C++ time measurements (MS VCPP)

Contents

- Microsoft Visual C++ Time Management
- Clock function
- Time function

Microsoft Visual C++ Time Management

Use these functions to get the current time and convert, adjust, and store it as necessary. The current time is the system time.

The _**ftime** and **localtime** routines use the **TZ** environment variable. If **TZ** is not set, the run-time library attempts to use the time-zone information specified by the operating system. If this information is unavailable, these functions use the default value of PST8PDT. For more information on **TZ**, see _tzset; also see _daylight, timezone, and _tzname.

Source: MS VCPP help doc

MS VC++ Time Routines

asctime, _wasctime, clock, ctime, _wctime, difftime, _ftime, _futime, gmtime, localtime, mktime, _strdate, _wstrdate, strftime, wcsftime, strtime, _wstrtime , time, _tzset, _utime, _wutime

Source: MS VCPP help doc

Function	Use
asctime, _wasctime	Convert time from type **struct tm** to character string
clock	Return elapsed CPU time for process
ctime, _wctime	Convert time from type **time_t** to character string
Difftime	Compute difference between two times
_ftime	Store current system time in variable of type **struct _timeb**
_futime	Set modification time on open file

Source: MS VCPP help doc

Function	**Use**
gmtime	Convert time from type **time_t** to **struct tm**
localtime	Convert time from type **time_t** to **struct tm** with local correction
mktime	Convert time to calendar value
_strdate, _wstrdate	Return current system date as string
strftime, wcsftime	Format date-and-time string for international use
_strtime, _wstrtime	Return current system time as string

Source: MS VCPP help doc

Function	Use
time	Get current system time as type **time_t**
_tzset	Set external time variables from environment time variable **TZ**
_utime, _wutime	Set modification time for specified file using either current time or time value stored in structure

Source: MS VCPP help doc

The Clock Function

```
#include <time.h>

clock_t clock( void );
```

The **clock** function tells how much processor time the calling process has used. The time in seconds is approximated by dividing the clock return value by the value of the **CLOCKS_PER_SEC** constant. In other words, **clock** returns the number of processor timer ticks that have elapsed. A timer tick is approximately equal to 1/**CLOCKS_PER_SEC** second.

If the amount of elapsed time is unavailable, the function returns –1, cast as a **clock_t**.

Source: MS VCPP help doc

clock_t

- Data structure used to store the time values
- Used by clock_t clock(void)
- Measures the number of timer ticks
- Can be cast and printed as unsigned integer

Processor Time Measurement

```
#include<time.h>       //  clock_t clock( void );
double sec(void)
{
      return double(clock())/double(CLOCKS_PER_SEC);
}

double millisec(void)
{
      return clock()*1000./CLOCKS_PER_SEC;
}
```

Time Function

- the **time** function returns the current time as the number of seconds elapsed since midnight on January 1, 1970. In Microsoft C/C++ version 7.0, **time** returned the current time as the number of seconds elapsed since midnight on December 31, 1899.

```
/************************************************************\
|  Program:  BUBBLE sort analysis                            |
|  Problem:  Sort an integer array using Bubble sort         |
|  Purpose:  Measure the CPU time for Bubble sort            |
|  Author :  Jozo J. Dujmovic                                |
|  Date   :  9/14/2008                                       |
\************************************************************/
#include <iostream.h>
#include <time.h>
#include <iomanip.h>
#include <stdlib.h>

void MakeRandArray(int a[], int n)
{   int i;
    for(i=0; i<n; i++) a[i]=rand();
}

double seconds() { return double(clock())/CLOCKS_PER_SEC; }

void swap(int& a, int& b)
{
        int temp = a; a = b; b = temp;
}
```

```
void BubbleSort(int a[], int na)
{
  int i, done=0;   // Sort termination flag
  while((! done) && (na > 1))
  {
    done = 1;
    for(i=0; i < na-1; i++)
       if(a[i] > a[i+1])
       {
              swap(a[i], a[i+1]); done = 0;
       }
    na--;
  }
}

int test(int a[], int na) // Test of correct sorting
{ ;
  for (int i=0; i < na-1; i++) if(a[i]>a[i+1]) return 0;
  return 1;
}
```

```
int main()
{  int a[100000], na, i, delta = 5000;
   double t1,t2;
   cout.setf(ios::fixed, ios::floatfield);
   cout.setf(ios::showpoint);
   cout << "\n\nRUN TIME ANALYSIS FOR A BUBBLE SORT PROGRAM\n";

   for(na=delta; na<=10*delta; na += (delta/2))
   {  MakeRandArray(a, na);
      t1 = seconds();
           BubbleSort(a, na);
      t2 = seconds();
      if(test(a, na))
          {cout << "\nn =" << setw(6) << na
                << "  T =" << setprecision(2) << setw (6) << t2-t1
                << " sec  |";
            for(i=0; i<int(0.5+t2-t1); i++) cout << "*"; flush(cout);
          }
      else
            cout << "Bubble sort error";
   }
   cout << "\n\n";
   return 0;
}
```

```
RUN TIME ANALYSIS FOR A BUBBLE SORT PROGRAM

n =  5000  T =  0.36 sec  |
n =  7500  T =  0.80 sec  |*
n = 10000  T =  1.42 sec  |*
n = 12500  T =  2.22 sec  |**
n = 15000  T =  3.20 sec  |***
n = 17500  T =  4.38 sec  |****
n = 20000  T =  5.73 sec  |******
n = 22500  T =  7.22 sec  |*******
n = 25000  T =  8.88 sec  |*********
n = 27500  T = 10.81 sec  |***********
n = 30000  T = 12.83 sec  |*************
n = 32500  T = 15.08 sec  |***************
n = 35000  T = 17.47 sec  |*****************
n = 37500  T = 20.17 sec  |********************
n = 40000  T = 22.95 sec  |***********************
n = 42500  T = 25.81 sec  |**************************
n = 45000  T = 28.88 sec  |*****************************
n = 47500  T = 32.31 sec  |********************************
n = 50000  T = 35.74 sec  |************************************
```

If the bubble sort time complexity is O(n^2) then T(n)=c n^2 and T(2n) = 4 T(n)

T(25000) = 8.88 sec
T(50000)=4*T(25000)=35.52 sec

Debug version

```
RUN TIME ANALYSIS FOR A BUBBLE SORT PROGRAM

n =  5000  T =  0.11 sec  |
n =  7500  T =  0.25 sec  |
n = 10000  T =  0.44 sec  |
n = 12500  T =  0.69 sec  |*
n = 15000  T =  0.98 sec  |*
n = 17500  T =  1.34 sec  |*
n = 20000  T =  1.77 sec  |**
n = 22500  T =  2.25 sec  |**
n = 25000  T =  2.75 sec  |***
n = 27500  T =  3.34 sec  |***
n = 30000  T =  3.97 sec  |****
n = 32500  T =  4.64 sec  |*****
n = 35000  T =  5.41 sec  |*****
n = 37500  T =  6.23 sec  |******
n = 40000  T =  7.06 sec  |*******
n = 42500  T =  7.97 sec  |********
n = 45000  T =  8.92 sec  |*********
n = 47500  T =  9.97 sec  |**********
n = 50000  T = 11.00 sec  |***********
```

If the bubble sort time complexity is O(n^2) then T(n)=c n^2 and T(2n) = 4 T(n)

T(25000) = 2.75 sec
T(50000)=4*T(25000)=11 sec

Release version

Speedup = 35.74/11 = 3.25 times

```
/************************************************************\
|  Program:  TIMES                                           |
|  Problem:  Measure the elapsed real time and CPU time      |
|  Purpose:  Test the clock and time functions from <time.h> |
|  Author :  Jozo J. Dujmovic                                |
|  Date   :  9/14/2008                                       |
\************************************************************/
#include <iostream.h>
#include <time.h>

int main()
{
   long int i,j;
   clock_t ticks;   // clock_t = an arithmetic type (defined in time.h)
                    // capable of representing time (usually long int)
   time_t t1,t2;    // time_t = an arithmetic type (defined in time.h)
                    // capable of representing time (usually long int)

   ticks = clock(); // CPU time from the beginning of program expressed as
                    // the number of "ticks" (real time clock increments)
   t1 = time(NULL); // real (calendar) time expressed in seconds;
                    // NULL = null pointer constant

   cout << "\n\nMEASUREMENT OF ELAPSED TIME\n\n"
        << "CLOCKS_PER_SEC = " << CLOCKS_PER_SEC << " [ticks]"
        << "\nClock() (initial value)     = " << ticks
        << "\nCPU seconds (initial value) = " << double(ticks)/CLOCKS_PER_SEC
        << "\nt1 = time(NULL) = real time in seconds = " << t1
        << "\n<<<<<<<< Counting to 5 billion >>>>>>>>\n";
   t1 = time(NULL);

   // Program segment whose elapsed time has to be measured
   for(i=0; i<50; i++)
   {
      for(cout << '.',j=0; j<100000000; j++);
      flush(cout);
   }
   // End of measured program segment

   ticks = clock(); t2 = time(NULL);

   cout << "\nClock() (final value)     = " << ticks
        << "\nCPU seconds (final value) = "
        << double(ticks)/CLOCKS_PER_SEC
        << "\nt2 = time(NULL) = real time in seconds = " << t2
        << "\nElapsed (real) time t2-t1 = " << t2 - t1
        << " seconds\n";
   return 0;
}
```

Debug version

Dell XPS400

MS Visual C++ 6.0

```
MEASUREMENT OF ELAPSED TIME

CLOCKS_PER_SEC = 1000 [ticks]
Clock() (initial value)     = 0
CPU seconds (initial value) = 0
t1 = time(NULL) = real time in seconds = 1221470237
<<<<<<<< Counting to 5 billion >>>>>>>>
..................................................
Clock() (final value)     = 12562
CPU seconds (final value) = 12.562
t2 = time(NULL) = real time in seconds = 1221470249
Elapsed (real) time t2-t1 = 12 seconds
Press any key to continue
```

Note: clock increment is either 15 or 16. Therefore, the measured time 12.562 seconds can be misleading because it does not have accurate milliseconds. The error can be up to 16 milliseconds (0.016 sec)

Clock resolution for Dell XPS400 (dual core) under Win XP

```
r = 0.015 sec  Clock increment = 15
r = 0.016 sec  Clock increment = 16
r = 0.016 sec  Clock increment = 16
r = 0.015 sec  Clock increment = 15
r = 0.016 sec  Clock increment = 16
r = 0.015 sec  Clock increment = 15
r = 0.016 sec  Clock increment = 16
r = 0.016 sec  Clock increment = 16
r = 0.015 sec  Clock increment = 15
r = 0.016 sec  Clock increment = 16
r = 0.016 sec  Clock increment = 16
MS VC++6.0, WinXP results for N = 1000 iterations
CLOCKS_PER_SEC = 1000
Clock increment min,ave,max = 15 , 15.625 , 16
r = 0.015625 sec,  1/r = 64 increments per sec
```

A measurement from 1995

Note low CLOCK_PER_SEC

```
MEASUREMENT OF ELAPSED TIME FOR A PC (25 MHz 486SX)

CLOCKS_PER_SEC = 18.2 [ticks]
Clock() (initial value)     = 0
CPU seconds (initial value) = 0
t1 = time(NULL) = real time in seconds = 816558504
<<<<<<<< Counting to 40 million >>>>>>>>
........................................
Clock() (final value)     = 412
CPU seconds (final value) = 22.637363
t2 = time(NULL) = real time in seconds = 816558527
Elapsed (real) time t2-t1 = 23 seconds
```