Programming with C

Terry Marris December 2010

20 Date and Time

Previously we looked at files. Now we take a look at date and time.

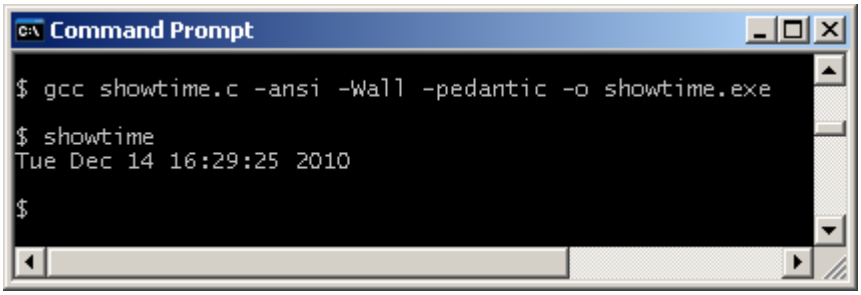
20.1 Calendar Time

Calendar time is the number of seconds since midnight on 1 January 1970. This number represents the current date and time according to the Gregorian calendar. The Gregorian calendar is the one we normally use in the UK, Australia, USA, Europe, ... Other calendars include the Hebrew, Hindu and Chinese calendars.

Calendar time is held within *time_t.* And *time_t* is defined in the standard library *time.h*.

Also defined in *time.h* are *time()* and *ctime()*. *time()* returns the current calendar time from the computer system - if it can. If it cannot, *time()* returns -1. *ctime()* converts calendar time into readable text.

```
/* showtime.c: shows current calendar time */

#include <stdio.h>
#include <time.h>

int main()
{
  time_t now;

  if (-1 == time(&now))
    printf("Time not available\n");
  else
    printf(ctime(&now));
  return 0;
}
```



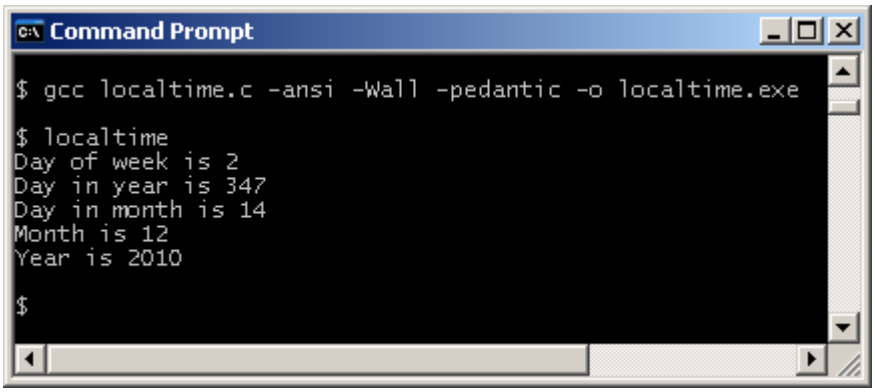
20.2 Locale Time

Local time is the time in years, months, days, minutes, seconds, ... Local time is held in a C structure named *tm*. tm is defined in *time.h*.

```
struct tm
{
  int tm_sec;   /* seconds after the minute: normally 0..59 */
  int tm_min;   /* minutes after the hour: 0..59 */
  int tm_hour;  /* hours since midnight: 0..23 */
  int tm_mday;  /* day of the month: 1..31 */
  int tm_mon;   /* months since January: 0..11 */
  int tm_year;  /* years since 1900 */
  int tm_wday;  /* days since Sunday: 0..6 */
  int tm_yday;  /* days since 1st January: 0..365 */
  int tm_isdst; /* Daylight Saving Time: > 0 if DST is in effect, 0 if not,
                   and < 0 if no information */
};
```

tm can get its values from the calendar time, or from the user.

```
/* localtime.c: gets local time from calendar time */

#include <stdio.h>
#include <time.h>

int main()
{
  time_t calTime;
  struct tm *locTime;

  time(&calTime);
  locTime = localtime(&calTime);

  printf("Day of week is %d\n", locTime->tm_wday);
  printf("Day in year is %d\n", locTime->tm_yday);
  printf("Day in month is %d\n", locTime->tm_mday);
  printf("Month is %d\n", (locTime->tm_mon + 1));
  printf("Year is %d\n", (locTime->tm_year + 1900));
  return 0;
}
```



The statement

```
time(&calTime);
```

gets the calendar time from the computer system and

```
locTime = localtime(&calTime);
```

initialises the members of local time with the calendar time.

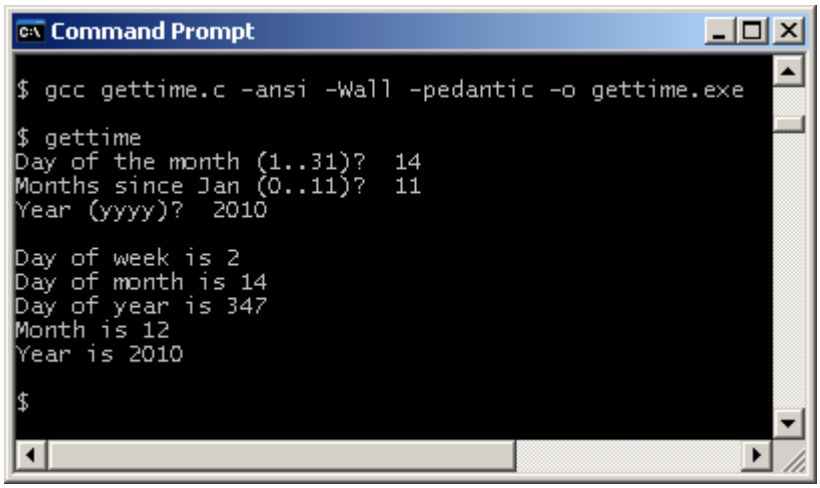
The statements

```
printf("Day of week is %d\n", locTime->tm_wday);
printf("Day in year is %d\n", locTime->tm_yday);
printf("Day in month is %d\n", locTime->tm_mday);
printf("Month is %d\n", (locTime->tm_mon + 1));
printf("Year is %d\n", (locTime->tm_year + 1900));
```

display various members of the *tm* structure. We are obliged to add 1 to the month since months stored are months since January, i.e. months stored are in 0..11. Similarly, we have to add 1900 to the year stored.

Now we see how the user can initialise members of the *tm* structure.

```
/* gettime.c: gets the date and time from the user */

#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int getInt(char *prompt)
{
  char string[BUFSIZ];
  printf("%s ", prompt);
  gets(string);
  return atoi(string);
}

int main()
{
  struct tm t;

  t.tm_mday = getInt("Day of the month (1..31)? ");
  t.tm_mon = getInt("Months since Jan (0..11)? ");
  t.tm_year = getInt("Year (yyyy)? ") - 1900;
  t.tm_hour = t.tm_min = t.tm_sec = 1;
  t.tm_isdst = -1;
  if (-1 == mktime(&t)) {
    printf("Invalid date\n");
    exit(EXIT_FAILURE);
  }
  printf("\nDay of week is %d\n", t.tm_wday);
  printf("Day of month is %d\n", t.tm_mday);
  printf("Day of year is %d\n", t.tm_yday);
  printf("Month is %d\n", t.tm_mon + 1);
  printf("Year is %d\n", t.tm_year + 1900);
  return 0;
}
```



We declare a variable, *t*, of type *tm struct*, then initialise members.

```
t.tm_mday = getInt("Day of the month (1..31)? ");
t.tm_mon = getInt("Months since Jan (0..11)? ");
t.tm_year = getInt("Year (yyyy)? ") - 1900;
t.tm_hour = t.tm_min = t.tm_sec = 1;
t.tm_isdst = -1;
```

Notice that we subtract 1900 from the year stored, that hours, minutes and seconds are set to 1, and that daylight saving time is set to unknown.

Then the structure is used as an argument to *mktime()*. *mktime()* computes appropriate values for day of the week as well as day of the year. *mktime()* returns the calendar time form the given local time, or -1 if the calendar time cannot be calculated.

20.3 Date Calculations

We see how to calculate the date thirty days from today. For today's date we get the calendar time and use it to initialise the local time.

```
time_t ct;
struct tm *lt;

time(&ct);
lt = localtime(&ct);

printf("Today's date is: %d-%d-%d\n",
    lt->tm_mday, (lt->tm_mon +1), (lt->tm_year + 1900));
```

Then we add 30 days to the *tm_mday* member and ask *mktime()* to recalculate.

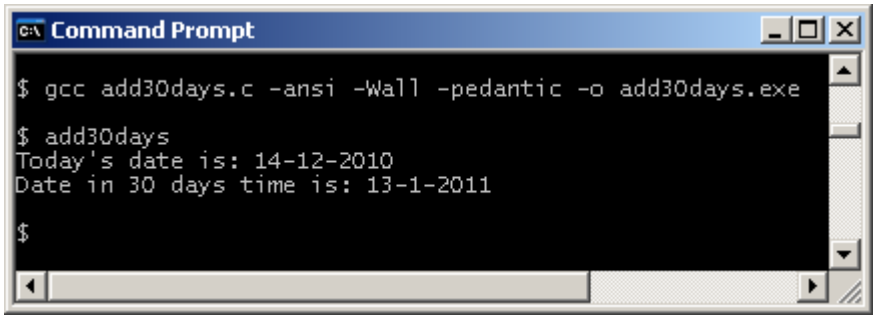
```
lt->tm_mday = lt->tm_mday + 30;
if (-1 == mktime(lt)) {
  printf("Invalid date\n");
  exit(EXIT_FAILURE);
}
```

Finally, we print the new date.

```
printf("Date in 30 days time is: %d-%d-%d\n",
lt->tm_mday, (lt->tm_mon +1), (lt->tm_year + 1900));
```

Here is the entire program and its run.

```
/* add30days.c: adds 30 days to today */

#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int main()
{
  time_t ct;
  struct tm *lt;

  time(&ct);
  lt = localtime(&ct);

  printf("Today's date is: %d-%d-%d\n",
      lt->tm_mday, (lt->tm_mon +1), (lt->tm_year + 1900));

  lt->tm_mday = lt->tm_mday + 30;
  if (-1 == mktime(lt)) {
    printf("Invalid date\n");
    exit(EXIT_FAILURE);
  }

  printf("Date in 30 days time is: %d-%d-%d\n",
      lt->tm_mday, (lt->tm_mon +1), (lt->tm_year + 1900));
  return 0;
}
```



Neat or what?

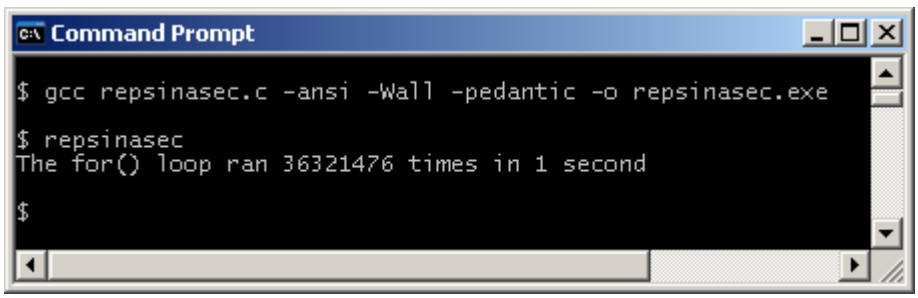
20.4 Timers

We use *difftime()* to determine the difference between two times. *difftime()* is in *time.h*. *type_t* is an arithmetic type used to represent time. It is also defined in *time.h*.

```
/* difftime.c: shows difference between two times */

#include <stdio.h>
#include <time.h>

int main()
{
  time_t t1, t2;

  time(&t1);
  printf("Press enter when you feel like it ... ");
  getchar();
  time(&t2);
  printf("You waited %0.0f seconds\n", difftime(t2, t1));
  return 0;
}
```

The next program counts the number of times a loop executes in about 1 second on my computer system.

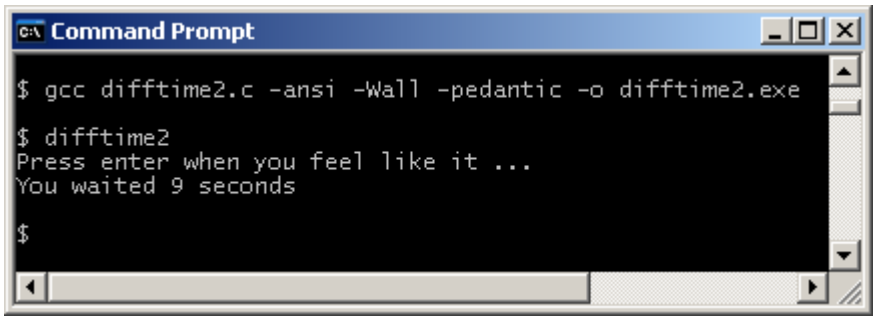
```
/* repsinasec.c: counts the  repetitions in a second */

#include <stdio.h>
#include <time.h>

int main()
{
  clock_t finish;
  int i;

  finish = clock() + CLK_TCK;
  for (i = 0; finish > clock(); i++)
    ;
  printf("The for() loop ran %d times in 1 second\n", i);
  return 0;
}
```



clock_t, defined in *time.h*, is an arithmetic data type that is returned by the *clock()* function. The *clock()* function calculates and returns the amount of processor time taken to reach a particular point in the program. *CLK_TCK* represents the number of ticks in a second, where a tick is the unit of time measured by the *clock()* function.

Incidentally, you could determine the number of seconds to reach a given point by dividing the value returned by c*lock()* by *CLK_TCK*.

```
/* difftime2.c: shows difference between two times */

#include <stdio.h>
#include <time.h>

int main()
{
  clock_t start, stop;

  start = clock();
  printf("Press enter when you feel like it ... ");
  getchar();
  stop = clock();
  printf("You waited %d seconds\n", ((stop - start)/CLK_TCK));
  return 0;
}
```

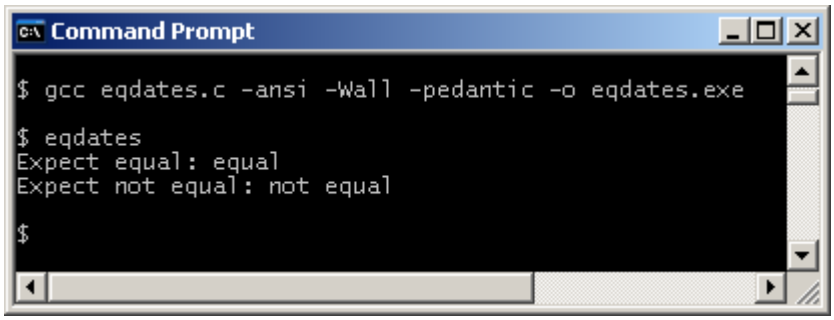


20.5 Comparing Dates

To determine whether two local dates are the same, or one precedes the other, we convert each to calendar time and then compare their calendar times.

We create a local date by initialising the fields of the *tm* structure. Then we create a calendar date with a call to *mktime()*.

```
/* eqdates.c: determines whether two dates are identical */

#include <stdio.h>
#include <time.h>

int main()
{
  struct tm locTime1, locTime2, locTime3;

  locTime1.tm_mday = 14;
  locTime1.tm_mon = 12 - 1;
  locTime1.tm_year = 2010 - 1900;
  locTime1.tm_hour = locTime1.tm_min = locTime1.tm_sec = 1;

  locTime2.tm_mday = 14;
  locTime2.tm_mon = 12 - 1;
  locTime2.tm_year = 2010 - 1900;
  locTime2.tm_hour = locTime2.tm_min = locTime2.tm_sec = 1;

  locTime3.tm_mday = 15;
  locTime3.tm_mon = 12 - 1;
  locTime3.tm_year = 2010 - 1900;
  locTime3.tm_hour = locTime3.tm_min = locTime3.tm_sec = 1;

  printf("Expect equal: ");
  if (mktime(&locTime1) == -1)
    printf("invalid date\n");
  else if (mktime(&locTime1) == mktime(&locTime2))
    printf("equal\n");
  else
    printf("not equal\n");

  printf("Expect not equal: ");
  if (mktime(&locTime3) == -1)
    printf("invalid date\n");
  else if (mktime(&locTime1) == mktime(&locTime3))
    printf("equal\n");
  else
    printf("not equal\n");
  return 0;
}
```



20.7 Formatting Date and Time

strftime(), found in *time.h*, is the function that formats components of local time.

strftime() uses the following conversion specifiers:

a	short weekday name
A	full weekday name
b	short month name
B	full month name
c	default date and time
d	day of the month, 00..31
H	hour (24 hour clock) 00..23
I	hour (12 hour clock) 00..12
J	day of the year, 001..365
m	month number, 01..12
M	minute, 00..59
P	AM or PM
S	second, 00..59
U	week of the year, 00..53. Sunday is the first day of the week.
w	day of the week, 0..6. Sunday is day zero.
W	week of the year, 00.53. Monday is the first day of the week,
x	default date
X	default time
y	year without century 00..99
Y	year with century
Z	time zone, if available, null otherwise

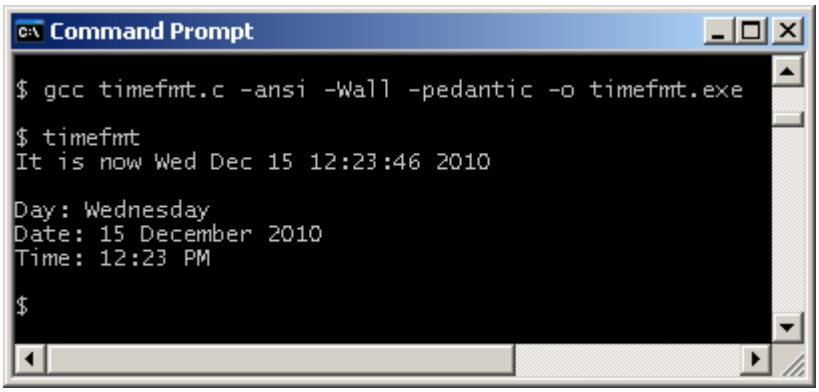
The following call to *strftime()* puts a date of the form *Date: 15 December 2010* into string from *localTime*.

```
strftime(string, BUFSIZ, "Date: %d %B %Y\n", localTime);
```

string must be declared large enough to hold the string plus null. *BUFSIZ* is the maximum number of characters that can be written into *string*.

Here is a complete program and its run.

```
/* timefmt.c: sets date and time formats */

#include <stdio.h>
#include <time.h>

int main()
{
  char string[BUFSIZ];
  time_t calendarTime;
  struct tm *localTime;

  calendarTime = time(NULL);
  localTime = localtime(&calendarTime);
  printf("It is now %s\n", asctime(localTime));

  strftime(string, BUFSIZ, "Day: %A\n", localTime);
  printf("%s", string);

  strftime(string, BUFSIZ, "Date: %d %B %Y\n", localTime);
  printf("%s", string);

  strftime(string, BUFSIZ, "Time: %H:%M %p\n", localTime);
  printf("%s", string);

  return 0;
}
```



Exercise 20.1

1. Make a list of all the date and time types and functions used in this chapter. Explain each item on the list.

2. Write a program to print the calendar date and time on your computer system.

3. Write a program that gets a date from the user and prints it.

4. A library lends books to students for up to two weeks only. Write a program that gets today's date from the computer system and calculates and prints the date in two weeks time.

5. Write and test a function that compares two dates and returns -1 if the first date come before the second, 0 if both dates are identical, and +1 if the first date comes after the second.

6. Write and test a program that will output the number of days between two dates.

7. A utility company gives its customers a small discount off their next bill if the customer pays within 14 days of the date on the bill. Write and test a program that inputs the billing date and the date the bill was paid, and output whether the customer is to receive a discount, or not.

We have looked at date and time. **Next** we take a look at ... who knows!.

Bibliography

Kernighan B and Ritchie D *The C Programming Language* Prentice Hall 1988
Mark Williams Company *ANSI C A Lexical Guide* Prentice Hall 1988