Exercise 1:

First, open up a terminal and create a directory $ **mkdir lab3**

$ **cd lab3**
Now open up an editor (for example geany, gedit, or scite) and write the following lines:

$ **gedit problem_3.c &**

```
#define Voltage 220
#define current 10

main()
{
float Resistor;
Resistor = Voltage / current;
printf("R = %6.2f \n", Resistor);
}
```

Save the file and switch to the terminal window. Precompile your created program by using the gcc command and the E option. With this option, only the preprocessor is activated. It will execute and output all commands that start with a pound "#" sign.

guest@ubuntu$ **gcc -E problem_3.c -o problem_3_pre.c**

This will cause the preprocessed file to be written to the file "problem_3_pre.c ".

Now, view the file contents and write everything to the answer sheet.

To view the file you may either load it into the editor, or view it from within the terminal by using the "cat" or "more" command, which is much faster.

guest@ubuntu$ **cat problem_3_pre.c**
Notice the difference between the two files, **problem_3.c** and **problem_3_pre.c .** Now the

purpose of the precompiler should be clear to you.

Proceed to compile the program. You may either compile the precompiled file, or the original file.

guest@ubuntu$ **gcc problem_3.c or**

guest@ubuntu$ **gcc problem_3_pre.c**

Oops ! You received a number of error messages. Don't worry, we will debug them. First, read the first line of the output. It is a warning message. It says something like the following:

```
Problem_3.c:4:1: warning: type specifier missing, defaults to
'int'
      [-Wimplicit-int]
main()
^
problem_3.c:8:3: warning: implicitly declaring library function
'printf'with type 'int (const char *, ...)'
  printf("R = %6.2f \n", Resistor);
  ^
problem_3.c:8:3: note: include the header <stdio.h> or explicitly
provide a declaration for 'printf'
2 warnings generated.
```

Basically, it says that we need to specify what type the main is expected to return (int). Also, it can not recognize “printf”. And the warning message is based on line number 8 in the source code problem_3.c .

Once we correct the “printf” issue, we will fix most of the problems. So, lets do it.

Printf is a function that needs to be defined before it can be used. Its definition is included in the **stdio.h** header file. Simply including this file before the main function should fix the problem.

```
#define Voltage 220
#define current 10
#include <stdio.h>

 main()
 {
 float Resistor;
 Resistor = Voltage / current;
 printf("R = %6.2f \n", Resistor);
}
```

Now compile it again with the following command $ **gcc problem_3.c**

If you still have an issue at the line where printf is located, then check your apostrophies. They must look like " and not like “.

Solución 1

problem_3.c (antes)

```c
#define Voltage 220
#define current 10

main()
{
    float Resistor;
    Resistor = Voltage / current;
    printf("R = %6.2f \n", Resistor);
}
```

Al compilar este código, obtenía el resultado siguiente…

problem_3.c:4:1: warning: return type defaults to 'int' [-Wimplicit-int]
4 | main()
| ^~~~
problem_3.c: In function 'main':
problem_3.c:8:5: warning: implicit declaration of function 'printf' [-Wimplicit-function-declaration]
8 | printf("R = %6.2f \n", Resistor);
| ^~~~~~
problem_3.c:8:5: warning: incompatible implicit declaration of built-in function 'printf'
problem_3.c:1:1: note: include '<stdio.h>' or provide a declaration of 'printf'
+++ |+#include <stdio.h>
1 | #define Voltage 220

…para solucionar estos errores, tuve que agregar un archivo de cabecera llamado *stdio.h* y agregar un valor de retorno entero antes del nombre del método *main()*.

problem_3.c (después)

```c
#define Voltage 220
#define current 10
#include <stdio.h>

int main()
{
    float Resistor;
    Resistor = Voltage / current;
    printf("R = %6.2f \n", Resistor);
}
```

Comandos usados

```
gitpod /workspace/LenguajesDeProgramacion $ mkdir lab3
gitpod /workspace/LenguajesDeProgramacion $ cd lab3
gitpod /workspace/LenguajesDeProgramacion/lab3 $ ls
problem_3.c
gitpod /workspace/LenguajesDeProgramacion/lab3 $ gcc -E problem_3.c -o problem_3_pre.c
gitpod /workspace/LenguajesDeProgramacion/lab3 $ cat problem_3_pre.c
# 1 "problem_3.c"
# 1 "<built-in>"
# 1 "<command-line>"
# 31 "<command-line>"
# 1 "/usr/include/stdc-predef.h" 1 3 4
# 32 "<command-line>" 2
# 1 "problem_3.c"



main()
{
   float Resistor;
   Resistor = 220 / 10;
   printf("R = %6.2f \n", Resistor);
}
gitpod /workspace/LenguajesDeProgramacion/lab3 $ gcc problem_3.c
problem_3.c:4:1: warning: return type defaults to ‘int’ [-Wimplicit-int]
   4 | main()
     | ^~~~
problem_3.c: In function ‘main’:
problem_3.c:8:5: warning: implicit declaration of function ‘printf’ [-Wimplicit-function-
declaration]
   8 |     printf("R = %6.2f \n", Resistor);
     |     ^~~~~~
problem_3.c:8:5: warning: incompatible implicit declaration of built-in function ‘printf’
problem_3.c:1:1: note: include ‘<stdio.h>’ or provide a declaration of ‘printf’
  +++ |+#include <stdio.h>
   1 | #define Voltage 220
gitpod /workspace/LenguajesDeProgramacion/lab3 $ gcc problem_3.c
problem_3.c:5:1: warning: return type defaults to ‘int’ [-Wimplicit-int]
   5 | main()
     | ^~~~
gitpod /workspace/LenguajesDeProgramacion/lab3 $ gcc problem_3.c
problem_3.c:5:1: warning: return type defaults to ‘int’ [-Wimplicit-int]
   5 | main()
     | ^~~~
problem_3.c: In function ‘main’:
problem_3.c:9:21: warning: format ‘%f’ expects argument of type ‘double’, but argument 2
has type ‘int’ [-Wformat=]
   9 |     printf("R = %6.2f \n", Resistor);
     |                    ~~~~^      ~~~~~~~~
     |                        |      |
```

```
    |                 double int
    |              %6.2d
gitpod /workspace/LenguajesDeProgramacion/lab3 $ gcc problem_3.c
problem_3.c:5:1: warning: return type defaults to ‘int’ [-Wimplicit-int]
   5 | main()
     | ^~~~
gitpod /workspace/LenguajesDeProgramacion/lab3 $ gcc problem_3.c
gitpod /workspace/LenguajesDeProgramacion/lab3 $
```

Exercise 2

Save this c code into a file (for example problem_3b.c) and switch to the terminal window. Compile the code with gcc.

```
#define Voltage 5
#include <stdio.h>

float resistor_fun(float i)
{
        float b;
        b=Voltage/i;
        return b;
}

 main()
 {
 float Resistor, current;

 printf("Enter the current value : ");
 scanf("%f", &current);

 Resistor = resistor_fun( current );

 printf("The required resistor should be  ");
 printf("%1.0f Ohms.\n", Resistor);
 }
```

$ **gcc problem_3b.c**

Now run the executable program.

The name of the created program is a.out. You may rename the program with the **mv** command, or you can specify the program name directly at compile time by using the **o** option as follows.

(gcc o myprog problem_3b.c)

$ **./a.out**

The **./** specifies that the program is in the current directory and not in the operating system predefined program places.

Now write down the screen outputs of the program onto your answer sheet.

Solución 2

problem_3b.c (antes)

```c
#define Voltage 5
#include <stdio.h>

float resistor_fun(float i)
{
    float b;
    b = Voltage/i;
    return b;
}

main()
{
    float Resistor, current;

    printf("Enter the current value: ");
    scanf("%f", &current);

    Resistor = resistor_fun( current );

    printf("The required resistor should be ");
    printf("%1.0f Ohms.\n", Resistor);
}
```

Al compilar este código, obtenía el resultado siguiente…

problem_3b.c:11:1: warning: return type defaults to ‘int’ [-Wimplicit-int]
11 | main()
| ^~~~

…para solucionar este error, tuve que agregar un valor de retorno entero antes del nombre del método *main()*.

problem_3b.c (después)

```c
#define Voltage 5
#include <stdio.h>

float resistor_fun(float i)
{
    float b;
    b = Voltage/i;
    return b;
}

int main()
```

```
{
    float Resistor, current;

    printf("Enter the current value: ");
    scanf("%f", &current);

    Resistor = resistor_fun( current );

    printf("The required resistor should be ");
    printf("%1.0f Ohms.\n", Resistor);
}
```

Comandos usados

```
gitpod /workspace/LenguajesDeProgramacion/lab3 $ gcc problem_3b.c
problem_3b.c:11:1: warning: return type defaults to ‘int’ [-Wimplicit-int]
  11 | main()
     | ^~~~
gitpod /workspace/LenguajesDeProgramacion/lab3 $ gcc problem_3b.c
gitpod /workspace/LenguajesDeProgramacion/lab3 $ gcc o myprog problem_3b.c
gcc: error: o: No such file or directory
gcc: error: myprog: No such file or directory
gitpod /workspace/LenguajesDeProgramacion/lab3 $ gcc o myprog.out problem_3b.c
gcc: error: o: No such file or directory
gitpod /workspace/LenguajesDeProgramacion/lab3 $ gcc -o myprog.out problem_3b.c
gitpod /workspace/LenguajesDeProgramacion/lab3 $ ./myprog.out
Enter the current value: 5.0
The required resistor should be 1 Ohms.
gitpod /workspace/LenguajesDeProgramacion/lab3 $
```

Exercise 3

Now we are going to compile the above code again, but first we will separate it into smaller pieces. Place each function in a separate file.

We have separated the two files. Note that the preprocessor commands are placed only in the related files. That is, the " **#include <stdio.h>** " definition is placed in the file where the **printf** and **scanf** functions are used, and the " **#define Voltage 5** " is placed in the file where the term **Voltage** is being used.

Due to the separation, we must let the main program know about the existence of the function resistor_fun(). Otherwise we may obtain a compilation error. So, before the usage of the function resistor_fun, we must place the following line anywhere before the function is being used.

extern float resistor_fun(float);

The first term **extern** specifies that the function is an external function. The second term **float** specifies that the return value of the function is of type *float*. The **float** definition inside the parenthesis specifies that the input value to the function is of type *float (İt is not required to be defined)*.

```
#include <stdio.h>

main()
{
float Resistor, current;

printf("Enter the current value : ");
scanf("%f", &current);

Resistor = resistor_fun( current );

printf("The required resistor should be  ");
printf("%1.0f Ohms.\n", Resistor);
}
```

File: program_3c-1.c

```
#define Voltage 5

float resistor_fun(float i)
{
      float b;
      b=Voltage/i;
      return b;
}
```

File: program_3c-2.c

Now let us compile the two files one by one.

$ **gcc c problem_3c1.c**

$ **gcc c problem_3c2.c**

If no errors occur, then you should have obtained two object files, which have a .o extension. Type **ls** to see them. They should have these names: **problem_3c1.o** and

problem_3c2.o.
Now link them to a single program.

$ **gcc o myprog problem_3c1.o problem_3c2.o** and run the program as follows.
$ **./myprog**

Make sure that the output is correct. For example, for a 1 amp current, the resistor value should be 5 Ohms.

Solución 3

problem_3c1.c

```c
#include <stdio.h>

main()
{
    float Resistor, current;

    printf("Enter the current value: ");
    scanf("%f", &current);

    extern float resistor_fun(float);

    Resistor = resistor_fun( current );

    printf("The required resistor should be ");
    printf("%1.0f Ohms.\n", Resistor);
}
```

En este código, agregue la línea…

```c
extern float resistor_fun(float);
```

…antes de la llamada de la función *resistor_fun()*, debido a que se hizo la separación de códigos en dos archivos, por lo que debo hacer saber al programa principal la existencia de la función *resistor_fun()*, ya que si no lo hago, habría un error de compilación.

problem_3c2.c

```c
#define Voltage 5

float resistor_fun(float i)
{
    float b;
    b = Voltage/i;
    return b;
}
```

Comandos usados

gitpod /workspace/LenguajesDeProgramacion/lab3 $ gcc c problem_3c1.c
gcc: error: c: No such file or directory
gitpod /workspace/LenguajesDeProgramacion/lab3 $ gcc -c problem_3c1.c
gitpod /workspace/LenguajesDeProgramacion/lab3 $ gcc -c problem_3c2.c
gitpod /workspace/LenguajesDeProgramacion/lab3 $ ls
myprog.out problem_3.c problem_3c1.o problem_3c2.o
problem_3b.c problem_3c1.c problem_3c2.c problem_3_pre.c

```
gitpod /workspace/LenguajesDeProgramacion/lab3 $ gcc -o myprog problem_3c1.o
problem_3c2.o
gitpod /workspace/LenguajesDeProgramacion/lab3 $ ./myprog
Enter the current value: 1
The required resistor should be 5 Ohms.
gitpod /workspace/LenguajesDeProgramacion/lab3 $
```

Exercise 4

Write a small program like this:

```c
/* My first code in C
   Gildardo Sanchez
*/

/* Instructions starting with pound (#) are preprocessor commands */

#define BIGNUM 1000000
#include<stdio.h>


int main(void)
{
        int a= BIGNUM;

        printf("Hello, world, I am %d happy units\n", a);

return 0;
}
```

Then, in a terminal try:

```
MacBookPro:Desktop gildardo$ gcc -c prog01.c
```

What happens? Why?

Now, try:

```
MacBookPro:Desktop gildardo$ gcc prog01.c -o prog
```

What happens now? Why?

Solución

En el primer caso, el programa compila correctamente y se crea un archivo ejecutable llamado *prog01.o*; en el segundo caso, el programa también compila correctamente, pero esta vez se crea un archivo ejecutable llamado *prog*, el cual básicamente linkea el contenido de *prog01*.c en el otro para que pueda ser ejecutado.

Now, let's create a complete set of files to perform a separated compilation.

Assume we want to create a very simple library with two functions, to compute the maximum and the minimum of two integer numbers.

So, we need to create two files for the library. One will contain only the declarations of the functions. That one is going to be a header file, so the extension is going to be ".h".

```
/* This is my first libary in C !

   This file contains the declarations of the functions of my library
*/

int the_biggest(int x, int y);

int the_smallest(int x, int y);
```

Then, we need the actual definitions of the functions. This is a real C program (without main):

```
/* This is my first libary in C !

   This file contains the definitions of the functions of my library
*/

int the_biggest(int x, int y)
{
  if (x>y)
    return x;
  else
    return y;
}

int the_smallest(int x, int y)
{
  if (x<y)
    return x;
  else
    return y;
}
```

We can compile this program, but we cannot do it like usual, because there is no main function. Just in case you don't believe me, try:

```
MacBookPro:Desktop gildardo$ gcc compare.c -o compare
```

You may have got something like:

```
MacBookPro:Desktop gildardo$ gcc compare.c -o compare
Undefined symbols for architecture x86_64:
  "_main", referenced from:
     implicit entry/start for main executable
ld: symbol(s) not found for architecture x86_64
clang: error: linker command failed with exit code 1 (use -v to see invocation)
```

Now, do:

```
MacBookPro:Desktop gildardo$ gcc -c compare.c
```

No complaints now, right? And we got a file named compare.o that is our target file for the library.

Let's create a file with a main to test our library:

```
/* My first code in C that uses my library

*/

/* Instructions starting with pound (#) are preprocessor commands */

#include <stdio.h>
#include "compare.h"

int main(void)
{
        int a= 10;
        int b = 4;


        printf("The biggest is: %d\n", the_biggest(a,b));

        printf("The smallest is: %d\n", the_smallest(a,b));
return 0;
}
```

We can then compile this program:

```
MacBookPro:Desktop gildardo$ gcc -c my_main.c
```

And link all:

```
MacBookPro:Desktop gildardo$ gcc my_main.o compare.o -o my_prog
```

Now we have our binary (executable) code. Try it!

Solución 4

prog01.c

```c
/* My first code in C
   Jesus Monterrubio
*/

/* Instructions starting with pound (#) are preprocessor commands */

#define BIGNUM 1000000
#include<stdio.h>

int main(void)
{
    int a= BIGNUM;

    printf("Hello, world, I am %d happy units\n", a);

return 0;
}
```

(Respuesta escrita debajo de las preguntas correspondientes, pero puesta aquí de nuevo por si no se llegará a ver)
De acuerdo con los comandos utilizados, en el primer caso, el programa compila correctamente y se crea un archivo ejecutable llamado *prog01.o*; en el segundo caso, el programa también compila correctamente, pero esta vez se crea un archivo ejecutable llamado *prog*, el cual básicamente linkea el contenido de *prog01*.c en el otro para que pueda ser ejecutado.

compare.h

```
/* This is my first library in C!

    This file contains the declarations of the functions of my library
*/

int the_biggest(int x, int y);

int the_smallest(int x, int y);
```

Este programa tiene las declaraciones de las funciones que se usarán después, básicamente funciona como una librería.

compare.c

```
/* This is my first library in C!

    This file contains the declarations of the functions of my library
*/

int the_biggest(int x, int y)
{
    if(x>y)
        return x;
    else
        return y;
}

int the_smallest(int x, int y)
{
    if(x<y)
        return x;
    else
        return y;
}
```

En este programa se desarrollan las funciones que se utilizarán durante el proceso de las comparaciones más adelante, además, no se puede compilar, ya que no cuenta con un método *main()*, por lo que surge el siguiente error…

/usr/bin/ld: /usr/lib/gcc/x86_64-linux-gnu/9/../../../x86_64-linux-gnu/Scrt1.o: in function `_start':
(.text+0x24): undefined reference to `main'
collect2: error: ld returned 1 exit status

…el cual debe arreglarse añadiendo un método principal o vinculando el archivo con otro programa que tenga el *main()*.

my_main.c

```c
/* My first code in C that uses my library

*/

/* Instructions starting with pound (#) are preprocessor commands */

#include <stdio.h>
#include "compare.h"

int main(void)
{
    int a= 10;
    int b = 4;

    printf("The biggest is: %d\n", the_biggest(a,b));
    printf("The smallest is: %d\n", the_smallest(a,b));
return 0;
}
```

Este programa representa el método principal que permite la compilación de todo, y ya que los archivos correspondientes son linkeados, se crea el archivo *my_prog* que podrá ser ejecutado demostrando cuál de los dos números contenidos en las variables *a* y *b* es mayor y menor.

Comandos usados

```
gitpod /workspace/LenguajesDeProgramacion/lab3 $ gcc -c prog01.c
gitpod /workspace/LenguajesDeProgramacion/lab3 $ gcc prog01.c -o prog
gitpod /workspace/LenguajesDeProgramacion/lab3 $ gcc compare.c -o compare
/usr/bin/ld: /usr/lib/gcc/x86_64-linux-gnu/9/../../../x86_64-linux-gnu/Scrt1.o: in function
`_start':
(.text+0x24): undefined reference to `main'
collect2: error: ld returned 1 exit status
gitpod /workspace/LenguajesDeProgramacion/lab3 $ gcc -c compare.c
gitpod /workspace/LenguajesDeProgramacion/lab3 $ gcc -c my_main.c
gitpod /workspace/LenguajesDeProgramacion/lab3 $ gcc my_main.o compare.o -o my_prog
gitpod /workspace/LenguajesDeProgramacion/lab3 $ ./my_prog
The biggest is: 10
The smallest is: 4
gitpod /workspace/LenguajesDeProgramacion/lab3 $
```

CONCLUSIONES

Después de seguir todas las instrucciones al pie de la letra, aprendí varias cosas sobre el uso de la terminal de Linux, el significado de varios comandos y procesos, un poco acerca de C (tanto de su lógica como de su sintaxis), también aprendí sobre todo lo relacionado a la compilación, detalles acerca del preprocesador, etc. Realizar este laboratorio fue una experiencia enriquecedora que me gustaría repetir con otros temas de por medio.