**Assignment 1: C Programming**

**Submitted By:Aryan agrahari Roll No. -51910037**

Ques1. Following questions are meant to check your if you can code in multiple files. In all the questions you need to **make different .c files for each function** and one file for main. Finally tell the process to compile, link and execute them

Write a C program to take two integer inputs from user and print the sum and the product of them. Sum and Product functions should be made in separate files

**Solution**: for this we need to create following files

**Main.c**

#include<stdio.h>

#include "add.h"

#include "mul.h"

int main()

{

int add1,mul1;

add1=add(1,2);

mul1=mul(1,2);

printf("Addition is :%d\n Multiplication is: %d",add1,mul1);

return 0;

}

**Add.c**

#include "add.h"

int add(int a,int b)

{

return a+b;

}

**mul.c**

#include "mul.h"

int mul(int a,int b)

{

return a\*b;

}

**mul.h**

#ifndef MUL\_H\_INCLUDED

#define MUL\_H\_INCLUDED

int mul(int a,int b);

#endif

**add.h**

#ifndef ADD\_H\_INCLUDED

#define ADD\_H\_INCLUDED

int add(int a,int b);

#endif

**For compiling the file we need to run these commands in command prompt at the location of program**

gcc main.c –o main.o –c

gcc add.c –o add.o -c

gcc mul.c –o mul.o -c

gcc add.o mul.o main.o –o prog.o

prog.o

**output:** addition :3 multiplication :2

**Ques2.** Write the program of reading and writing a file that takes name of the files as argument of main.

**Solution:**

#include <stdio.h>

int main(int argc, char \*argv[])

{

FILE \*fptr1,\*fptr2;

int i;

char ch;

printf("Number of arguments : %d\n", argc);

if( (fptr1 = fopen(argv[1], "r") ) == NULL )

{

printf("Cannot open file: %s\n", argv[1]);

return -1;

}

fptr2 = fopen(argv[2], "w");

while( !feof(fptr1) )

{

ch = getc(fptr1);

if(ch != EOF) {

putc(ch, fptr2);

}

}

fclose(fptr1);

fclose(fptr2);

printf("Content of %s copied to %s\n", argv[1], argv[2]);

return 0;

}

**Ques3.** What is pre-processor directives? Explain with an example.

**Solution:**

* Before a C program is compiled in a compiler, source code is processed by a program called preprocessor. This process is called preprocessing.
* Commands used in preprocessor are called preprocessor directives and they begin with “#” symbol.

**Example:**

|  |  |
| --- | --- |
| Macro | **Syntax:** #defineThis macro defines constant value and can be any of the basic data types. |
| Header file inclusion | **Syntax:** #include <file\_name> The source code of the file “file\_name” is included in the main program at the specified place. |
| Conditional compilation | **Syntax:** #ifdef, #endif, #if, #else, #ifndef Set of commands are included or excluded in source program before compilation with respect to the condition. |
| Other directives | **Syntax:** #undef, #pragma #undef is used to undefine a defined macro variable. #Pragma is used to call a function before and after main function in a C program. |

#include <stdio.h>

// macro definition

#define LIMIT 5

int main()

{

     for (int i = 0; i < LIMIT; i++)

{

        printf("%d \n",i);

     }

    return 0;

}

**Ques4.** What is “include guards”?

**Solution:**

In the C and C++ programming languages, an #include guard, sometimes called a macro guard, header guard or file guard, is a particular construct used to avoid the problem of double inclusion when dealing with the include directive.

The C preprocessor processes directives of the form #include <file> in a source file by locating the associated file on disk and transcluding ("including") its contents into a copy of the source file known as the translation unit, replacing the include directive in the process. The files included in this regard are generally header files, which typically contain declarations of functions and classes or structs. If certain C or C++ language constructs are defined twice, the resulting translation unit is invalid. #include guards prevent this erroneous construct from arising by the double inclusion mechanism.

**Example**

#Pretty much every header file should follow the include guard idiom:

my-header-file.h

#ifndef MY\_HEADER\_FILE\_H

#define MY\_HEADER\_FILE\_H

// Code body for header file

#endif

This ensures that when you #include "my-header-file.h" in multiple places, you don't get duplicate declarations of functions, variables, etc.

**Ques5.** WAP that reads temperature in degree Celsius from a file and write temperature in degree Fahrenheit in other file.

**Solution:**

#include<stdio.h>

int main()

{

FILE \*fp1,\*fp2;

int temp;

fp1=fopen("celcius.txt","r");

fp2=fopen("fahrenheit.txt","w");

if(fp1==NULL || fp2 == NULL)

{

printf("cannot open file \n");

return 0;

}

while (fscanf(fp1,"%d\n",&temp)!=EOF)

fprintf(fp2,"%d\n",(temp\*(9/5))+32);

printf("file written");

fclose("celcius.txt");

fclose("fareheit.txt");

return 0;

}

**Ques 6.** Read the dimensions of container from a file and write the volume of the container in output file.

**Solution.**

#include<stdio.h>

int main()

{

FILE \*fp1,\*fp2;

int length,breadth,height;

fp1=fopen("dimensions.txt","r");

fp2=fopen("volume.txt","w");

if(fp1==NULL || fp2 == NULL)

{

printf("cannot open file \n");

return 0;

}

while (fscanf(fp1,"%d %d %d\n",&length,&breadth,&width)!=EOF)

fprintf(fp2,"%d\n",(lenght\*breadth\*height);

printf("file written");

fclose("dimensions.txt");

fclose("volume.txt");

return 0;

}

**Ques7.** What is relative path of a file and what is absolute path of a file? What is the difference between them?

**Solution:**

**Absolute Path:** An absolute or full path points to the same location in a file system, regardless of the current working directory. To do that, it must include the root directory.

**Relative path:** A relative path starts from some given working directory, avoiding the need to provide the full absolute path. A filename can be considered as a relative path based at the current working directory. If the working directory is not the file's parent directory, a file not found error will result if the file is addressed by its name.

**Difference between Absolute and Relative Path**

**Meaning**

Absolute path, also referred to as file path or full path, refers to a specific location in the file system, irrespective of the current working directory.

A relative path, on the contrary, refers to the location of a directory using current directory as a reference, which avoids the need to specify the full absolute path.

**URL**

There are two basic choices when it comes to selection of URLs, absolute URL and relative URL. An absolute URL points to a very specific location on the Web and contains all the necessary information to locate a resource. Absolute URLs must be used to link to other websites that are not located on the same domain.

**Ques8.** Following is the directory structure

Users (folder)

|-Desktop (folder)

|      |-SupportiveFiles (folder)

|      |         |-file1.txt (file)

|-Cprogramming

|         |-main.c

Write main.c so that the executable file of this program can read file1.txt using relative path onto the console(terminal) of your system.

**Solution:**

#include<stdio.h>

int main()

{

FILE \*fp1,\*fp2;

char l[255];

fp1=fopen("c:\\Users\\aryan\\desktop\\c workspace\\file1.txt","r");

if(fp1==NULL || fp2 == NULL)

{

printf("cannot open file \n");

return 0;

}

while (fscanf(fp1,"%s\n",&l)!=EOF)

fprintf(fp2,"%d\n",l;

fclose(fp1);

fclose(fp2);

return 0;

}