# CS1217 Spring 2023

# Assignment 1

**Due 11.59 PM, Tuesday 7th February 2023**

**60 Points**

**Instructions**

1. This exercise is being provided to get you started with the environment.
2. Part 0 carries no points.You should sign up for a partner **before** submitting this assignment.
3. **Only one** person of a team (of 2) should submit the assignment. This is the person who will then be responsible for submitting the assignments **for the rest of the semester**.
4. Only Part 1 will be graded. The answers should be submitted as a PDF file. The PDF file should have names of both the team members.
5. This assignment constitutes **1%** of the total course grade.
6. Please name the file as **<Team_Member_1_Name>_CS1217_Assignment1.pdf.** Team member 1 should be the same member as was specified in the signup form.

## Part 0

1. Download and install VirtualBox on your laptop. Import the VM image from the link provided under the “Resources” section on the course webpage using accompanying instructions.

   **Note:** Regarding installing Virtual Machine(VM)- Students, who have Ubuntu 18.04 installed, can directly work with xv6, without using the VM. If you are doing so, you need to install xv6 on your system.
   We do not know yet, about the compatibility of Ubuntu 20.04 with the code base for the labs and the assignments.
   For windows and Mac users, use of VM is recommended to avoid dependencies issues on these operating systems.

## Part 1

1. Write a "hello, world" program in C. The following program can be used to carry out this exercise.

```c
#include<stdio.h>
int main()
/* this is a comment */
{
   int i;
   /* What is the statement below doing?*/
   printf("%s", "Hello World!\n" );
   i = 3;
   printf("%d \n", i);
   return 0;
}
```

Use any editor of your choice (gedit is the easiest editor to get started with, Sublime Text also has a *nix version that can be freely downloaded) to type this into a file named `myhello.c`. It is suggested that you create a separate directory for this assignment (use the `mkdir` command) to create this file in. Answer these questions (a) What, in general, is the `#include` command for? (b) What specifically is `#include <stdio.h>` allowing the program to do? (**15 points**)

2. Compile the "hello, world" program using the command

```
gcc -v myhello.c
```

The -v option (for verbose) shows the name and arguments of each program the compiler executes. Post a screenshot of the output of the process. (**0 points**)

3. If the process in step 2 completes successfully, you should now have an executable file `a.out` which you can run by simply typing the following at the command prompt (in the shell).

```
$   ./a.out
```

Answer the following question (a) Why is the `./` in front of a.out needed? (b) What could potentially be the problem if this wasn't supplied? (**10 points**)

4. Type the following lines into a file named `Makefile`, to be created in the same folder as the files created above (preserve the indentation, more below)

```
myhello: myhello.c myhello.h
    echo first message
    @echo another message
    gcc -o myhello myhello.c
```

Note that the indented lines have to start with a tab character, not a series of spaces. Create a file called `myhello.h` with the following line in it.

```
            const int VAL = 3;
```

Finally, edit `myhello.c` and change it so that it reads as follows:

```
#include <stdio.h>
#include "myhello.h"
int main()
/* this is a comment */
{
   int i;
   /* What is the statement below doing?*/
   printf("%s", "Hello World!\n" );
   i = VAL;
   printf("%d \n", i);
   return 0;
}
```

Type `make` at the command prompt and notice the output. You should now have a file called `myhello` which you can run. Type `make` again and notice that it does ***not*** recompile `myhello`.

Which part of the Makefile is responsible for creating the `myhello` executable? Is changing the name from what we created in step 3 (`a.out`), a function of `gcc` or `make`? (**10 points**)

5. To see an example of separately compiled source files, type the following into a file called `main.c`

```
#include <stdio.h>
#include "dumb.h"
```

```c
int main()
{
    printf("%s", "Hello World!\n" );
    dumb(10);

    return 0;
}
```

and type the following lines into a file called `dumb.c`.

```c
#include <stdio.h>
#include "dumb.h"

void dumb(int n)
{
    printf("n = %d\n", n);

}
```

The header file `dumb.h` should have this line in it:

```c
void dumb(int n);
```

Finally, change your Makefile to read as follows:

```
.c.o:
    gcc -c $*.c
 hello: main.o dumb.o
    gcc -o hello main.o dumb.o
main.o: main.c dumb.h
dumb.o: dumb.c dumb.h
```

Now type make and observe what happens. What is the strange looking new rule at the top of the Makefile now doing?

Change each of the files `main.c`, `dumb.c`, and `dumb.h` (do simple changes; can add whitespaces) and run `make` after each change. Does `make` recompile the binary if none of the source files have been changed? Does a change in one file force the recompilation of all the other, unrelated files? (**10 points**)

6. Use `gdb` to run one of the above programs or one of your own. You should change your Makefile so that it compiles your program with the **-g** option, which facilitates debugging. You can force make to use this option automatically by including it in the `.c.o` rule:

```
.c.o:
    gcc -c -g $*.c
```

To use `gdb` type `gdb filename` where filename is the name of the executable file. If you have not used gdb before, type `help` at the prompt. Experiment with setting breakpoints and single-stepping, and use the commands `list`, `display`, `where`, and `print`. Provide screenshots of the output of these commands as the answer to the question. (**5 points**)

7. Read about `nm`, `od`, or `objdump` tools which provide mechanisms to look at and inspect object (machine language) files. Experiment with these tools and any of the object files or executables that are created in the previous steps. What are the different kinds of information that these tools can provide? Another useful tool that you can look at is `file`. You can find out if a file is text or data by using the `file` command with the filename as the argument. (**10 points**)