**Programming for problem solving lab project**

**ES-CS-291**

****

**CALENDER**

**Name of the students including class roll, section and university roll**

FARHAN HAI KHAN I-20

HADEEQA NISHAT I-21

SHREYA SHREE I-51

SHWETA BHARTI I-53

TANNISTHA PAL I-65

ADARSH KUMAR BARNWAL. I-03

**Abstract**

This project is a venture to understand the basic concepts of the calendar and to understand its implemenatation in the C programming language.The program is lucid to understand and involves simple functions with arguments.

Using the Zeller’s formula in our functions, we are now able to not only predict the day that a particular date is,but are also able to print the whole calendar year as an output. In fact , by using the this formula, we can predict the day for any given date.User friendly code with self explanatory comments for all help everyone to align with the logic used in this code and therefore points to the ease of usage.

In this mini project, you can find out the dates corresponding to a given year and view them. The source code is not that long,just 130 lines. It is compiled in Code::Blocks IDE with GCC compiler.

This in addition to separated modules for each process involved to get the modern day calendar,enhances the readability of the code and helps us enjoy learning to code with fun!

**Introduction and History**

All of us use the calendar to predict patterns of time and seasons. Since times immemorial,the calendar affects how man planted, harvested, hunted, and performed many other of the tasks necessary to their very survival. Using a calendar assists us with creating routine in our day & hence is a an integrated and inseparable part of the human life.

However the calendar in the current form has been invented after various different versions came up.The simple Julian calendar came up, but caused lot of confusion for change of seasons, then the modern day calendar was brought which solved these complications.

It takes the earth about 365.2422 days to go around the sun, but a normal calendar year is only 365 days. The extra fraction of a day adds up: circling the sun four times takes 1460.9688 days, but four calendar years would only be 1460 days. That .9688 is almost a whole day, so every four years we add an extra day to our calendar, February 29. We call that year leap year. To make things easier, leap years are always divisible by four: 2004 and 2008 will both be leap years.

For hundreds of years, people used a calendar called the Julian calendar that followed this rule, adding a leap year every four years. However, because .9688 isn't *exactly* a whole day, the Julian calendar slowly began to disagree with the real seasons. In 1582, Pope Gregory fixed this problem by ordering everyone to use a new set of rules. These rules are named the Gregorian calendar, after him. They work like this:

**The Gregorian Calendar**

|  |  |
| --- | --- |
| **Rule** | **Examples** |
| Every fourth year is a leap year. | 2040, 2016, and 1996 are leap years. |
| However, every hundredth year is *not* a leap year. | 1700 and 2100 are *not* leap years. |
| Every four hundred years, there's a leap year after all. | 2000 and 2400 are leap years. |

People in English-speaking countries didn't start using the Gregorian calendar until 1752. Some countries, such as Iran, still use other systems.

**The Working Principle : Zeller's Rule**

The following formula is named Zeller's Rule after a Reverend Zeller. [x] means the greatest integer that is smaller than or equal to x. You can find this number by just dropping everything after the decimal point. For example, [3.79] is 3. Here's the formula:

f = k + [(13\*m-1)/5] + D + [D/4] + [C/4] - 2\*C.

* k is the day of the month. Let's use January 29, 2064 as an example. For this date, k = 29.
* m is the month number. Months have to be counted specially for Zeller's Rule: March is 1, April is 2, and so on to February, which is 12. (This makes the formula simpler, because on leap years February 29 is counted as the last day of the year.) Because of this rule, January and February are always counted as the 11th and 12th months *of the previous year*. In our example, m = 11.
* D is the last two digits of the year. Because in our example we are using January (see previous bullet) D = 63 even though we are using a date from 2064.
* C stands for century: it's the first two digits of the year. In our case, C = 20.

Now let's substitute our example numbers into the formula.

f = k + [(13\*m-1)/5] + D + [D/4] + [C/4] - 2\*C  
= 29 + [(13\*11-1)/5] + 63 + [63/4] + [20/4] - 2\*20  
= 29 + [28.4] + 63 + [15.75] + [5] - 40  
= 29 + 28 + 63 + 15 + 5 - 40  
= 100.

Once we have found f, we divide it by 7 and take the remainder. Note that if the result for f is negative, care must be taken in calculating the proper remainder. Suppose f = -17. When we divide by 7, we have to follow the same rules as for the greatest integer function; namely we find the greatest multiple of 7 *less* than -17, so the remainder will be positive (or zero). -21 is the greatest multiple of 7 less than -17, so the remainder is 4 since -21 + 4 = -17. Alternatively, we can say that -7 goes into -17 twice, making -14 and leaving a remainder of -3, then add 7 since the remainder is negative, so -3 + 7 is again a remainder of 4.

A remainder of 0 corresponds to Sunday, 1 means Monday, etc. For our example, 100 / 7 = 14, remainder 2, so January 29, 2064 will be a Tuesday.

**Basic Algorithm(Flow Chart) with Logical Understanding of Code**

**The Source Code**

#include<stdio.h>

//global variables

int days\_in\_month[]={0,31,28,31,30,31,30,31,31,30,31,30,31};

/\*first value is initialised as 0 as we do not want to keep confusion

between index of array and the corresponding month

to put it simply, month 0 does not exist so we put an arbitrary value to it.\*/

char \*months[]= { " ", "\n\n\nJanuary", "\n\n\nFebruary", "\n\n\nMarch", "\n\n\nApril", "\n\n\nMay","\n\n\nJune",

"\n\n\nJuly", "\n\n\nAugust", "\n\n\nSeptember", "\n\n\nOctober", "\n\n\nNovember", "\n\n\nDecember" };

//2D array created using ptr to 1D array

//function to get input year

int inputyear()

{

int year;

printf("Please enter a year (example: 1999) : ");

scanf("%d", &year);

return year;

}

//

int determinedaycode (int y)

{ // function will return sun=0,mon=1,tue=2,wen=3,thu=4,fri=5,sat=6

//y is the year

int f,a,b,c,d,daycode;

//we need to find out the daycode for the 1st of February for a particular month

//Using original Zeller's formula with

y--; /\*since february is considered as 11th month of the previous year

in zeller's function,(since months in zeller formula start from March)

hence we subtract a year to find out its first of february\*/

//use m=11 & k=1

a=(y%100);

b=a/4;

c=(y/100);

d=c/4;

f=29+a+b+d-2\*c;

/\*zeller's formula- f = k + [(13\*m-1)/5] + D + [D/4] + [C/4] - 2\*C

for more info see readme file

[] is called greatest integer function8\*/

/\*here b & d are used as we need the greatest

integer function value for (y%100)/4 and (y/100)/4

respectively.This is done to avoid using the gif function

explicitly over floating pt values.\*/

/\*for a year like 1700 , f becomes -2(negative)

so we have to add 7n to f where n is such

an integer which gives f as a positive value\*/

//the negative case of f

if(f<0)

{ int temp=-f;

/\*we use the theory here that n=[mod(f)/7] +1

for f to be positive and in between 0 & 7

since we are using integer data type here,

we don't actually need to use floating point

numbers and then find out the Greatest Integer function for that\*/

int n=(temp/7)+1;

//corrected f would be

f=f+7\*n;

}

daycode=f%7;

return daycode;

}

/\*below function returns 1 for leap year and 0 for non leap year

suitably changing the days\_in\_month value for february as 28

for non leap and 29 for leap\*/

int determineleapyear(int year)

{

if(year% 4 == 0 && year%100 != 0 || year%400 == 0)

{

days\_in\_month[2] = 29;

return 1;

}

else

{

days\_in\_month[2] = 28;

return 0;

}

}

/\*output handling function with suitable decoration\*/

void calendar(int year, int daycode)

{

int month, day;

for ( month = 1; month <= 12; month++ )

{

printf("%s", months[month]);

printf("\n\nSun Mon Tue Wed Thu Fri Sat\n" );

// Correct the position for the first date

for ( day = 1; day <= 1 + daycode \* 5; day++ )

{

printf(" "); //space for correctly aligning the days

}

// Print all the dates for one month

for ( day = 1; day <= days\_in\_month[month]; day++ )

{

printf("%2d", day ); //to format the output to display only 2 digits

// Is day before Sat? Else start next line Sun.

if ( ( day + daycode ) % 7 > 0 )

printf(" " );

else

printf("\n " );

}

// Setting the position for next month

daycode = ( daycode + days\_in\_month[month] ) % 7;

}

}

int main()

{

int year, daycode, leapyear;

year = inputyear(); //takes in the input for a year

daycode = determinedaycode(year);

//daycode holds the following essential values sun=0,mon=1,tue=2,wen=3,thu=4,fri=5,sat=6

determineleapyear(year);

// returns one for leap and 0 for non leap year . Modifies the month February accordingly

calendar(year, daycode);

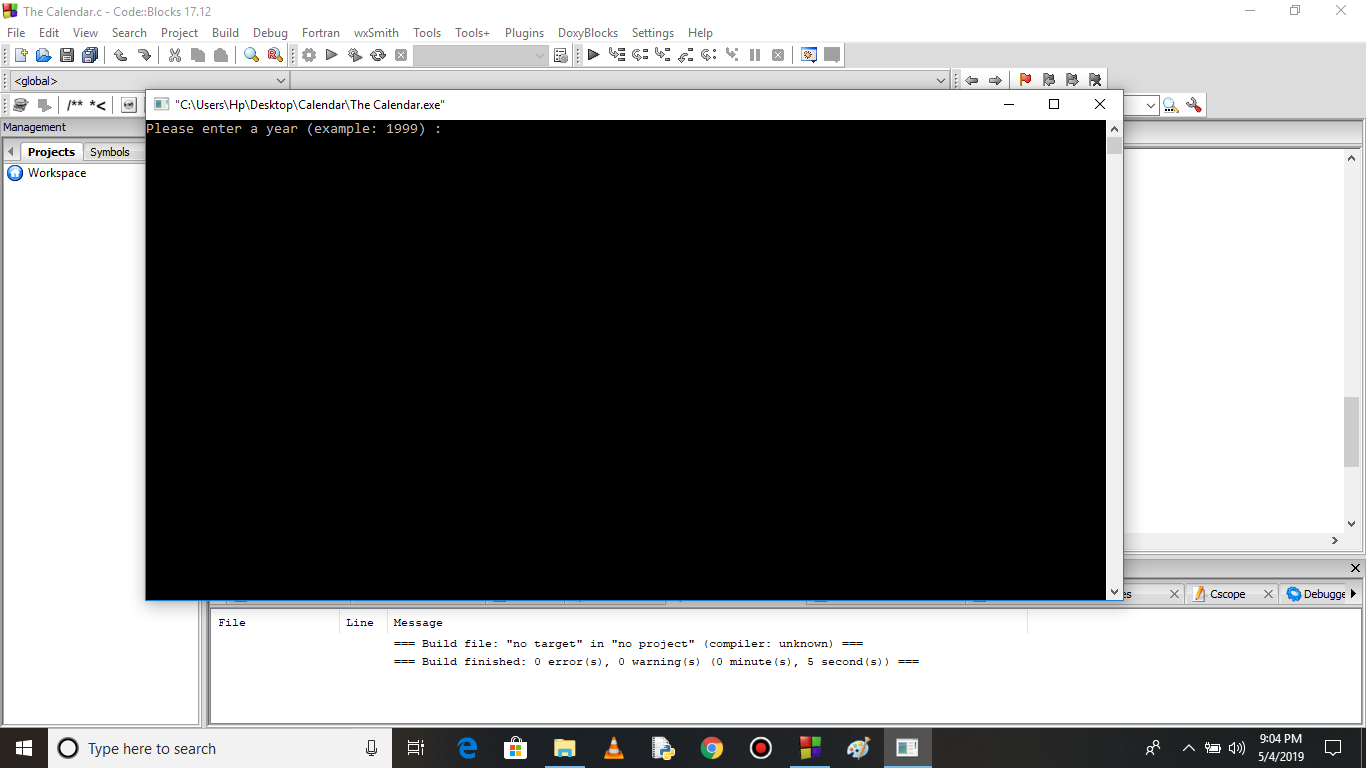
// the code responsible for handling the output part

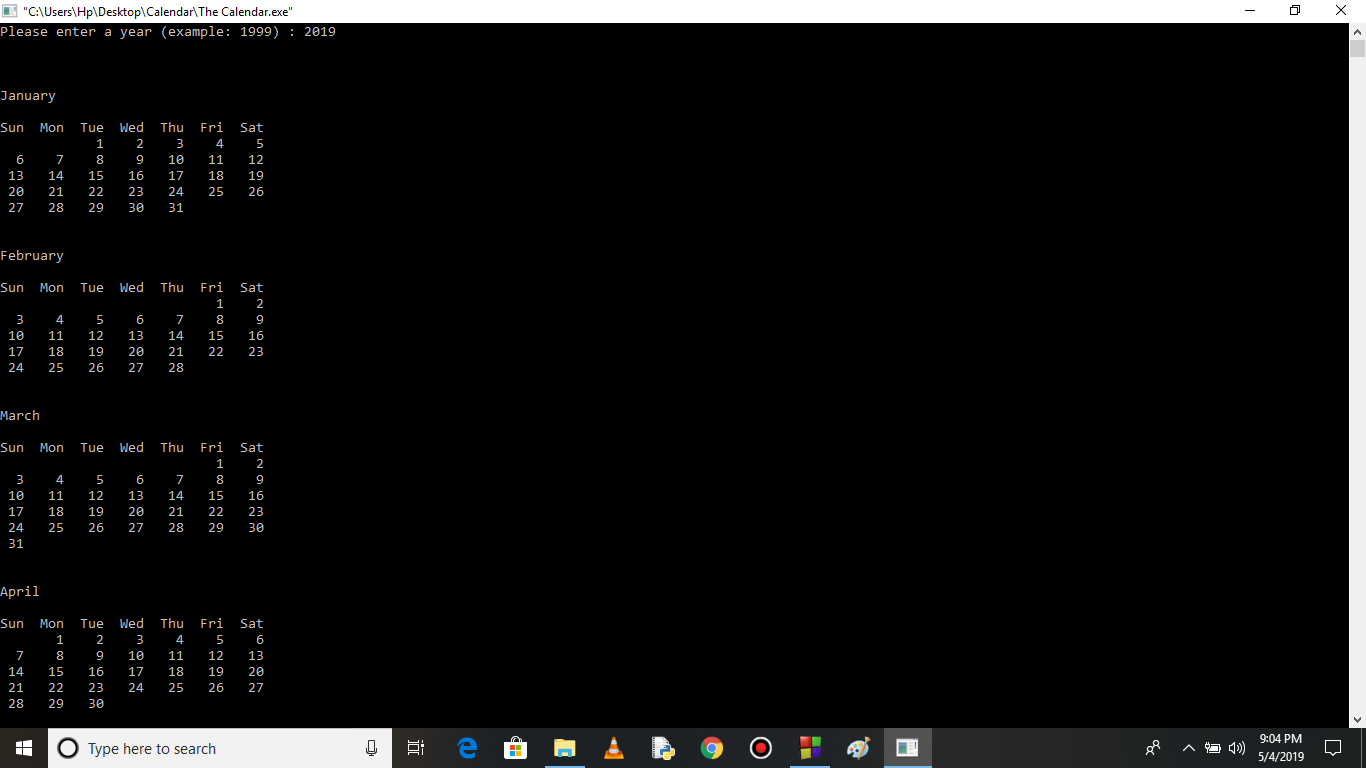
printf("\n");

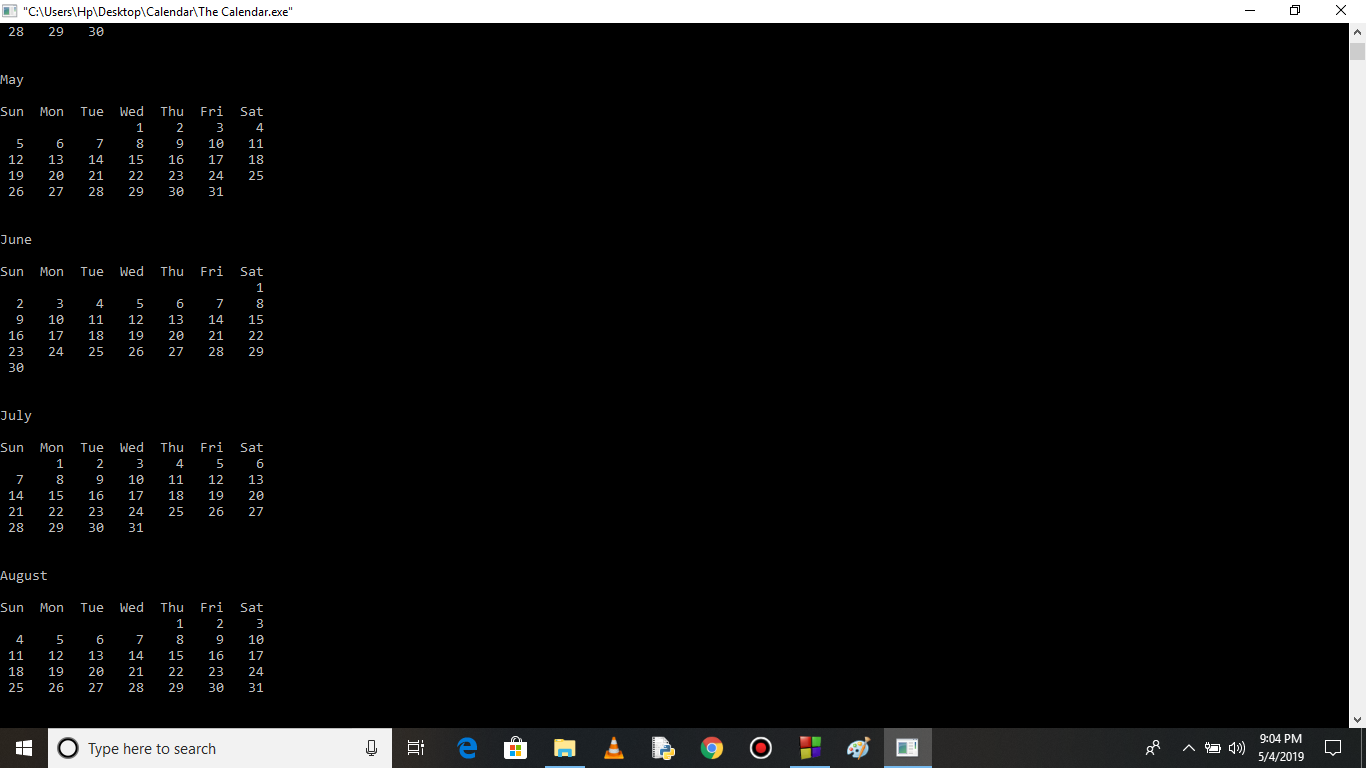
}

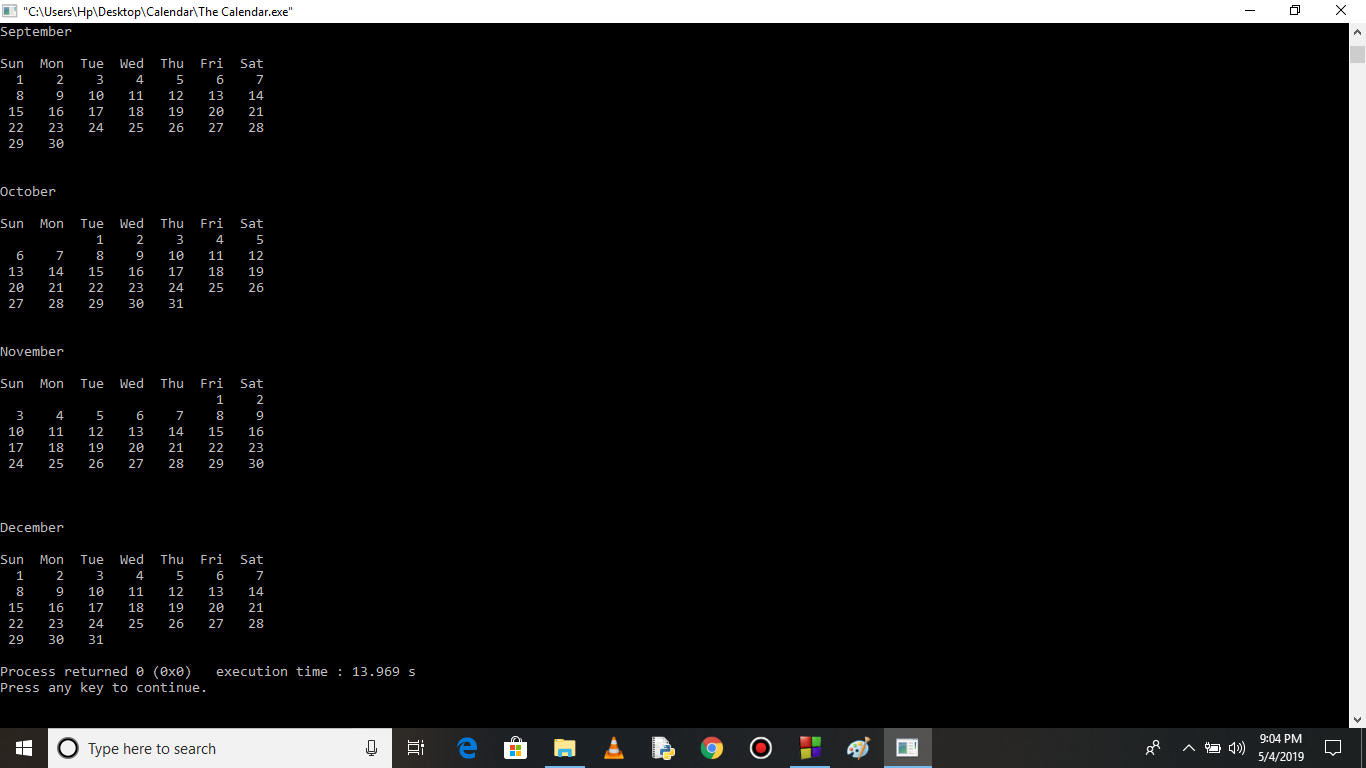
**Output**

Taking input year as 2019!









**References &Acknowledgement**

I would like to express my special thanks of gratitude to all my teachers, my mentors, supporting staff & also the sources I referred to while understanding the calendar. In particular I would like to thank my dean sir, HOD sir,for believing in us and inspiring us throughout this journey. And also I would like to thank all those who have indirectly guided and helped me in preparation of this project.

I have used various citations from <http://mathforum.org/dr.math/faq/faq.calendar.html>

for understanding minute elements of the zeller’s formula. Also I used the CodeBlocks IDE to compile & code.The screenshots enclosed on this project are all taken from the same.