```
1   #include <COOL/Timer.h>                          // Includes COOL timer class

2   int main (void) {
3    Timer t1;                                       // Create a timer object
4    t1.mark ();                                     // Set start reference point
5    for (int i = 0, j = 0; i < 10000; i++)          // Loop for 10000 times and
6     j = j + i;                                     // Sum up numbers
7    cout << "Summation of integers from 0 through 10000 took ";
8    cout << t1.real () << " milliseconds\n";        // Output time since mark
9    return 0;                                       // Return valid completion code
10  }
```

Line 1 includes the COOL **Timer** header file. Line 3 creates a new timer object and line 4 establishes the starting point of the timing operation by setting the mark. Lines 5 and 6 implement a loop counting from 1 to 10000 that calculates the sum of these values. Line 8 contains an embedded call to the timer object to report the elapsed time from the mark to now. Note that since this call is embedded in the output statements, the time reported is not technically correct. A more accurate reading could be established by calling this function and saving the value in a temporary variable for later use in the output statement. Finally, line 9 returns a successful completion code.

The following shows the output of the program:

```
Summation of integers from 0 through 10000 took 20 milliseconds
```

Due to operating system dependencies, the accuracy of all member function results may not be as documented. For example, some operating systems do not support timers with microsecond resolution. In those cases, the values returned are provided to the nearest millisecond or other unit of time as appropriate. See the `Timer.h` header file for system-specific notes.

Name: **Timer** — A timing facility for C++

Synopsis: **#include** <COOL/Timer.h>

Base Classes: **Generic**

Friend Classes: None

Constructors:

Timer ();
Creates an instance of the **Timer** class with the mark set to creation time.

Member Functions:

long all();
Returns the number of milliseconds spent in the user process and the operating system since the last reference point (mark).

long all_usec();
Returns the number of microseconds spent in the user process and the operating system since the last reference point (mark).

void mark ();
Sets the reference time to now.

long real();
Returns the number of milliseconds of wall clock time since the last reference point (mark).

long system();
Returns the number of milliseconds spent in the operating system since the last reference point (mark).

long system_usec();
Returns the number of microseconds spent in the operating system since the last reference point (mark).

long user();
Returns the number of milliseconds spent in the user process since the last reference point (mark).

long user_usec();
Returns the number of microseconds spent in the user process since the last reference point (mark).

Timer Example

4.9 The following program uses the COOL **Timer** class to calculate the time for a loop to sum up a sequence of integer values. Note that although this example reports results in milliseconds, support timing granularity on your particular computer and operating system may be different.

```
1   #include <COOL/Date_Time.h>                  // Include Date_Time class

2   int main (void) {
3    set_default_country (UNITED_STATES);        // Set default country code
4    set_default_time_zone (US_CENTRAL);         // Set default time zone
5    Date_Time d1;                               // Create Date_Time object
6    d1.set_local_time ();                       // Set current system time
7    cout << "Local date/time is: " << d1 << "\n"; // Output date in US format
8    d1.set_country (UNITED_KINGDOM);            // Set country to UK
9    d1.set_time_zone (GB_EIRE);                 // Set Greenwich Mean Time
10   cout << "GMT date/time is: " << d1 << "\n";  // Output date/time at GMT
11   d1.parse("1 April 1890, 4:30pm");           // Parse some date in UK format
12   cout << "Date/time parsed is: " << d1 << "\n"; // Output date/time parsed
13   d1.set_country (FRANCE);                    // Set country to France
14   d1.set_time_zone (WET);                     // Western European Time zone
15   cout << "Date/time in France: "<< d1 << "\n"; // Output date/time in France
16   Date_Time d2;                               // Create another object
17   d2.set_local_time ();                       // Set current system time
18   cout << "Date/time set is: " << d2 << "\n";  // Output date in US format
19   d2.decr_month (3);                          // Move back three months
20   cout << "Date/time three months earlier: " << d2 << "\n"; // Output date
21   cout << "Duration between dates is ";       //
22   cout << d1.ascii_duration (d2) << "\n";     // Output time duration
23   return 0;                                   // Return valid success code
24  }
```

Line 1 includes the COOL **Date_Time** class header file. Lines 3 and 4 establish the default country and time zone for all **Date_Time** objects in this application to be UNITED_STATES and US_CENTRAL, respectively. Line 5 instantiates an uninitialized object, line 6 sets its value to be the local system date and time, and line 7 outputs this value. Lines 8 and 9 change the country to UNITED_KINGDOM and the time zone to GB_EIRE (Greenwich Mean Time). Line 10 outputs the time zone corrected date and time values in English format. Line 11 sets the new value of the **Date_Time** object by parsing a character string, and line 12 outputs the new setting. Lines 13 and 14 change the country to FRANCE and the time zone to WET and output the value again. Note that the time zone didn't affect the value printed, but the format based on the country code changed. Lines 16 through 18 output another **Date_Time** object for the UNITED_STATES in US_MOUNTAIN time zone, and sets its value to the current system time. Line 19 decrements the date by three months, and line 20 shows the resulting value. Lines 21 and 22 output in ASCII format the time difference between the two objects. Finally, line 23 exits the program with a valid successful completion code.

The following shows the output of the program:

```
Local date/time is: United States 02-13-1990 11:28:40 US/Central
GMT date/time is: United Kingdom 13-02-1990 17:28:40 GB-Eire
Date/time parsed is: United Kingdom 01-04-1890 16:30:00 GB-Eire
Date/time in France: France 01-01/1990 16:30:00 WET
Date/time set is: United States 02-13-1990 10:28:40 US/Mountain
Date/time three months earlier: United States 11-15-1989 10:28:40 US/Mountain
Duration between dates is 99 years, 35 weeks, 2 days, 0 hours, 58 minutes, 40 seconds
```

Timer Class

4.8 The **Timer** class is publicly derived from the **Generic** class and provides an interface to system timing. It allows a C++ program to record the time between a reference point (mark) and now. This class uses the system **time**(2) interface to provide time resolution at either millisecond or microsecond granularity, depending upon operating system support and features. Since the time duration is stored in a 32-bit word, the maximum time period before rollover occurs is about 71 minutes.

SUNDAY	“Sunday”
MONDAY	“Monday”
TUESDAY	“Tuesday”
WEDNESDAY	“Wednesday”
THURSDAY	“Thursday”
FRIDAY	“Friday”
SATURDAY	“Saturday”

Enumeration Declaration	*Character String*
JANUARY	“January”
FEBRUARY	“February”
MARCH	“March”
APRIL	“April”
MAY	“May”
JUNE	“June”
JULY	“July”
AUGUST	“August”
SEPTEMBER	“September”
OCTOBER	“October”
NOVEMBER	“November”
DECEMBER	“December”

Date_Time Example

4.7 The following program creates two **Date_Time** objects and initializes one to the current system date and time and the other to the date and time specified in a character string. Several conversions between country formats and time zones are performed, along with manipulating one of the dates by subtracting three months. Finally, the length of time between the two objects is displayed.

Country.h File

4.5 The `country.h` include file contains enumeration declarations for country names of type **country**. The file declares a static **char*** array of printable country names. The constants in the enumerated type can be used as indexes for these names. In the following table, the enumeration declaration is on the left and the static **char*** string is on the right.

Name: `country.h` — Symbolic and string country names

Synopsis: **#include** <COOL/country.h>

Enumeration Declaration	*Character String*
UNKNOWN_COUNTRY	"Unknown Country"
UNITED_STATES	"United States"
FRENCH_CANADIAN	"French Canadian"
LATIN_AMERICA	"Latin America"
NETHERLANDS	"Netherlands"
BELGIUM	"Belgium"
FRANCE	"France"
SPAIN	"Spain"
ITALY	"Italy"
SWITZERLAND	"Switzerland"
UNITED_KINGDOM	"United Kingdom"
DENMARK	"Denmark"
SWEDEN	"Sweden"
NORWAY	"Norway"
GERMANY	"Germany"
PORTUGAL	"Portugal"
FINLAND	"Finland"
ARABIC_COUNTRIES	"Arabic Countries"
ISRAEL	"Israel"

Calendar.h File

4.6 The `calendar.h` include file contains enumeration declarations for day and month names of the types **day_of_week** and **months**. The file declares two static **char*** arrays of printable day and month names. The constants in the enumerated types can be used as indexes for these names. In addition, an array indexed by type **month** specifying the number of days in the month is also provided. Finally, the file defines several macros for typical date and time constants, along with a macro determining if a year is a leap year. In the following tables, the enum declaration is on the left and the static **char*** string is on the right.

Name: `calendar.h` — Symbolic and string calendar names

Synopsis: **#include** <COOL/calendar.h>

Enumeration Declaration	*Character String*

Time_zone.h File

4.4 The `time_zone.h` include file contains enumeration declarations for time zone names of type **time_zone**. The file declares a static **char*** array of printable names. The constants in the enumerated type can be used as indexes for these names. In the following table, the enum declaration is on the left and the matching static **char*** string is on the right.

Name: `time_zone.h` — Symbolic and string time zone names

Synopsis: **#include** <COOL/time_zone.h>

Enumeration Declaration	*Character String*
UNKNOWN_TIME_ZONE	"Unknown Time Zone"
US_EASTERN	"US/Eastern"
US_CENTRAL	"US/Central"
US_MOUNTAIN	"US/Mountain"
US_PACIFIC	"US/Pacific"
US_PACIFIC_NEW	"US/Pacific–New"
US_YUKON	"US/Yukon"
US_EAST_INDIANA	"US/East–Indiana"
US_ARIZONA	"US/Arizona"
US_HAWAII	"US/Hawaii"
CANADA_NEWFOUNDLAND	"Canada/Newfoundland"
CANADA_ATLANTIC	"Canada/Atlantic"
CANADA_EASTERN	"Canada/Eastern"
CANADA_CENTRAL	"Canada/Central"
CANADA_EAST_SASKATCHEWAN	"Canada/East–Saskatchewan"
CANADA_MOUNTAIN	"Canada/Mountain"
CANADA_PACIFIC	"Canada/Pacific"
CANADA_YUKON	"Canada/Yukon"
GB_EIRE	"GB–Eire"
WET	"WET"
ICELAND	"Iceland"
MET	"MET"
POLAND	"Poland"
EET	"EET"
TURKEY	"Turkey"
W_SU	"W–SU"
PRC	"PRC"
KOREA	"Korea"
JAPAN	"Japan"
SINGAPORE	"Singapore"
HONGKONG	"Hongkong"
ROC	"ROC"
AUSTRALIA_TASMANIA	"Australia/Tasmania"
AUSTRALIA_QUEENSLAND	"Australia/Queensland"
AUSTRALIA_NORTH	"Australia/North"
AUSTRALIA_WEST	"Australia/West"
AUSTRALIA_SOUTH	"Australia/South"
AUSTRALIA_VICTORIA	"Australia/Victoria"
AUSTRALIA_NSW	"Australia/NSW"
NZ	"NZ"

void set_local_time ();
Sets the date and time to local time as determined by the time zone and country code values.

inline void set_time_zone (**time_zone** *tz*);
Sets the time zone to the value *tz*.

void start_day (**int** *n* = 1);
Advances the time the specified number of days, setting the time to 00:00:00. The default is one.

void start_hour (**int** *n* = 1);
Advances the time by the specified number of hours, setting the time to hh:00:00. The default is one.

void start_min (**int** *n* = 1);
Advances the time by the specified number of minutes, setting the time to hh:mm:00. The default is one.

void start_month (**int** *n* = 1);
Advances the time by the specified number of months, setting the time to 01/mm/yyyy 00:00:00. The default is one.

void start_week (**int** *n* = 1);
Advances the time by the specified number of weeks, setting the time to Monday 00:00:00. The default is one.

void start_year (**int** *n* = 1);
Advances the time by the specified number of years, setting the time to 01/01/yyyy 00:00:00. The default is one.

Friend Functions:

friend istream operator>> (**istream&** *is*, **Date_Time&** *dt*);
Overloads the input operator to read the input stream *is*, and parses the character string containing the date and time information. The result is returned in the date and time object *dt*.

friend ostream operator<< (**ostream&** *os*, **const Date_Time*** *dt*);
Overloads the output operator for a pointer to a date-and-time object. The object writes the output stream *os* with a character string representing the date and time object *dt* formatted for the appropriate country.

friend ostream operator<< (**ostream&** *os*, **const Date_Time&** *dt*);
Overloads the output operator for a reference to a date-and-time object. The object writes the output stream *os* with a character string representing the date and time object *dt* formatted for the appropriate country.

inline friend void set_default_country (**country** *c*);
Sets the default country for the class to the value *c*.

inline friend void set_default_time_zone (**time_zone** *tz*);
Sets the default time zone for the class to the value *tz*.

inline long operator– (**const Date_Time&** *dt*);
Computes the interval of time between the date and time object and *dt*.

Date_Time& operator= (**const Date_Time&** *dt*);
Overloads the assignment operator to replicate the value of one date and time object to another.

Date_Time& operator+= (**long** *seconds*);
Performs interval addition and assignment.

Date_Time& operator–= (**long** *seconds*);
Performs interval subtraction and assignment.

inline Boolean operator== (**const Date_Time&** *dt*) **const**;
Overloads the equality operator for the **Date_Time** class. This function returns **TRUE** if two objects represent the same time; otherwise, this function returns **FALSE**.

inline Boolean operator!= (**const Date_Time&** *dt*) **const**;
Overloads the inequality operator for the **Date_Time** class. This function returns **FALSE** if two objects represent the same time; otherwise, this function returns **TRUE**.

inline Boolean operator< (**const Date_Time&** *dt*) **const**;
Overloads the less-than operator for the **Date_Time** class. This function returns **TRUE** if the date and time object represents a date and time before *dt*; otherwise, this function returns **FALSE**.

inline Boolean operator<= (**const Date_Time&** *dt*) **const**;
Overloads the less-than-or-equal operator for the **Date_Time** class. This function returns **TRUE** if the date and time object represents a date and time before or equal to *dt*; otherwise, this function returns **FALSE**.

inline Boolean operator> (**const Date_Time&** *dt*) **const**;
Overloads the greater-than operator for the **Date_Time** class. This function returns **TRUE** if the date and time object represents a date and time after *dt*; otherwise, this function returns **FALSE**.

inline Boolean operator>= (**const Date_Time&** *dt*) **const**;
Overloads the greater-than-or-equal operator for the **Date_Time** class. This function returns **TRUE** if the date and time object represents a date and time equal to or after *dt*; otherwise, this function returns **FALSE**.

void parse (**char*** *str*, **int** *settz* = 0);
Parses the character string *str* input and fills all appropriate data members of the date and time object. If no value is provided for *settz,* the parsing algorithm does not search for a time zone. The parser recognizes most valid input and always parses relative to the time zone. Fields not specified are defaulted where appropriate. Illegal input results in an **Error** exception being raised.

inline void set_country (**country** *c*);
Sets the country to the value *c*.

void set_gm_time ();
Sets the date and time to Greenwich mean time.

void end_year (**int** *n* = 1);
Advances the time by the specified number of months, setting the time to 31/12/yyyy 23:59:59. The default is one.

inline const char* get_country () **const**;
Returns the country in ASCII format.

inline int get_hour () **const**;
Returns the value of the hour data member in the object (0–23).

inline int get_mday () **const**;
Returns the value of the day of the month data member in the object (1–31).

inline int get_min () **const**;
Returns the value of the minutes data member in the object (0–59).

inline int get_mon () **const**;
Returns the value of the months data member in the object (0–11).

inline int get_sec () **const**;
Returns the value of the seconds data member in the object (0–59).

inline const char* get_time_zone () **const**;
Returns the time zone in ASCII format.

inline int get_wday () **const**;
Returns the value of the day of the week data member in the object (Sunday=0).

inline int get_yday () **const**;
Returns the value of the day of the year data member in the object (0–365)

inline int get_year () **const**;
Returns the value of the year data member in the object.

void incr_day (**int** *n* = 1);
Increments the time by the specified number of days. The default is one.

void incr_hour (**int** *n* = 1);
Increments the time by the specified number of hours. The default is one.

void incr_min (**int** *n* = 1);
Increments the time by the specified number of minutes. The default is one.

void incr_month (**int** *n* = 1);
Increments the time by the specified number of months. The default is one.

void incr_sec (**int** *n* = 1);
Increments the time by the specified number of seconds. The default is one.

void incr_week (**int** *n* = 1);
Increments the time by the specified number of weeks. The default is one.

void incr_year (**int** *n* = 1);
Increments the time by the specified number of years. The default is one.

inline Boolean is_day_light_savings () **const**;
Returns **TRUE** if daylight saving time is in effect; otherwise, returns **FALSE**.

Date_Time (**time_zone** *tz*, **countr***y c*);
Allocates a date and time object with time zone *tz* and country code *c*.

Member Functions:

const char* ascii_date () **const**;
Returns the date in ASCII format for the appropriate time zone and country.

const char* ascii_date_time () **const**;
Returns the date and time in ASCII format for the appropriate time zone and country.

const char* ascii_duration (**const Date_Time&** *dt*) **const**;
Returns the duration of time between the date/time object and *dt* in ASCII format.

const char* ascii_time () **const**;
Returns the time in ASCII format for the appropriate time zone and country.

inline void decr_day (**int** *n* = 1);
Decrements the time by the specified number of days. The default is one.

inline void decr_hour (**int** *n* = 1);
Decrements the time by the specified number of hours. The default is one.

inline void decr_min (**int** *n* = 1);
Decrements the time by specified number of minutes. The default is one.

void decr_month (**int** *n* = 1);
Decrements the time by specified number of months. The default is one.

inline void decr_sec (**int** *n* = 1);
Decrements the time by specified number of seconds. The default is one.

inline void decr_week (**int** *n* = 1);
Decrements the time by the specified number of weeks. The default is one.

void decr_year (**int** *n* = 1);
Decrements the time by the specified number of years. The default is one.

void end_day (**int** *n* = 1);
Advances the time by the specified number of days, setting the time to 23:59:59. The default is one.

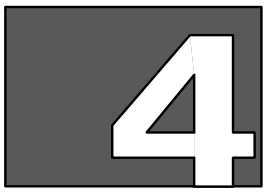
void end_hour (**int** *n* = 1);
Advances the time by the specified number of hours, setting the time to hh:59:59. The default is one.

void end_min (**int** *n* = 1);
Advances the time by the specified number of minutes, setting the time to hh:mm:59. The default is one.

void end_month (**int** *n* = 1);
Advances the time by the specified number of months, setting the time to 31/mm/yyyy 23:59:59. The default is one.

void end_week (**int** *n* = 1);
Advances the time by the specified number of weeks, setting the time to Sunday 23:59:59. The default is one.

SYSTEM INTERFACE CLASSES



Introduction

4.1 The COOL system interface classes encapsulate common system-specific functionality such as date-and-time manipulation and timing facilities. These classes provide a single interface for an application program no matter which of the supported platforms it is running on. This facilitates a single source base for an application designed to run on several types of hardware. The following classes are discussed in this section:

- **Date_Time**
- **Timer**

The **Date_Time** class implements time zone-independent date and time functions, including time zone changes, calendar date manipulation, and complete input parsing and output formatting capability for significant country or language formats. The **Timer** class uses the system **time**(2) interface to provide time resolution between a reference point and now. The accuracy of the time period reported is system-dependent, but will generally be either at millisecond or microsecond granularity.

Requirements

4.2 This section discusses the system interface classes. It assumes you have a working understanding of the C++ language and type system. In addition, you should understand the distinction between overloaded operators and friend functions.

Date_Time Class

4.3 The **Date_Time** class executes time zone-independent date and time functions. This class supports calendar operations and input and output based upon the value of an environmental synonym, such as **US_CENTRAL.** This class supports all time zones in the world, along with several special cases requiring alternate handling based upon political or daylight saving time differences. Unlike the ANSI C date and time functions, this class supports dates before the epoch (January 1, 1970). Year values specified between 0 and 99 are assumed to be in the twentieth century.

Name: **Date_Time** — Time zone–independent date and time class

Synopsis: **#include** <COOL/Date_Time.h>

Base Classes: **Generic**

Friend Classes: None

Public Constructors: **Date_Time** ();

Allocates a date and time object with the default time zone and country. A Warning exception is raised if the default country or the default time has not been set for the class.

Date_Time (**const Date_Time&** *dt*);

Duplicates the size and entries of a date and time object *dt*.

Printed on: Wed Apr 18 07:04:29 1990

Last saved on: Tue Apr 17 13:45:25 1990

Document: s4

For: skc