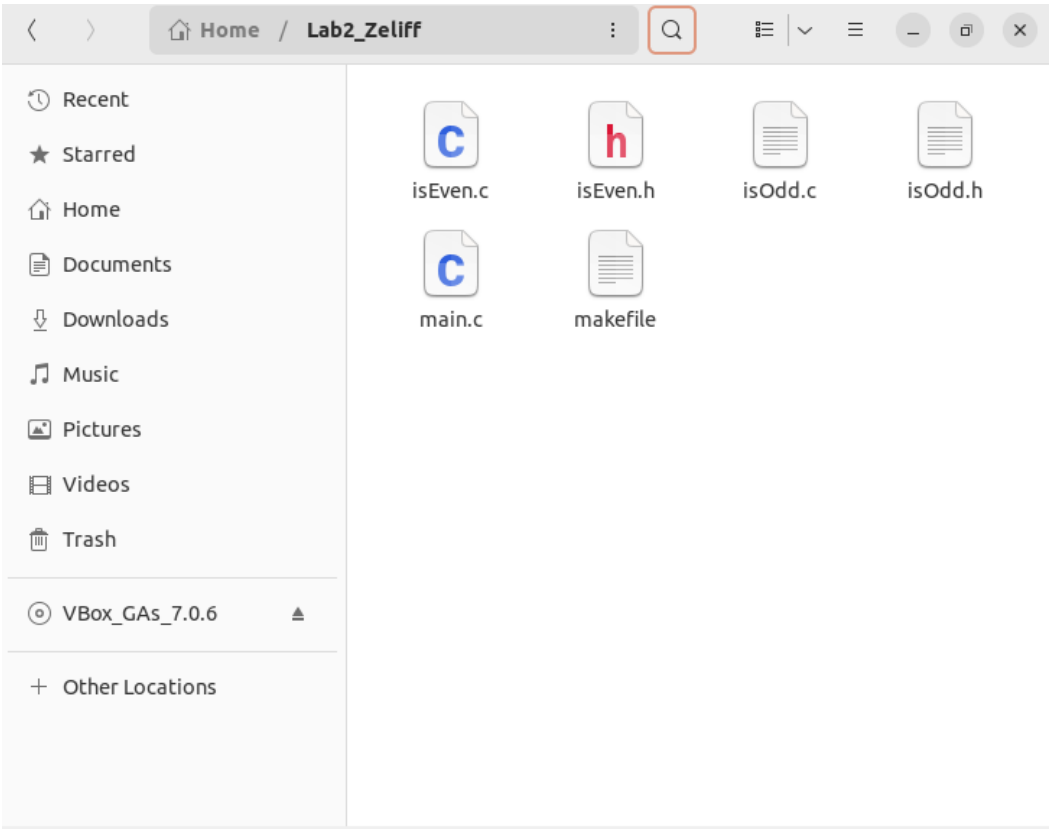
First you should download the .zip file and extract the files into the folder you have been working with so far.

**unzip provided_code.zip**

The unzip command is used to extract files from a zipped archive, after the files are extracted you can delete the .zip file




Currently we have a program in **main.c**, this program will read whether or not a given number is even.

Let's try to compile it!

**gcc main.c**

You should see an undefined reference error. An "Undefined Reference" error occurs when we have a reference to an object in our program and the linker cannot find its definition when it tries to search for it in all the linked object files and libraries.

```c
#include <stdio.h>
#include "isEven.h"
int main() {
    int num;
    printf("Enter an integer: ");
    scanf("%d", &num);
    isEven(num);
    return 0;
}
```

*main.c*

As you can see in the main.c file, we call a function “isEven” which is defined in the isEven.c file.

A fix for this could be to compile both of the files

**gcc main.c isEven.c**

This will create an executable that successfully runs the program, however it's easy to see that as a program requires more and more files to be created it can get increasingly complicated to create the executable a fix for this is in makefiles.

A makefile is a set of commands (similar to terminal commands) with variable names and targets to create object files and to remove them.

It essentially does all the linking and compilation for us in one command.

Let's open the makefile and begin to add some code to it.

*makefile*

```
main: main.o isEven.o
	cc -o main main.o isEven.o
```

Now that the makefile has some commands in it, we can try running it (additional resource)

**make**

The terminal should now of created an executable called “main” you can run this like any other executable

**./main**
**ls**

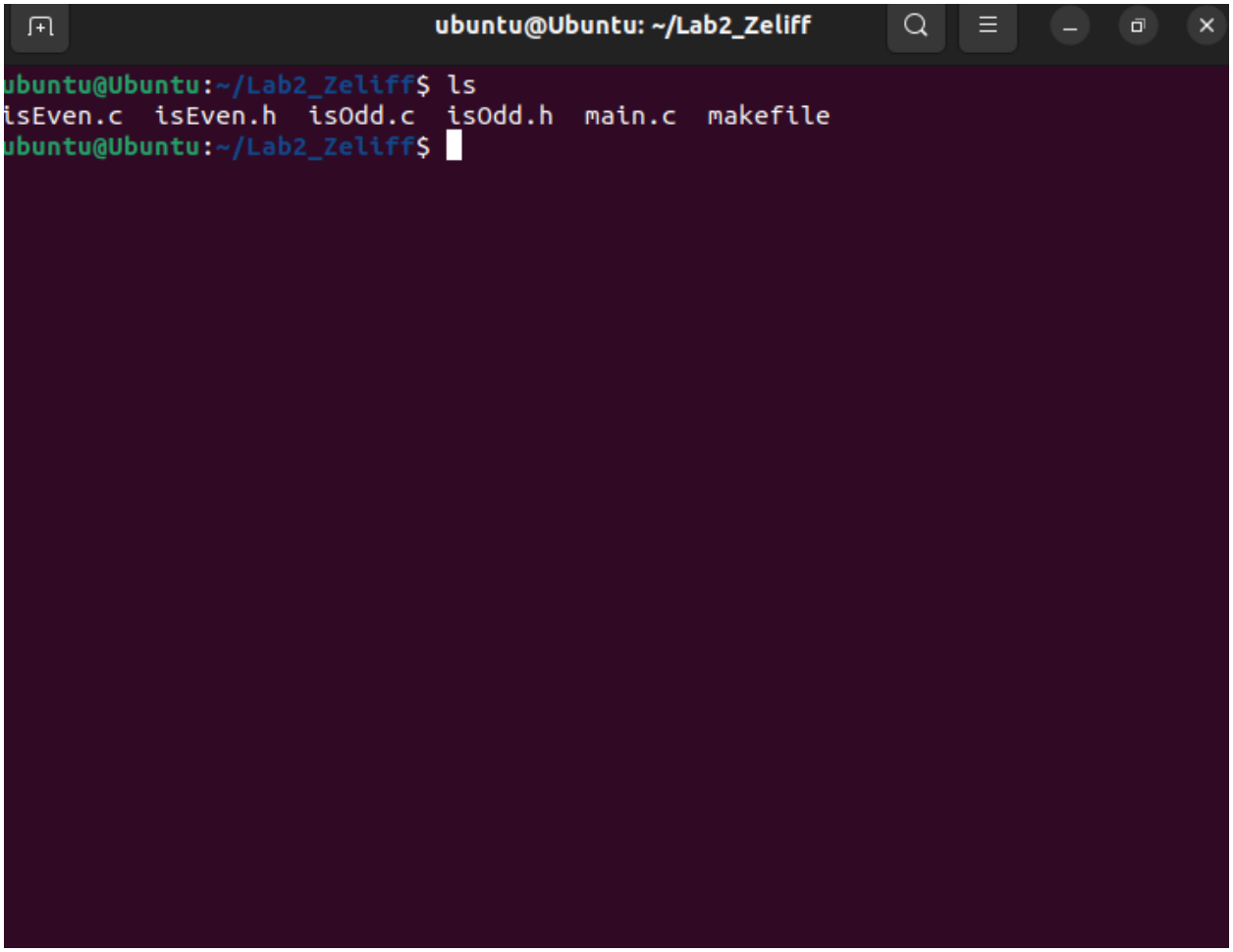
Now you should see that your folder contains multiple .o files, these are object files. They consist of function definitions in binary form, but they are not executable by themselves and by convention their names end with .o. Binary executables file(.exe): These files are produced as the output of a program called a “linker“

If we continue working on our program, there are times where we want to remove the .o files because sometimes the compiler will link or compile files incorrectly and the only way to get a fresh start is to remove all the object and executable files. We can do this by adding “make clean” to the makefile.

*makefile*

```
main: main.o isEven.o
	cc -o main main.o isEven.o
clean:
	rm -f main main.o isEven.o isOdd.o *~
```

**make clean**
**ls**



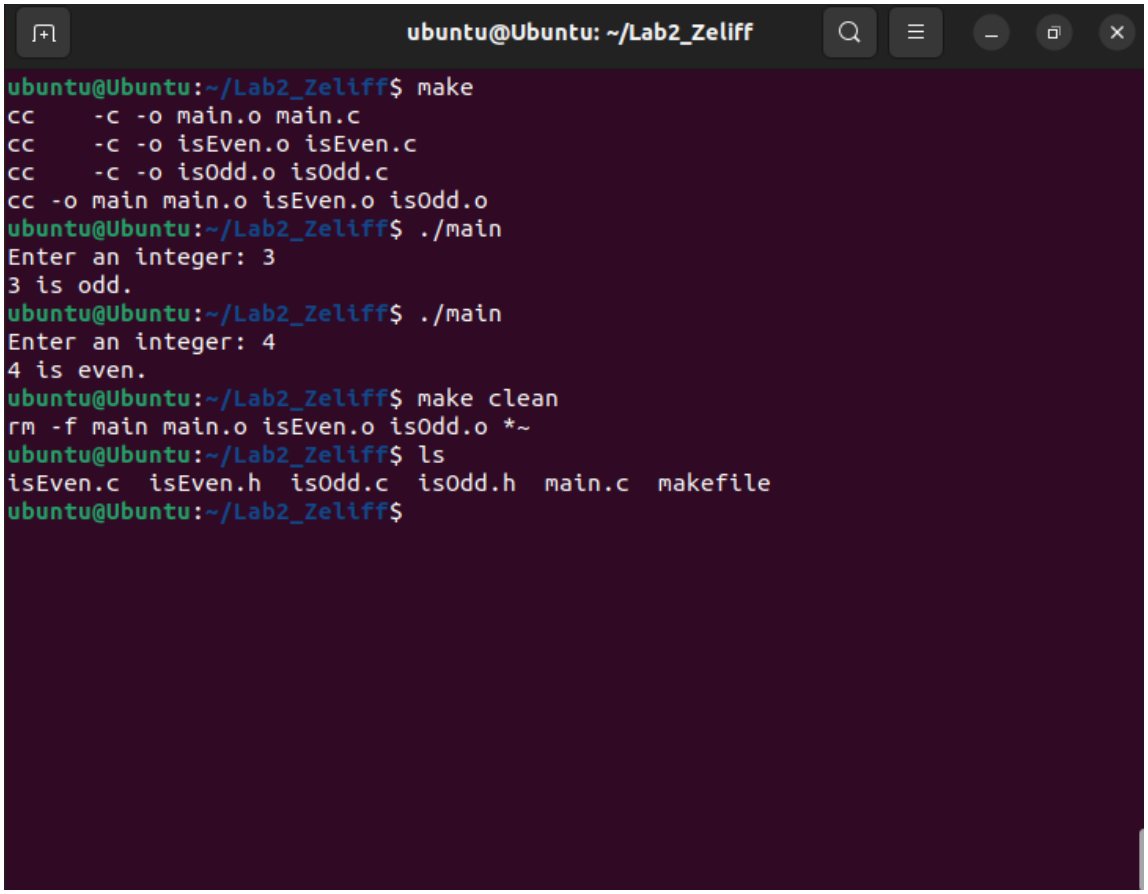

All of the object files should be removed and your folder should look something like this.

Now lets have you try to add some functionality to the program,

**Your goal is to add the “isOdd” functionality to both the isOdd.h, isOdd.c and main.c as well as updating the makefile.**

Hint: The provided isEven.h and isEven.c will be **EXTREMELY** similar to isOdd.h and isOdd.c

*Example output:*




Once your program has the added functionality and a working makefile, there is one more command to learn about in order to turn in the assignment.

The Linux tar command is used for saving several files into an archive file. We can later extract all of the files or just the desired ones in the archive file. tar stands for “tape archive”. (additional resource)

From your home directory:

**tar -zcvf Lab2_YourLastName.tar.gz Lab2_YourLastName**
*tar -zcvf file.tar.gz /path/to/dir/*

A tar file should be created, and **this is what you will be turning into canvas for grading.**

If you want to decompress your tar file and check its contents you can

**tar -zxvf Lab2_YourLastName.tar.gz**

Don’t forget to check the grading rubric!