MULTI-FILE C PROGRAM DEVELOPMENT

PRABHJOT BHATIA - 102027
SAMANYU BHATE - 102026
SWAMIL RANDIVE - 102062

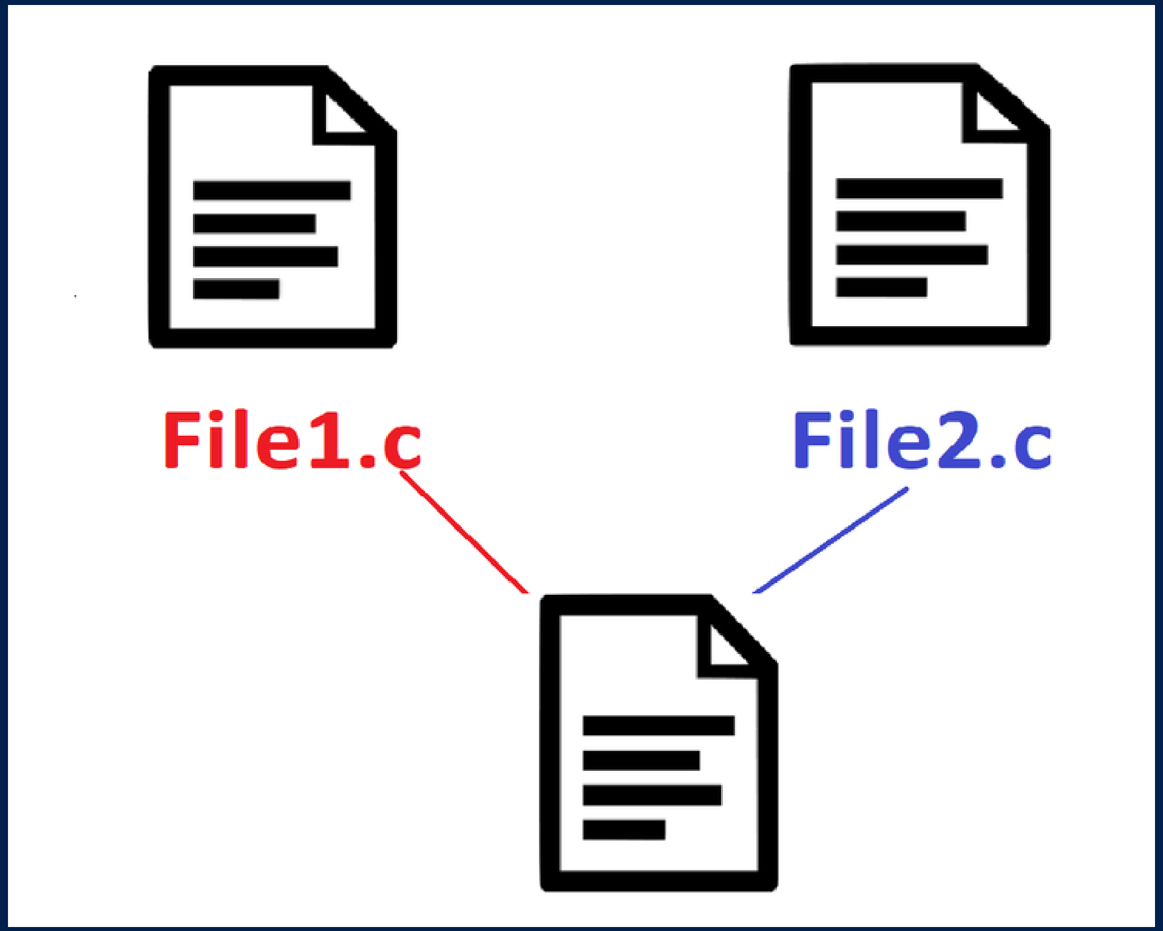
PRE-REQUISITE

- Most of our current work was and is done using only single. However actual large-scale projects are done using combing multiple c files.
- Using a single c-file to write the entire code generates certain problems. As the code gets bigger and complicated it is difficult for even the person who wrote the code to properly function around the code.
- In the case of any problem/bug, it is very difficult to navigate and locate it.
- But having a multi file program in which different functions are stored in different files, helps u navigate through each and locate any problem or find a piece of code.

How should the files be divided

- In general, the code should be divided along natural divisions in the program. Each file should contain a group of functions that do related things.
- We can imagine another program that might use some of these functions: include in one file all the functions that this other program would require.
- We can create a file that is common to two or more programs to reduce writing the same type of code twice.
- Note- documenting the code is important to later use it properly.

Overview



This is how the working of a multi file c program looks like.

HOW DOES IT WORK

- First, we create multiple files
- In the command line, while using the gcc command, we can compile all these files together.

```
command- gcc -o run file1.c file2.c file3.c
```

- Or we can compile them individually and then link them together
- By linking the files, the files can access each other and access the various functions in it.
- They also have a common user made header file, which can be used by all the files.
- This way, to edit a code only a single file may need to be changed, to change that part of the code, helping in large-scale projects.

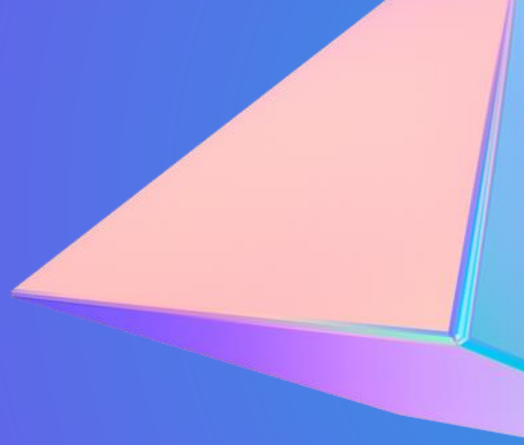
Header files

Declarations are frequently moved into separate files, called header files. If a piece of code is named test.C, the corresponding header file is traditionally named test.h.The main purpose of header files is to make definitions and declarations accessible to functions in more than one file. For example, if functions from two files need to access the same global constant (e.g. a definition of a number that will stay constant throughout all files), that variable should be defined in a header file. Conversely, if a declaration is only meant to be used by the functions in one code file, it could be left in that code file and need not be put it in a header file.

IN THE HEADER FILE

The header file should generally contain the following:

- definitions of global variables and global constants
- definitions of classes, structs, unions, enumerated types
- typedef statements used to create type names
- declarations of functions, called "prototypes"
- include statements for other files, including C library files
- comments associated with all of the above



CALLING HEADER FILES

#include <filename>

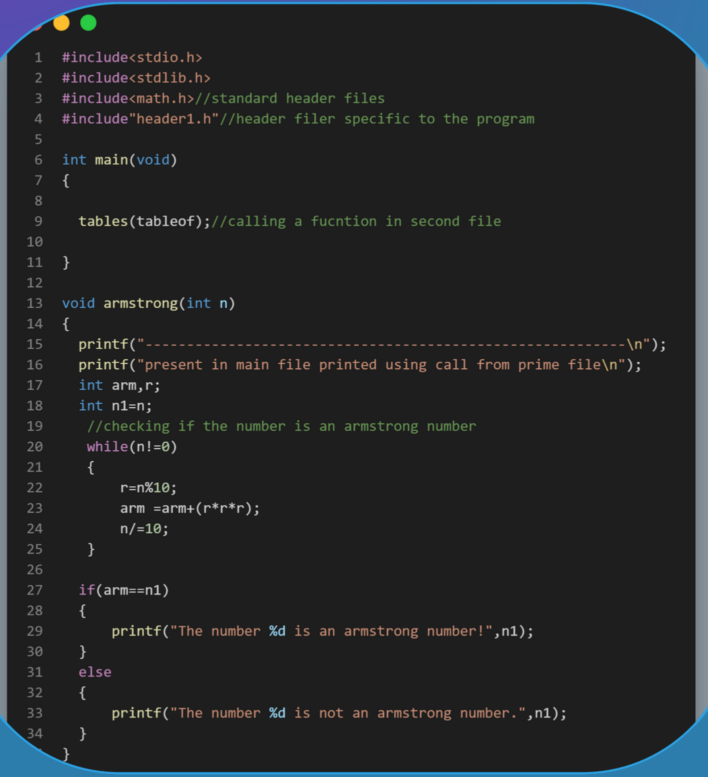
This method is normally used to include header files for the C standard library and other header files associated with the target platform.

#include "filename"

Normally used to include programmer-defined header files and typically includes same directory as the file containing the directive.

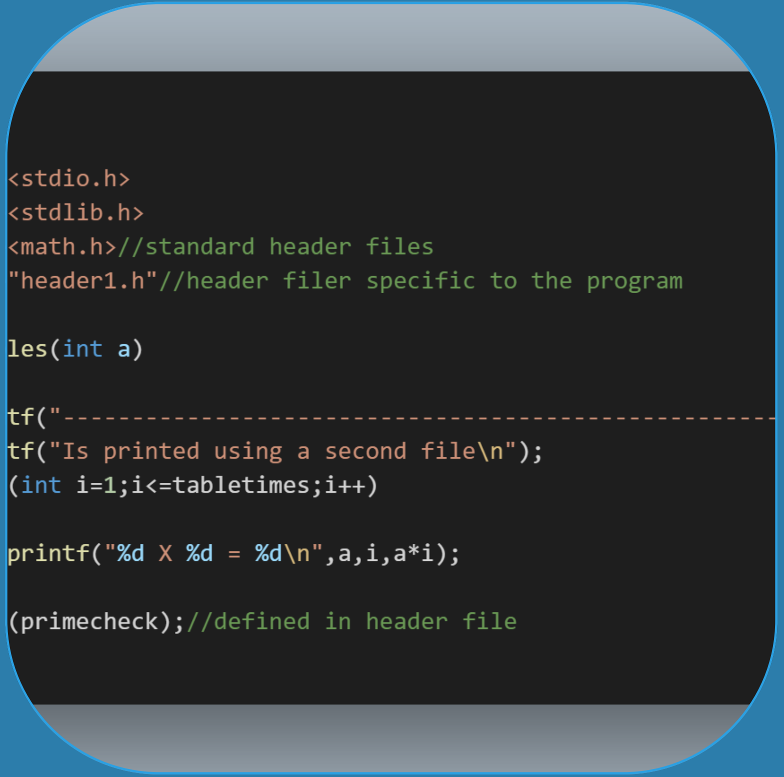
EXAMPLE

THE FILES



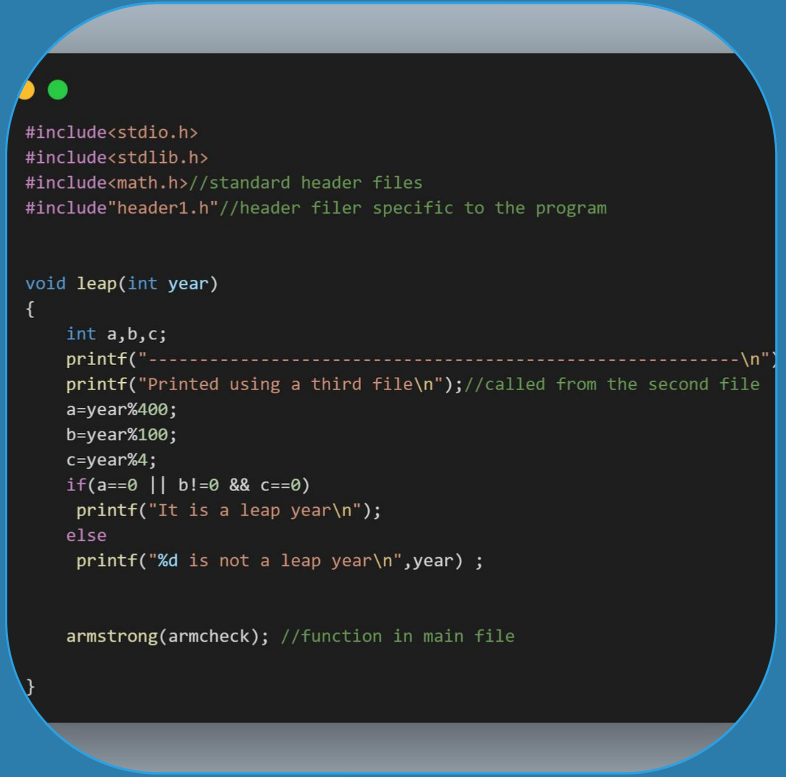
```c
#include<stdio.h>
#include<stdlib.h>
#include<math.h>//standard header files
#include"header1.h"//header filer specific to the program

int main(void)
{

  tables(tableof);//calling a fucntion in second file

}

void armstrong(int n)
{
  printf("----------------------------------------------------------\n");
  printf("present in main file printed using call from prime file\n");
  int arm,r;
  int n1=n;
   //checking if the number is an armstrong number
   while(n!=0)
   {
       r=n%10;
       arm =arm+(r*r*r);
       n/=10;
   }

  if(arm==n1)
  {
      printf("The number %d is an armstrong number!",n1);
  }
  else
  {
      printf("The number %d is not an armstrong number.",n1);
  }
}
```

The main file



```c
<stdio.h>
<stdlib.h>
<math.h>//standard header files
"header1.h"//header filer specific to the program

les(int a)

tf("-----------------------------------------------------
tf("Is printed using a second file\n");
(int i=1;i<=tabletimes;i++)

printf("%d X %d = %d\n",a,i,a*i);

(primecheck);//defined in header file
```

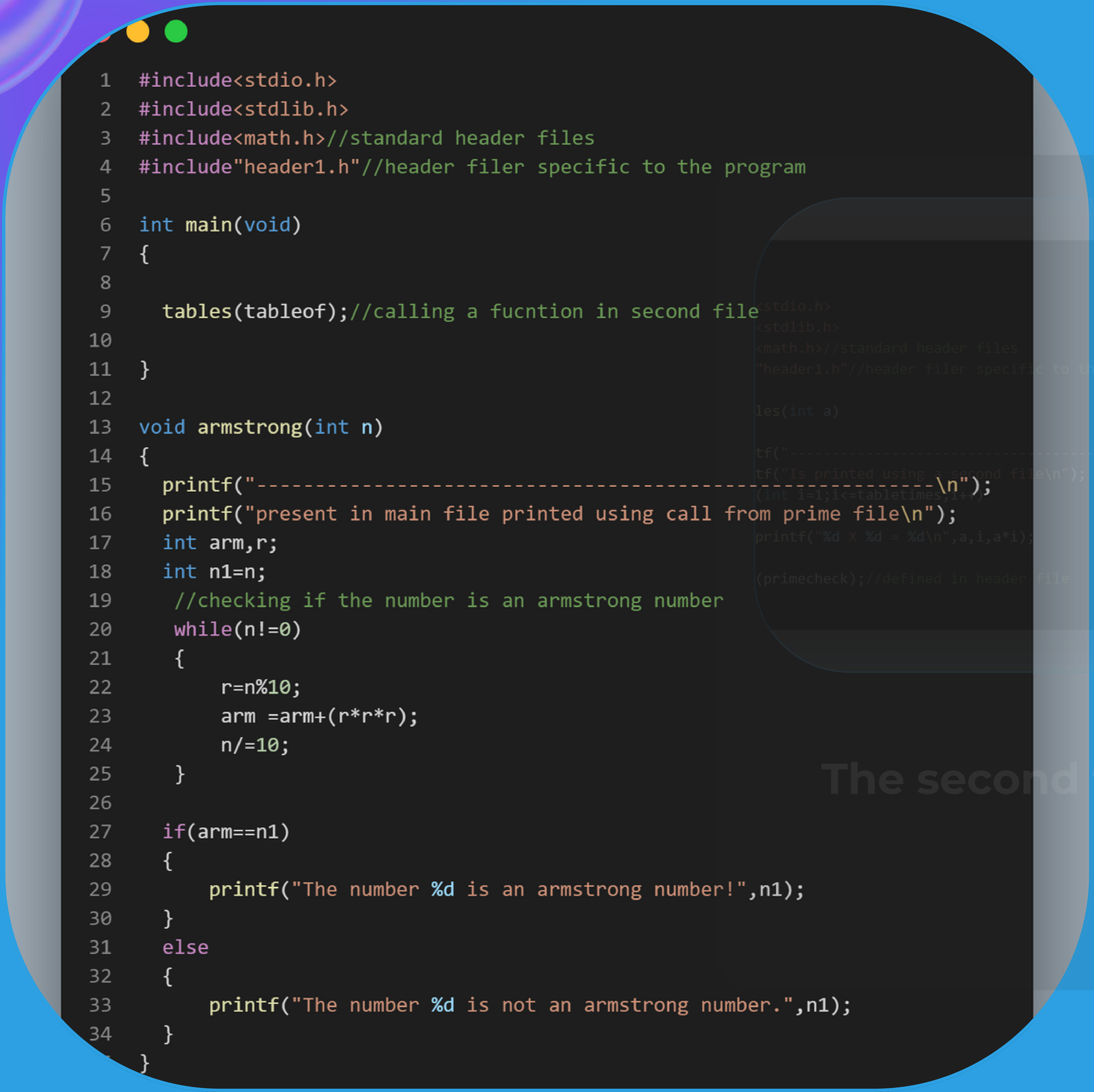
The second file



```c
#include<stdio.h>
#include<stdlib.h>
#include<math.h>//standard header files
#include"header1.h"//header filer specific to the program


void leap(int year)
{
    int a,b,c;
    printf("---------------------------------------------------------\n")
    printf("Printed using a third file\n");//called from the second file
    a=year%400;
    b=year%100;
    c=year%4;
    if(a==0 || b!=0 && c==0)
     printf("It is a leap year\n");
    else
     printf("%d is not a leap year\n",year) ;


    armstrong(armcheck); //function in main file

}
```

The third file

THE FILES

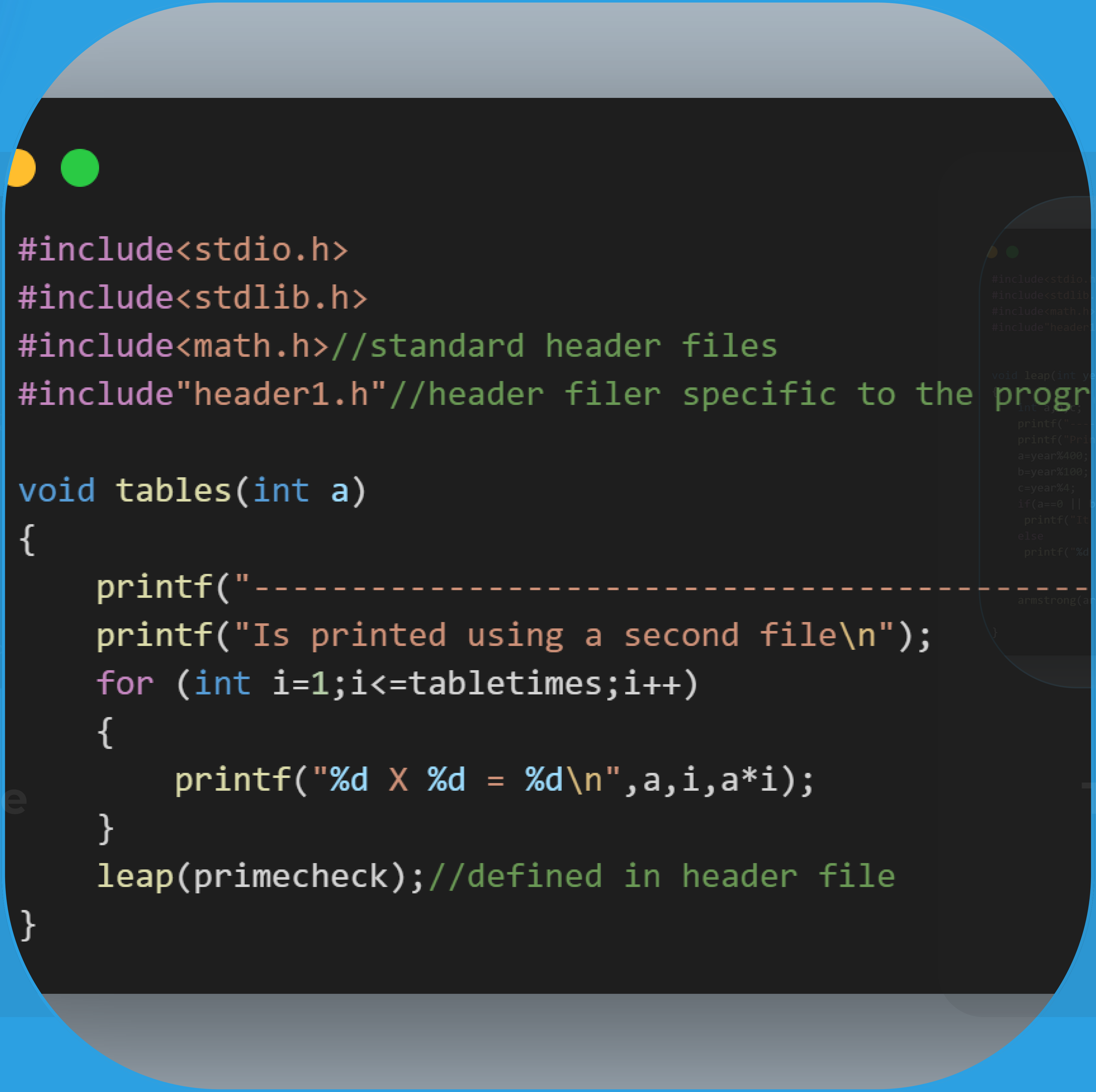


```c
#include<stdio.h>
#include<stdlib.h>
#include<math.h>//standard header files
#include"header1.h"//header filer specific to the program

int main(void)
{

  tables(tableof);//calling a fucntion in second file

}

void armstrong(int n)
{
  printf("----------------------------------------------------------\n");
  printf("present in main file printed using call from prime file\n");
  int arm,r;
  int n1=n;
   //checking if the number is an armstrong number
   while(n!=0)
   {
       r=n%10;
       arm =arm+(r*r*r);
       n/=10;
   }

  if(arm==n1)
  {
      printf("The number %d is an armstrong number!",n1);
  }
  else
  {
      printf("The number %d is not an armstrong number.",n1);
  }
}
```

This file contains the main function. It also contains another function called "armstrong".This is later called by the 3rd file. The main function is calling a function in the second file called tables and passing a constant value to it.

THE FILES

This is the second file called tables. After recieving a call from file 1, it prints the table of the number passed.

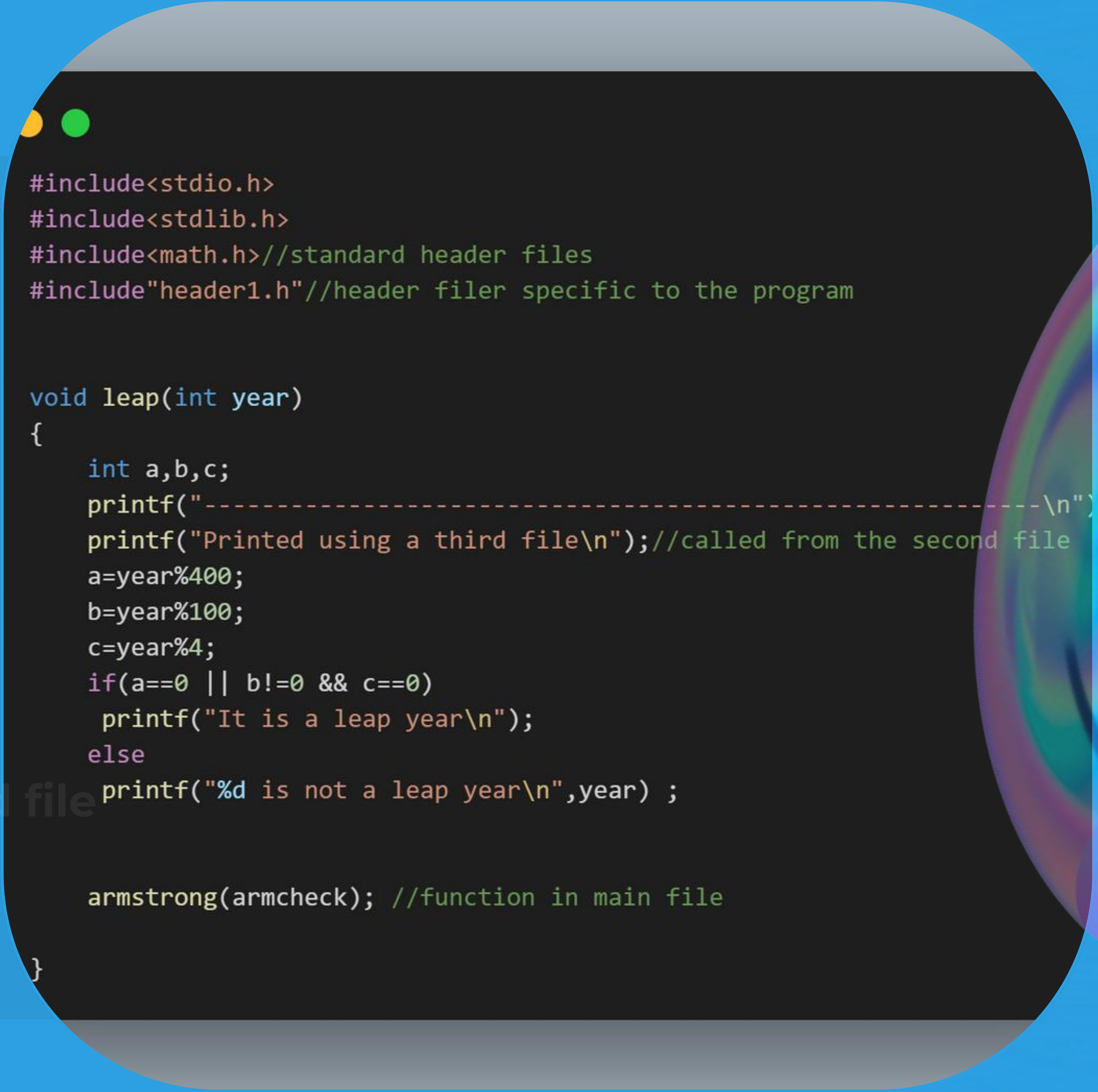


```
#include<stdio.h>
#include<stdlib.h>
#include<math.h>//standard header files
#include"header1.h"//header filer specific to the progr

void tables(int a)
{
    printf("-------------------------------------------
    printf("Is printed using a second file\n");
    for (int i=1;i<=tabletimes;i++)
    {
        printf("%d X %d = %d\n",a,i,a*i);
    }
    leap(primecheck);//defined in header file
}
```

Another thing happening in this file, is that it is calling a function called leap from the 3rd file.

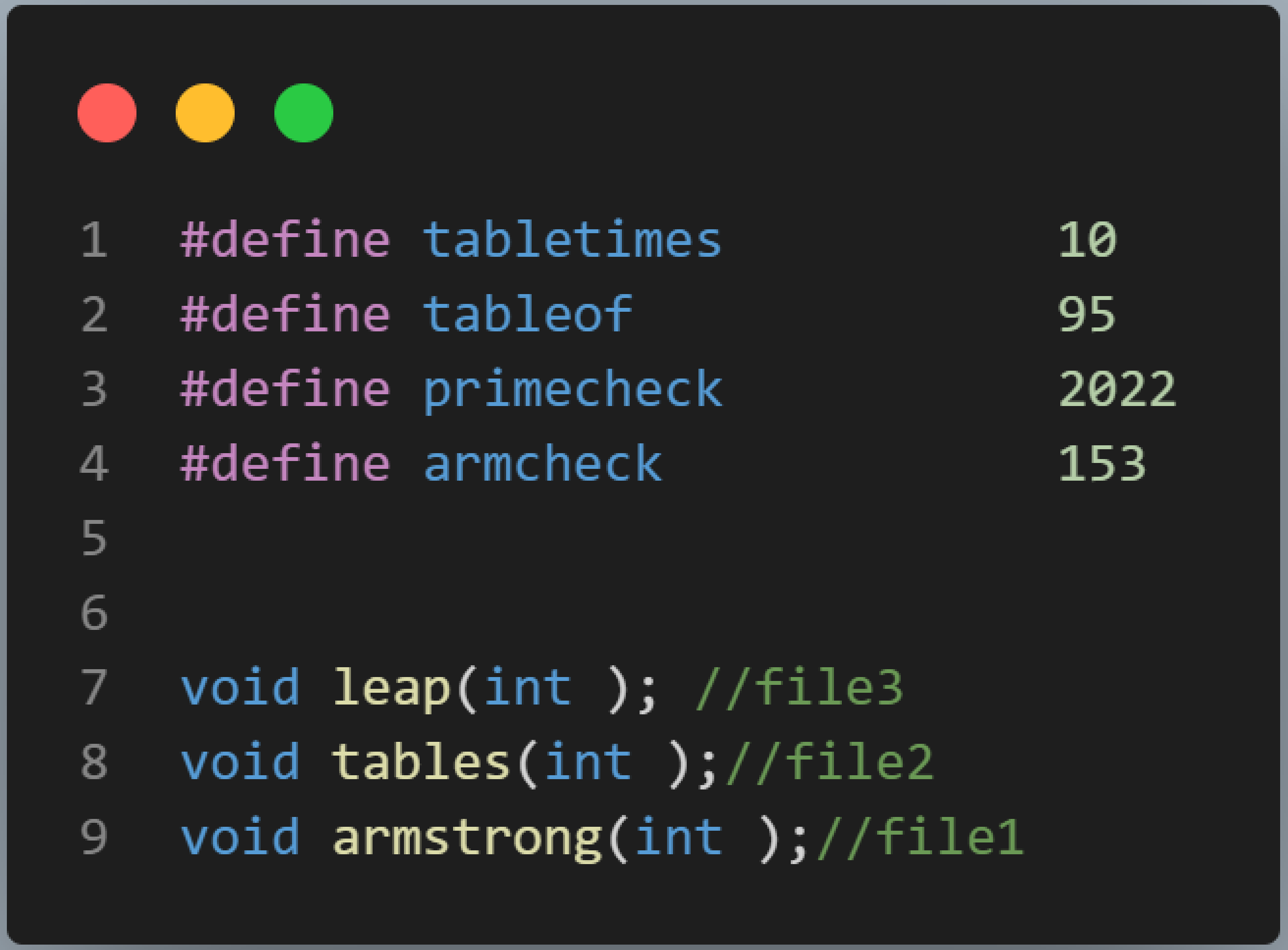
THE FILES

This is the 3rd file in which the year passed by the second file is being checked and the result is being printed.
Not only that, but it is also calling a function from file 1 and passing a number to check if it is armstrong or not. This is executed by the 1st file.



```c
#include<stdio.h>
#include<stdlib.h>
#include<math.h>//standard header files
#include"header1.h"//header filer specific to the program


void leap(int year)
{
    int a,b,c;
    printf("---------------------------------------------------------\n")
    printf("Printed using a third file\n");//called from the second file
    a=year%400;
    b=year%100;
    c=year%4;
    if(a==0 || b!=0 && c==0)
     printf("It is a leap year\n");
    else
     printf("%d is not a leap year\n",year) ;


    armstrong(armcheck); //function in main file

}
```

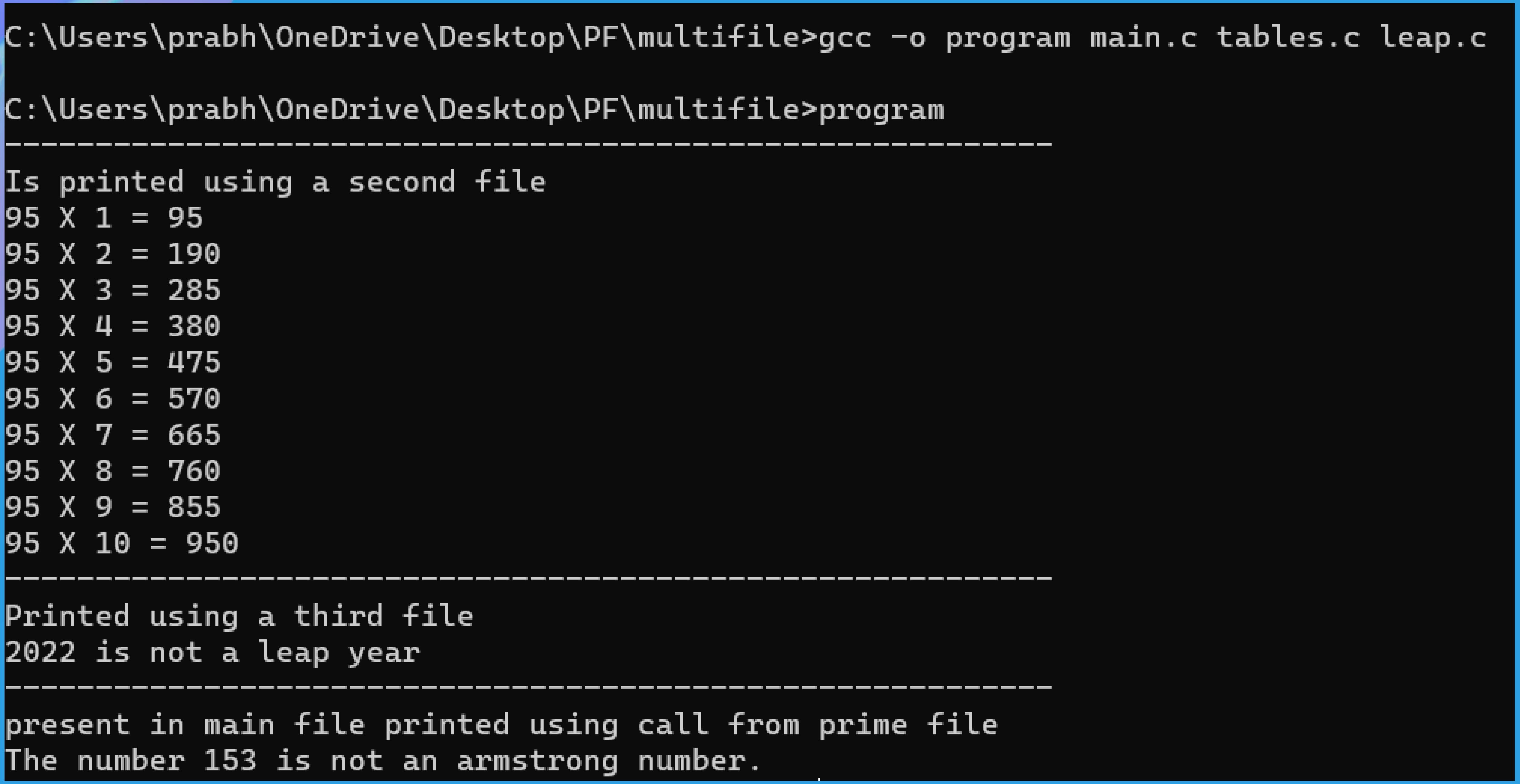
THE HEADER FILE



```
#define tabletimes          10
#define tableof             95
#define primecheck          2022
#define armcheck            153


void leap(int ); //file3
void tables(int );//file2
void armstrong(int );//file1
```

The header file used in our example code
It contains a few parameters already declared which can be used in any file and at any time.

THE OUTPUT



```
C:\Users\prabh\OneDrive\Desktop\PF\multifile>gcc -o program main.c tables.c leap.c

C:\Users\prabh\OneDrive\Desktop\PF\multifile>program
----------------------------------------------------------
Is printed using a second file
95 X 1 = 95
95 X 2 = 190
95 X 3 = 285
95 X 4 = 380
95 X 5 = 475
95 X 6 = 570
95 X 7 = 665
95 X 8 = 760
95 X 9 = 855
95 X 10 = 950
----------------------------------------------------------
Printed using a third file
2022 is not a leap year
----------------------------------------------------------
present in main file printed using call from prime file
The number 153 is not an armstrong number.
```

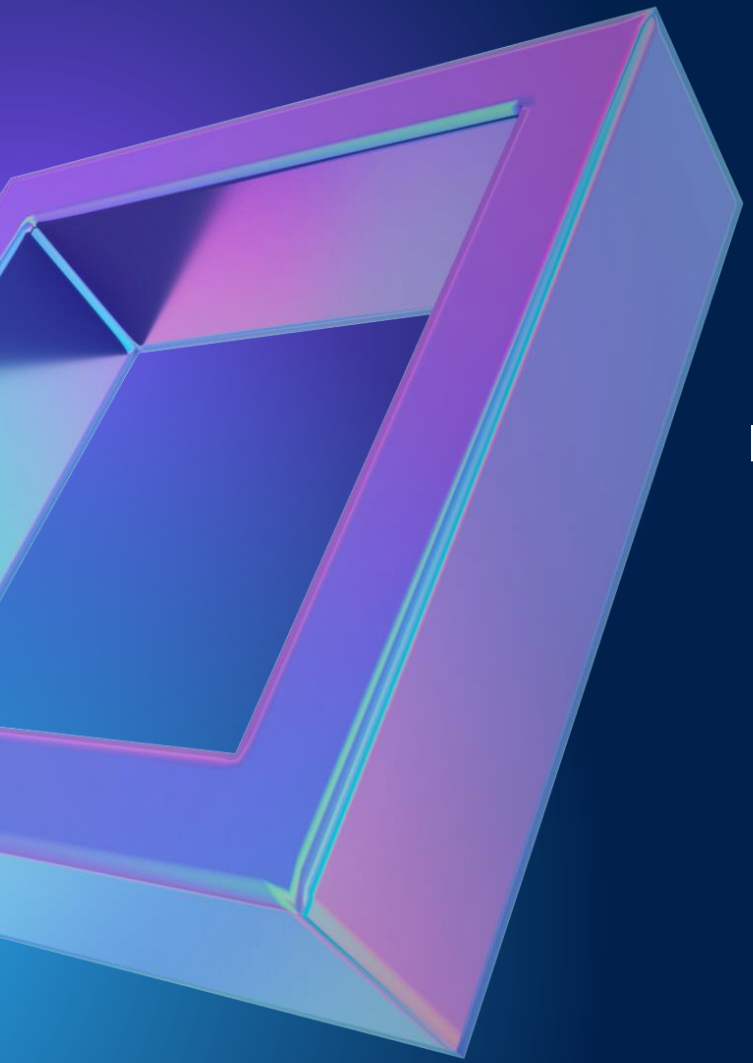
COMPILING

SECOND FILE

THIRD FILE

FIRST FILE

THIS SHOWS HOW A MULTI-FILE C PROGRAM CAN BE RUN BY LINKING ALL THE FILES TOGETHER



Thank You

CODE
REPOSITORY