Student Name: Ajay Sahani
Student ID: 0801CS211010

Mini project

Submitted To: Sir Surendra Gupta

---

1. **Acknowledgment**

   I would like to express my special thanks of gratitude to my Sir Surendra Gupta who gives me the golden opportunity to do this wonderful project CALENDAR, which also help me a lot,I am really thankful to him.

2. **objective**

   This main objective of the calendar is to identify days. This to be informed about or to agree on a future events and to record an events that has happened .Days may be significant for agriculture ,civil,religious or social reasons.

3. **Function Description**
   The first function inputyear() is used to get the user input. The second function determinedaycode() is used to get the day number of the first day in that year ,so we can print the date on the correct position. The function determineleapyear() is used to determine if input of the user is a leap year. If so, the number if days in February is changed to 29. The last function calendar() is used to print each month onto the screen. The first for loop is used to loop through all months. We then print the month's name and all the days of the week.

4. **overview**
   This main objective of the calendar is to identify days. This to be informed about or to agree on a future events and to record an events that has happened .Days may be significant for agriculture ,civil,religious or social reasons. events that has happened .Days may be significant for agriculture ,civil,religious or social reason.

5. **code in c**

```
    #include<stdio.h>

#define TRUE    1
#define FALSE   0

int days_in_month[]={0,31,28,31,30,31,30,31,31,30,31,30,31};
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"
};
int inputyear(void)
{
    //system("color 3F");
int year;

printf("Enter the desired year : ");
scanf("%d", &year);
return year;
}

int determinedaycode(int year)
{
int daycode;
int d1, d2, d3;

d1 = (year - 1.)/ 4.0;
d2 = (year - 1.)/ 100.;
d3 = (year - 1.)/ 400.;
daycode = (year + d1 - d2 + d3) %7;
return daycode;
}


int determineleapyear(int year)
{
if(year % 4 == 0 || year%100 != 0 || year%400 == 0)
{
days_in_month[2] = 29;
return 0;
}
else
{
days_in_month[2] = 28;
return -1;
}
}

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" );

for ( day = 1; day <= 1 + daycode * 5; day++ )
{
printf(" ");
}


for ( day = 1; day <= days_in_month[month]; day++ )
{
printf("%2d", day );

if ( ( day + daycode ) % 7 > 0 )
printf("   " );
else
printf("\n " );
}

daycode = ( daycode + days_in_month[month] ) % 7;
}
}

int main(void)
{
int year, daycode;

year = inputyear();
daycode = determinedaycode(year);
determineleapyear(year);
calendar(year, daycode);
printf("\n");
}
```

6. **Profiling**

```
    Flat profile:

Each sample counts as 0.01 seconds.
 no time accumulated

  %   cumulative   self              self     total
 time   seconds   seconds    calls  Ts/call  Ts/call  name
  0.00      0.00     0.00        1     0.00     0.00  calendar
  0.00      0.00     0.00        1     0.00     0.00  determinedaycode
  0.00      0.00     0.00        1     0.00     0.00  determineleapyear
  0.00      0.00     0.00        1     0.00     0.00  inputyear
```

```
 %         the percentage of the total running time of the
time       program used by this function.

cumulative a running sum of the number of seconds accounted
 seconds   for by this function and those listed above it.

 self      the number of seconds accounted for by this
seconds    function alone.  This is the major sort for this
           listing.

calls      the number of times this function was invoked, if
           this function is profiled, else blank.

 self      the average number of milliseconds spent in this
ms/call    function per call, if this function is profiled,
   else blank.

 total     the average number of milliseconds spent in this
ms/call    function and its descendents per call, if this
   function is profiled, else blank.

name       the name of the function.  This is the minor sort
           for this listing. The index shows the location of
   the function in the gprof listing. If the index is
   in parenthesis it shows where it would appear in
   the gprof listing if it were to be printed.

Copyright (C) 2012-2017 Free Software Foundation, Inc.

Copying and distribution of this file, with or without modification,
are permitted in any medium without royalty provided the copyright
notice and this notice are preserved.

     Call graph (explanation follows)


granularity: each sample hit covers 4 byte(s) no time propagated

index % time    self  children    called     name
                0.00    0.00       1/1           main [81]
[2]      0.0    0.00    0.00       1         calendar [2]
-----------------------------------------------
                0.00    0.00       1/1           main [81]
[3]      0.0    0.00    0.00       1         determinedaycode [3]
-----------------------------------------------
                0.00    0.00       1/1           main [81]
[4]      0.0    0.00    0.00       1         determineleapyear [4]
-----------------------------------------------
                0.00    0.00       1/1           main [81]
[5]      0.0    0.00    0.00       1         inputyear [5]
```

```
-----------------------------------------------

 This table describes the call tree of the program, and was sorted by
 the total amount of time spent in each function and its children.

 Each entry in this table consists of several lines.  The line with the
 index number at the left hand margin lists the current function.
 The lines above it list the functions that called this function,
 and the lines below it list the functions this one called.
 This line lists:
     index A unique number given to each element of the table.
Index numbers are sorted numerically.
The index number is printed next to every function name so
it is easier to look up where the function is in the table.

     % time This is the percentage of the `total' time that was spent
in this function and its children.  Note that due to
different viewpoints, functions excluded by options, etc,
these numbers will NOT add up to 100%.

     self This is the total amount of time spent in this function.

     children This is the total amount of time propagated into this
function by its children.

     called This is the number of times the function was called.
If the function called itself recursively, the number
only includes non-recursive calls, and is followed by
a `+' and the number of recursive calls.

     name The name of the current function.  The index number is
printed after it.  If the function is a member of a
cycle, the cycle number is printed between the
function's name and the index number.


 For the function's parents, the fields have the following meanings:

     self This is the amount of time that was propagated directly
from the function into this parent.

     children This is the amount of time that was propagated from
the function's children into this parent.

     called This is the number of times this parent called the
function `/' the total number of times the function
was called.  Recursive calls to the function are not
included in the number after the `/'.

     name This is the name of the parent.  The parent's index
```

```
number is printed after it.  If the parent is a
member of a cycle, the cycle number is printed between
the name and the index number.

 If the parents of the function cannot be determined, the word
 '<spontaneous>' is printed in the 'name' field, and all the other
 fields are blank.

 For the function's children, the fields have the following meanings:

     self This is the amount of time that was propagated directly
from the child into the function.

     children This is the amount of time that was propagated from the
child's children to the function.

     called This is the number of times the function called
this child '/' the total number of times the child
was called.  Recursive calls by the child are not
listed in the number after the '/'.

     name This is the name of the child.  The child's index
number is printed after it.  If the child is a
member of a cycle, the cycle number is printed
between the name and the index number.

 If there are any cycles (circles) in the call graph, there is an
 entry for the cycle-as-a-whole.  This entry shows who called the
 cycle (as parents) and the members of the cycle (as children.)
 The '+' recursive calls entry shows the number of function calls that
 were internal to the cycle, and the calls entry for each member shows,
 for that member, how many times it was called from other members of
 the cycle.

Copyright (C) 2012-2017 Free Software Foundation, Inc.

Copying and distribution of this file, with or without modification,
are permitted in any medium without royalty provided the copyright
notice and this notice are preserved.

Index by function name

   [2] calendar                [4] determineleapyear
   [3] determinedaycode        [5] inputyear
```

7. **Debugging**

8. **Output**

Figure 1: fig.0

Figure 2: fig.1

Figure 3: fig.2

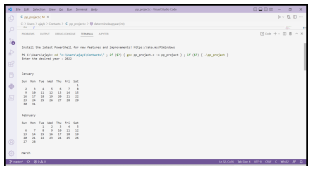

Figure 4: fig output.1

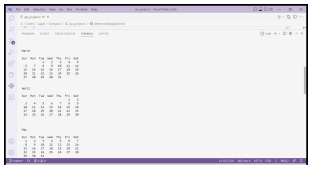

Figure 5: fig output.2

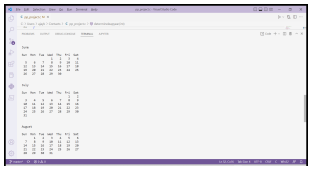

Figure 6: fig output.3

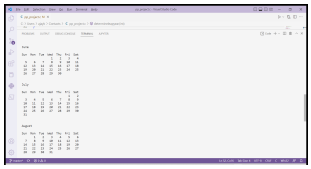

Figure 7: fig output.4

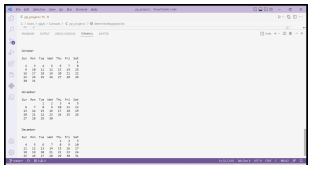

Figure 8: fig output.5