Console Application" as the type of project and name it **hello**. Also, select "C Project" and check check-mark it as default language. Then only click Ok. In the next window that pops-up, navigate to **M:\Dev-Cpp\cristinel\project1**, as the location where we want this project to be stored.

This will automatically create a simple **main.c** template file. Replace its content with the following listing:

```
#include <stdio.h>
int main ()
{
  printf( "Hello World!\n");
  return 0;
}
```

Then save it as **hello.c**. It could be very well saved as **main.c** (the default name) but we just want to change its name. This first program will simply print "Hello World!" at the command prompt.

To compile this program, select **Execute->Compile**
Which creates the executable **hello.exe** in **M:\Dev-Cpp\cristinel\project1**

To execute our program, start a command prompt window **Start->Run** and type **cmd**. Then navigate to the project directory **M:\Dev-Cpp\cristinel\project1**, where simply type **hello.exe**. This should execute our program, which simply prints at the prompt "**Hello World!**"

A few notes about this first program:

- The #include command is a preprocessor directive to load the stdio library.
- Execution starts in a function (method) called main:
- There are other signatures for main as we will see.
- Although a return type is declared, nothing needs to be returned.
- The printf method is used for screen output.
- The "newline" character \n is explicitly required.

3. Working on Linux/Unix: Example 1

If you have a Linux machine, then you can simply use the GNU compiler. You do not need to install anything else. Use your favorite text editor to create and save the **hello.c** file in a new directory, say **project1**.

To compile it:

```
$ gcc -Wall hello.c -o hello
```

This produces an executable called **hello** which can be executed as:

```
$ hello
```

or, if you don't have the current directory in your path:

```
$ /hello
```

Which should print at the prompt "**Hello World!**"

One can also work in Linux-like environments on Windows. For example, **cygwin** is a collection of tools which provide a Linux look and feel environment for Windows. If you want, you can install cygwin and use it for your C programming projects. There is lots of online information that describes how cygwin can be used. Google and search for it. You can download cygwin here: http://www.cygwin.com/

4. Working on Windows: Example 2

A program can be split up into multiple files. This makes it easier to edit and understand, especially in the case of large programs. In addition, it also allows the individual parts to be compiled independently.

In our second program, we split up the program into three files, which we create and store in a new project directory in **M:\Dev-Cpp\cristinel\project2**:
main.c
my_utils.c
my_utils.h
For the purposes of this example (which will become clear when you will read the **main.c** source file), also create a simple text file (using any text editor you would like, such as NotePad) called **age.txt** with only one integer in one line. Store it in the same project directory **project2**.

The listing of **main.c** is:

```
#include <stdio.h>
#include <stdlib.h>
```

```c
#include <string.h>
#include "my_utils.h"

#define NUM_TIMES_TO_PRINT 3

int main(int argc, char *argv[])
{
    // (1) declare variables;
    USER_RECORD myself;
    char temp_last_name[50];
    FILE *fp; // file pointer; used to read from pre-saved age.txt;

    // (2) ask user to type in her first name; read from standard input
    // what the user types and store up to 100 characters in member "first_name"
    // of object "myself";
    printf( "Type your first name please and press Enter: \n");
    // read 100 bytes from standard input;
    fgets( myself.first_name, sizeof(myself.first_name), stdin);
    // remove the newline character from the end of the string;
    remove_newline( myself.first_name);

    // (3) ask user to type in her last name; read from standard input
    // what the user types and store what the user types in member "last_name"
    // of object "myself";
    printf( "Type your last name please and press Enter: \n");
    fgets( temp_last_name, sizeof(temp_last_name), stdin);
    // remove the newline character from the end of the string;
    remove_newline( temp_last_name);
    // allocate memory to store the last name; needed because last_name
    // variable inside the USER_RECORD structure is declared as a pointer only;
    myself.last_name = malloc( (strlen(temp_last_name) + 1) * sizeof(char));
    strcpy( myself.last_name, temp_last_name);

    // (4) ask user the gender;
    printf( "What's your gender? Type F or M. \n");
    myself.gender = getc( stdin);

    // (5) in this example's directory we have created a text file
    // that contains only one line with an integer number; we mean for
    // this number to be the age of the user; we do this only to
    // also illustrate reading from a file;
    fp = fopen("age.txt", "r"); // you should error-check this; not done here;
    fscanf( fp, "%d", &myself.age);
    fclose(fp);

    // (5) entertain use;
    printf("--- \n");
    long i;
    for ( i=0; i < NUM_TIMES_TO_PRINT; i++) {
        print_hello_message( &myself);
    }

    // (6) clean up and return;
    free( myself.last_name);
    return 0;
}
```

The listing of **my_utils.c** is:

```
#include <stdio.h>
#include <string.h>
#include "my_utils.h"

void print_hello_message( USER_RECORD *this_user)
{
    if ( this_user->gender == 'F') {
        printf( "Hello MRS. %s %s, you are %d years old! \n",
            this_user->first_name, this_user->last_name, this_user->age);
    } else if (this_user->gender == 'M') {
        printf( "Hello MR. %s %s, you are %d years old! \n",
            this_user->first_name, this_user->last_name, this_user->age);
    } else {
        printf( "Hello %s. No soup for you! \n", this_user->first_name);
    }
}

char *remove_newline( char *s)
{
    int len = strlen( s);
    // if there's a newline truncate the string
    if (len > 0 && s[len-1] == '\n') {
        s[len-1] = '\0';
    }
    return s;
}
```

The listing of **my_utils.h** is:

```
typedef struct {
    char first_name[100];
    char* last_name;
    char gender;
    int age; // in years;
} USER_RECORD;

void print_hello_message( USER_RECORD *this_user);
char *remove_newline( char *s);
```

The file **my_utils.c** contains the source of two functions, which we utilize by calling them from within the **main.c** file. Our program also includes the header file **my_utils.h**, which contains the declaration of the two functions defined in **my_utils.c**. These declarations are used to ensure that the types of the arguments and return value match up correctly between the function calls and the function definitions.

The difference between the two forms of the include statement #include "FILE.h" and #include <FILE.h> is that the former searches for FILE.h in the current directory before looking in the system header file directories. The include statement #include <FILE.h> searches the system header files, but does not look in the current directory by default.

Now, start Dev-C++ and create a new project called **newhello** in **project2** directory. Add the above three files to the project and compile. Do not forget to also create the **age.txt** file and store it in **project2** before executing the program. Start a command prompt window, then navigate to **project2** and execute **newhello.exe**. What do you get?

5. Working on Linux/Unix: Example 2

Create a new directory, say **project2**, where you should copy the three files:
main.c
my_utils.c
my_utils.h
Do not forget to also create the **age.txt** file and store it in **project2**.

To compile these source files with gcc, we use the following command:

```
$ gcc -Wall main.c my_utils.c -o newhello
```

This produces an executable called **newhello** which can be executed as:

```
$ newhello
```

or, if you don't have the current directory in your path:

```
$ ./newhello
```

6. Assignment

(Part 1)
Using Google as well as the C Programming pointers provided bellow, read the source code of the second example and explain what each line of code does.

C Programming pointers:

[1] C Tutorial, http://www.cprogramming.com/tutorial/c-tutorial.html
[2] EE 472 Course Note Pack, Univ. of Washington, 2009,
http://abstract.cs.washington.edu/~shwetak/classes/ee472/notes/472_note_pack.pdf
[3] Teach Yourself C in 21 Days, http://kldp.org/files/c+in+21+days.pdf
[4] Jacob Navia (author of lcc-win), C Tutorial, http://www.cs.virginia.edu/~lcc-win32/C-Tutorial.pdf
[5] Brian W. Kernighan and Dennis M. Ritchie. The C Programming Language. Computer Press. 1989.
(very popular book, some refer to it as the bible?)
http://net.pku.edu.cn/~course/cs101/2008/resource/The_C_Programming_Language.pdf

(Part 2)
Write a new C program that reads two matrices from two different text files (**matrixA.txt**, **matrixB.txt**), multiplies the two matrices, and write the result into a third text file (**matrixC.txt**).

The listing of file **matrixA.txt** is:

```
1 2 3 4
8 7 6 5
```

The listing of file **matrixB.txt** is:

```
1 1
1 1
1 1
1 2
```

(**Part 3**)
Search online for simple examples C programs and verify them. Learn via examples found online or discussed in the above C Programming pointers.

7. Others

Dev-C++ is only one example of "C compiler + IDE (integrated development environment)" that one can use on Windows. There are many others out there. You can search for others, read online about them (advantages, disadvantages) and select to work with whichever you feel more comfortable. Some examples include:
--Eclipse http://www.eclipse.org/cdt/
--NetBeans http://netbeans.org/features/cpp/index.html
-- Visual C++ Express http://www.microsoft.com/visualstudio/eng/products/visual-studio-express-products
--DigitalMars http://www.digitalmars.com/
--DJGPP http://www.delorie.com/djgpp/
--lcc-win http://www.cs.virginia.edu/~lcc-win32/
--See many more on the list at http://www.bloodshed.net/compilers/index.html

The compiler of choice for most users of the Linux operating system is the GNU C++ Compiler (GCC), although many others are available too. Linux is primarily a console-based environment. So GCC is a command-line compiler rather than a point-and-click GUI.

Most Mac users recommend the G++ compiler which is part of the free Xcode package. There is also a version of Eclipse for the Mac.