Hello World Program

Problem Statement

Write a C Program to print a `Hello World` on the screen

C Program

```
#include <stdio.h>

int main()
{
    printf("Hello World");
}
```

Source Code with line numbers for explanation

```
1  #include <stdio.h>
2
3  int main()
4  {
5          printf("Hello, World!");
6  }
```

Explanation of the Source Code

The source code is explained below line by line.

Line #1 - Include stdio.h

- This is a way to include the library files (stdio.h - *st*an*d*ard *i*nput and *o*utput) for the method `printf` used in in the program below on the line # 5.
- This is how we reuse the common methods like `printf` defined in a separate library file called `Header` file in `C`.
- Before we use certain methods in the program, we need to include/insert the corresponding library file so that the function can be properly used, else the compiler will throw a `warning`, which we can see at the bottom of the page for an additional information.

Line #2 - Space for legibile reading

- Any whitespaces in the program is not executed, and they are just for a better clarity while reading.
- This whitespace is actually for the humans and not for the computers (compilers) as it does not make any sense or add any value.
- The programmers are recommended to add the necessary whitespaces for the legibility and it acts a visual clue, while a human is reading a program.

Line #3 - int main()

- This is the beginning, entry point of a C Program, like how it is a `door` to a `house` and `booting/turning on` and `logging in` for a PC (Personal Computer).
- For now, we will follow the lines as it is like a grammar of a language.
- It is to be followed exactly the same way, as `int main()`.

Line #4 and #6 - braces as a delimiter

- The `{` and `}` are commonly called as `curly braces` and more specifically as `opening` curly brace and `closing` curly brace.
- Like how we have a `Paragraph` in English language to add a visual clue/aid for a section of words/sentences being grouped together, the curly braces are actually to denote/demarcate/identify the grouped lines/statements of a method.
- For now, we will follow as what it is in the program to mark the boundaries of the method `main`.

Line #5 - printf statement

- This is an `*executable statement* in C - meaning this line of statement will be actually run and it is expected to do some changes in the running code.
- It is actually a `method` or `function` named `printf` defined in the `stdio.h` (Remember, we had included this on the very first line of our source code).

- The `printf` method actually prints something to the screen/console. Think of it like a `*Printer*`, which prints something from the computer (say, a Railway Ticket) from the file to a paper in the printer. It is used for printing any message or more specifically, the *Output* of the Program.
- The actual information/message to be printed is given *inside a pair of double quotes* (`"`). Whatever the set of characters are present inside the double quotes, they get printed as it is. *Note*: *Of course, we have some variations to this, which we will see later*.

Compilation and Output

- As we know, the source code is written in a high level language like English in C. For example, `include` and `print` are meaningful words for a human to understand what the program is supposed to do. But the computers can understand only the machine language or binary language, for which we need to convert the source code into the machine readable form - which is called `*Compilation*`.
- We have an utility called `*gcc*` which stands for `GNU C Compiler`, which is the standard tool in Unix/Linux based Operating Sytem for compiling and running the C/C++ Programs.
- In the Unix/Linux based OS, the `gcc` is available by default/already. We do *NOT* need to explicitly/manually install the C Compiler. That is one other advantage of using th Unix/Linux based machines, as the core of the Operating System binaries are based on C.

```
raghs@LAPTOP-63DBKP7Q:~/cPgms/newDir/demo$ ls -ltrh
total 0
-rw-rw-rw- 1 raghs raghs 60 Apr 12 20:59 HelloWorld.c
raghs@LAPTOP-63DBKP7Q:~/cPgms/newDir/demo$ gcc HelloWorld.c
raghs@LAPTOP-63DBKP7Q:~/cPgms/newDir/demo$ ls -ltrh
total 16K
-rw-rw-rw- 1 raghs raghs   60 Apr 12 20:59 HelloWorld.c
-rwxrwxrwx 1 raghs raghs 8.2K Apr 12 20:59 a.out
```

Explanation of the Compilation

- We invoke the `gcc` compiler by supplying the Program with this filename along with the extension `HelloWorld.c` in the `Command Window` or `Terminal`. *Example:* `gcc HelloWorld.c`

- If the invoation of `gcc` *returns to the prompt immediately without any message*, that means *the compilation was successful*. We are good to proceed further with the execution.

- The `gcc` program however produces a file called `a.out` which is called *intermediate file*, that has the translated/compiled version of the source code written in English like high level language - in `HelloWorld.c`.

- *Note*: The GCC has the default file name as `a.out` unless you supply a file name for the output with an additional command line argument to the program as `-o` (for output), *Example:* `gcc HelloWorld.c -o HelloWorld.out`

- The last few lines of the output shows the same, as the result of `ls -ltrh` (*long listing with the human readable format of file size, in the reverse order of timings modified for the file*) shows that we have two files.

Type	Level	File Name	Remarks
Source Code	High Level	`HelloWorld.c`	Using an editor like `nano`, `vi`, `Notepad`, `Notepad++` etc.,
Intermediate File	Low Level	`a.out`	Using the compiler like `gcc` (in Unix/Linux) or `Turbo CPP` (for Windows) etc.,

Execution and Output

- We are now ready to execute the program, meaning from the intermediate file (which is undrestandable by the Computer).
- Once the program is compiled successfully, we will focus only on the `a.out` file for execution and NOT the source code.

```
raghs@LAPTOP-63DBKP7Q:~/cPgms/newDir/demo$ ls -ltrh
total 16K
-rw-rw-rw- 1 raghs raghs   60 Apr 12 20:59 HelloWorld.c
-rwxrwxrwx 1 raghs raghs 8.2K Apr 12 20:59 a.out
raghs@LAPTOP-63DBKP7Q:~/cPgms/newDir/demo$ ./a.out
Hello, Worldraghs@LAPTOP-63DBKP7Q:~/cPgms/newDir/demo$
```

Explanation of the Execution

- We execute the *intermediate file* (`a.out`) with the syntax `./`. The character `.` is to denote the *current directory* and the character `/` denotes the *exectuion* of the file passed an argument next, which is the `a.out`.
- Upon executing the intermediate file, we get the output as instructed in the Program Source Code `HelloWorld.c`, we get a message printed on the console - which says *Hello, World*.
- *Note*: The last line might be a bit confusing because the output of the program (`Hello, World`) follows the `Terminal prompt` - which is the host name `ragsh@LAPTOP-63DBKP7Q` (which can vary on your machine as per your configuration of the machine name), followed by the `path` which is `/cPgms/newDir/demo` immediately. That is why you see both of them together *Hello, World* `raghs@LAPTOP-63DBKP7Q:~/cPgms/newDir/demo$`.

Extra Information - Source code without including the library/header file

- The program below shows the source code without the `#include <stdio.h>` line and when compiling, the `gcc` (GNU C Compiler) issues a warning that the method `printf` is not known to this and it needs further information as to how it should refer/invoke this method in the program.
- It also gives a suggestion to include the suitable header file in the next line.

```
raghs@LAPTOP-63DBKP7Q:~/cPgms/newDir/demo$ cat HelloWorldWithoutInclude.c
int main()
{
        printf("Hello, World");
}
raghs@LAPTOP-63DBKP7Q:~/cPgms/newDir/demo$ gcc HelloWorldWithoutInclude.c
HelloWorldWithoutInclude.c: In function `main`:
HelloWorldWithoutInclude.c:3:2: warning: implicit declaration of function `printf` [-Wimplicit-function-declaration]
  printf("Hello, World");
  ^~~~~~
HelloWorldWithoutInclude.c:3:2: warning: incompatible implicit declaration of built-in function `printf`
HelloWorldWithoutInclude.c:3:2: note: include `<stdio.h>` or provide a declaration of `printf`
raghs@LAPTOP-63DBKP7Q:~/cPgms/newDir/demo$
```