Analysis of the "hello.c" program

For ease of reference, we first reproduce the "hello.c" program here.

```
/* Simple C Program to print Hello Mars! */
#include <stdio.h>
int main(void)
{
    printf("Hello Mars!\n");
    return 0;
}
```

Listing 2: C program "hello.c"

A preprocessor is built into the C compiler. When the command `$ gcc hello.c↩` is given, the file "hello.c" is first preprocessed and then compiled.

- ☐ `/* Simple C Program to print Hello Mars! */`

  Anything written between the characters `/*` and `*/` is a comment. The preprocessor, in the first stage of compilation process, removes comments from the source code before it is passed to the compiler. Each comment is replaced by a single whitespace. We will discuss comments in detail later.

- ☐ `#include <stdio.h>`

  Lines that start with `\#` communicate with preprocessor. The line `#include <stdio.h>` causes the preprocessor to include a copy of the header file "stdio.h" at this place of the code. This is same as remove the line `#include <stdio.h>` and paste the content of "stdio.h" at this place. This header file "stdio.h" is provided by the C system and it is stored at some folder depending on the system. The angle brackets around "stdio.h" indicate that the header file is to be searched in the usual folder which is system dependent. This header file "stdio.h" is needed because we have used printf() function whose information is stored here. Linux commands are available to locate "stdio.h" in the system. For example, executing the command

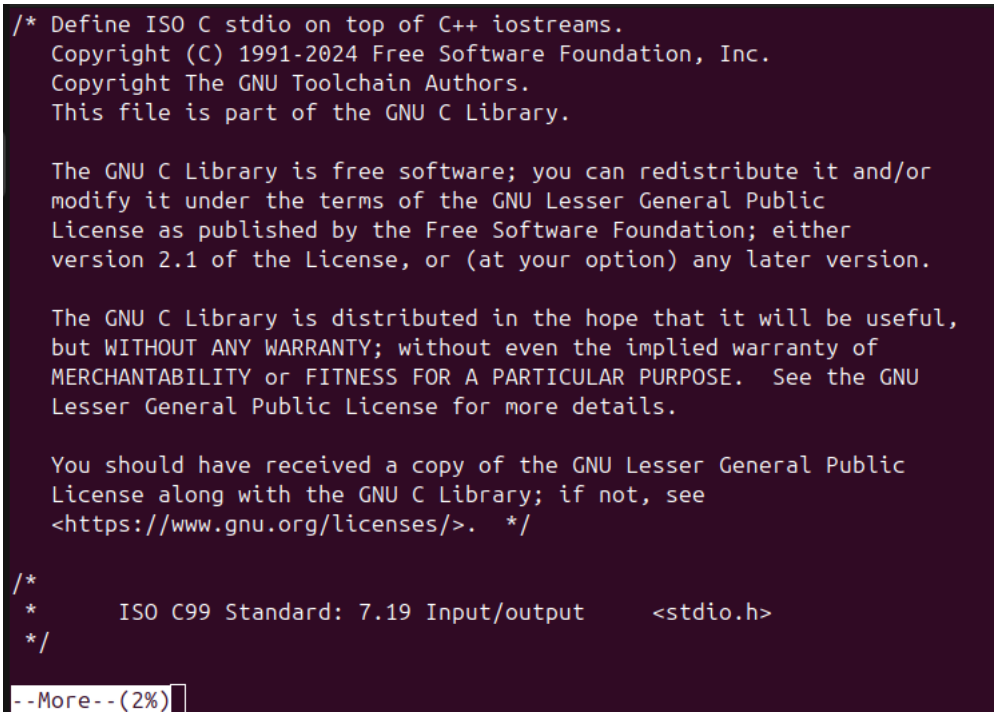
  `$ whereis stdio.h↩`

  in a Linux terminal produces the following output:

  ```
  stdio.h: /usr/include/stdio.h
  ```

  This implies that "stdio.h" is in the folder `/usr/include`. To see that content of "stdio.h", you may use `$ cat /usr/include/stdio.h↩` or `$ more /usr/include/stdio.h↩`

  A snippet of "stdio.h" using "more" command is shown below:



If you scroll down further in "stdio.h", you will see many functions such as printf(), getchar(), scanf(), etc. The name "stdio" stands for "standard input output" and thus the functions used for input and output are described in "stdio.h".

☐ `int main(void)`

Every C program must have a function named main() and program execution starts with this function. The statement "`int main(void)`" means that the main() function accepts no argument and returns an `int` value. Here, `int` means integer. The parenthesis following `main` implies that `main` is a function and `void` implies no argument. We can also write `int main()` instead of `int main(void)`. When we mention any C function, we use parenthesis [e.g., main(), printf()], which does not say anything about the argument(s). However, when we declare a function such as "`int main()`", then it implies that "main" has no argument and it returns an integer.

☐ `{`

Curly braces surround the body of a function definition. Here, the left curly brace is used for main() function. Curly braces are also used to group statements together.

☐ `printf("Hello Mars!\n");`

Here, "printf()" is a function that prints on the screen. Its details are mentioned in the header file "stdio.h". The precompiled "printf()" function is stored in some library folder of the operating system. A string constant in C is a series of characters enclosed by double quotes. Here `"Hello Mars!\n"` is the string constant which is an argument to "printf()" function. It controls what get printed. The two characters `\` and `n` at the end together (i.e., `\n` read as backslash n) constitute a single character called newline

character. It advances the cursor on the screen to the beginning of the next line. The line `printf("Hello Mars!\n");` constitutes a statement. Many statements in C end with a semicolon.

☐ `return 0;`

This is a return statement that returns zero to the operating system. Note that "main()" is called by the system and it returns an integer. Here, the integer value 0 is returned by "main()".

☐ `}`

The right curly brace matches with left curly brace [just after the "main()" definition] and it indicates the ending of the "main()" function.

Remark 1: The C and C++ standards explicitly mandate that "main()" must have a return type of `int`. Compilers like GCC follow these standards, and attempting to use `void main()` may result in a compilation error or a warning, depending on the compiler and its configuration.
Remark 2: In C and C++, the return statement "`return 0;`" in main() reports an integer exit code to the operating system. By convention, writing 0 explicitly signals successful termination and any nonzero value indicates an error or abnormal termination. If we omit the statement "`return 0;`" at the end of the main() and the execution reaches the end of main(), then it returns 0 by default. Note that you can stop the execution of main() at any location by placing a "`return 0;`" statement there.
More about comments: As mentioned earlier, C compiler ignores the comments. Comments are used to make the code easier to read and understand. There are two types of comments, namely single-line comment and multi-line comment. A single-line comment starts with two forward slashes "//" and anything written after in the same line is treated as a comment. On the other hand, a multi-line comment is enclosed between "/*" and "*/" that can span multiple lines. Any single-line comment can be set using "/*" and "*/" too (see the comment at the top in "hello.c"). Keep in mind that a multi-line comment cannot be nested within other multi-line comments.

Suppose that we modify the fille "hello.c" by including an extra statement that prints the message "I'm from Earth" too. For, this we do not need to write a separate program from the scratch. We copy "hello.c" to another file "hellom.c" and modify the newly created file "hellom.c" using a text editor. Below are the sequence of comments in a Linux terminal:

```
$ cp  hello.c  hellom.c↩
$ gedit  hellom.c &↩
```

We add an extra statement to print the message "I'm from Earth" along with some comments to illustrate the use comments.

```
/* Simple C Program to print "Hello Mars!"
and "I'm from Earth"
in two lines */
//Header file for printf()
#include <stdio.h>
int main(void)
{
    //print the 1st message
    printf("Hello Mars!\n");
    printf("I'm from Earth\n"); //print the 2nd message
    return 0;
}
```

Listing 3: C program "hellom.c"

Similar to "hello.c", we save "hellom.c", compile and run. This results in the following output:

```
Hello Mars!
I'm from Earth
```

However, the following with nested comment results in error:

```
/* Simple C Program to print "Hello Mars!"
and "I'm /*from*/ Earth"
in two lines */
```

The reason for the error is that the "*/" at the end of the word "from" completes the comment and the left-out part

```
                    Earth"
in two lines */
```

causes error(s).

We should use comments in a reasonable way. Using too many comments makes the code cluttered. Also, we should not use comments for statements that are obvious to understand such as "`//Header file for printf()`" used just above "`#include <stdio.h>`" in "hellom.c".

Return to Lecture Topics