CONTENIDO DE LA LECCIÓN 13

BIBLIOTECA ESTÁNDAR DE *C++*

LECCIÓN 13

BIBLIOTECA ESTÁNDAR DE *C++*

INTRODUCCIÓN

Los módulos en ***C++*** se llaman ***funciones*** y *clases*. Los programas en ***C++*** generalmente se escriben combinando funciones nuevas que el programador escribe con funciones ***preempacadas*** disponibles en la ***biblioteca estándar*** *de* ***C++***, y combinando clases nuevas que el programador escribe con clases ***preempacadas*** disponibles en varias ***bibliotecas*** *de clases*. En esta lección nos concentraremos en las funciones; el estudio de las clases es motivo de otro curso en la especialidad de computación.

La biblioteca estándar de ***C++*** contiene un vasto conjunto de funciones para realizar cálculos matemáticos comunes, manipulaciones de cadenas, manipulaciones de caracteres, entrada/salida, comprobación de errores y muchas otras operaciones útiles. Esto simplifica el trabajo del programador, pues estas funciones ofrecen muchas de las capacidades que se necesitan. Las funciones de la biblioteca estándar de ***C++*** se proporcionan como parte del entorno de programación de ***C++***.

Objetivos de esta lección:

- ***Utilizar*** *en sus programas, la* ***biblioteca estándar*** *de* ***C++***.
- ***Incluir*** *los* ***archivos de cabecera*** *para cada librería requerida en su programa.*

FUNCIONES DE LA BIBLIOTECA MATEMÁTICA (*math.h*)

Las funciones de la biblioteca matemática le permiten al programador efectuar ciertos cálculos matemáticos comunes. Las mismas han sido analizadas en la **lección 6** (**Operaciones básicas**) en el ejemplo ***FUNMAT.CPP***. (**Ejemplo 6.49**, *página* ***58***) Simplemente presentaremos en esta sección una tabla que resuma algunas de las funciones vistas: ***tabla 13.1***, en dicha tabla ***x*** y ***y*** son de la clase ***double***. También presentaremos algunos ejemplos interesantes en el que se manejen algunas de las funciones matemáticas.

Todas las funciones de la biblioteca matemática devuelven el tipo de datos ***double***. Para utilizar las funciones de la biblioteca matemática, incluya el archivo de encabezado ***math.h*** (en la nueva biblioteca estándar de ***C++*** se llama ***cmath***) Los argumentos de una función pueden ser ***constantes***, ***variables*** o ***expresiones***

Método	Descripción	Ejemplo
***ceil*(x)**	**Redondea *x*** al entero más pequeño no menor que **x**	***ceil*(9.2)** es **10.0** ***ceil*(-9.8)** es **–9.0**
***cos*(x)**	**Coseno trigonométrico** de ***x*** (***x*** en radianes)	***cos*(0.0)** es **1.0**
***exp*(x)**	**Función exponencial e^x**	***exp*(1.0)** es **2.71828** ***exp*(2.0)** es **7.38906**

Método	Descripción	Ejemplo
***fabs*(x)**	**Valor absoluto** de *x*	Si **x > 0** ***fabs*(x)** es **x** Si **x = 0** ***fabs*(x)** es **0.0** Si **x < 0** ***fabs*(x)** es **–x**
***floor*(x)**	**Redondea** *x* al entero más grande no mayor que **x**	***floor*(9.2)** es **9.0** ***floor*(-9.8)** es **–10.0**
***fmod*(x, y)**	**Residuos** de *x/y* como número de punto flotante	***fmod*(13.657, 2.333)** es **1.992**
***log*(x)**	**Logaritmo natural** de *x* (base *e*)	***log* (2.718282)** es **1.0** ***log*(7.389056)** es **2.0**
***log10*(x)**	**Logaritmo** de *x* (base 10)	***log*(10.0)** es **1.0** ***log*(100.0)** es **2.0**
***pow*(x, y)**	***x* elevado a la potencia *y*** (x^y)	***pow*(2, 7)** es **128** ***pow*(9., .5)** es **3**
***sin*(x)**	**Seno trigonométrico** de *x* (*x* en radianes)	***sin*(0.0)** es **0**
***sqrt*(x)**	**Raíz cuadrada** de *x*	***sqrt*(900.0)** es **30.0** **sqrt(9.0)** es **3.0**
***tan*(x)**	**Tangente trigonométrica** de *x* (*x* en radianes)	***tan*(0.0)** es **0**

*Tabla 13.1. Funciones más comúnmente usadas de la **biblioteca matemática***

GENERACIÓN DE NÚMEROS ALEATORIOS

El azar puede agregarse a las aplicaciones de cómputo mediante la función ***rand()*** de la biblioteca estándar. Considere la siguiente instrucción:

```
i = rand();
```

La función ***rand()*** genera un entero entre ***0*** y ***RAND_MAX*** (una constante simbólica definida en el archivo de encabezado <***stdlib.h***>) El valor de ***RAND_MAX*** debe ser cuando menos ***32767***, el valor máximo positivo para un entero de dos bytes (16 bits) Si ***rand()*** en realidad produce enteros al azar, todos los números entre ***0*** y ***RAND_MAX*** tienen la misma ***posibilidad*** (o probabilidad) de ser seleccionados cada vez que se invoca a ***rand()***

El rango de valores producido directamente por ***rand()*** con frecuencia es distinto de lo que se necesita para una aplicación específica. Por ejemplo, un programa que simula el lanzamiento de una moneda requeriría solo ***0*** (para ***cara***) y ***1*** (para ***sol***) Un programa que simule el lanzamiento de un dado necesitaría enteros al azar del ***1*** al ***6*** (debido a que el dado tiene seis lados) Otro que prediga al azar el siguiente tipo de nave espacial (de cuatro posibilidades) que volará por el horizonte de un juego de video necesitaría enteros aleatorios del ***1*** al ***4***.

Para mostrar ***rand()***, desarrollaremos un programa que simule ***20*** lanzamientos de un dado e imprima el valor de cada lanzamiento. El prototipo de la función ***rand()*** puede encontrarse en <***stdlib.h***>. Utilizaremos el operador de módulo (***%***) junto con ***rand()***, como sigue:

```
rand() % 6
```

para producir enteros del *0* al *5*. A esto se le llama ***escalamiento***. El número **6** se llama ***factor de escalamiento***. Luego ***desplazamos*** el rango de los números producidos sumando *1* a nuestro resultado previo.

Ejemplo 13.1

El siguiente programa: ***DADOS.CPP***, ilustra la idea anterior.

```
/* El siguiente programa: DADOS.CPP, simula 20 lanzamientos de un dado. */

#include <iostream.h>          //Para cout y cin
#include <iomanip.h>           //Para setw()
#include <stdlib.h>            //Para rand()

void main(void)
        {
                for(int i = 1; i <= 20; i++)
                        {
                                cout << setw(10) << (1+rand() % 6);

                                if(i % 5 == 0)
                                        cout << endl;
                        }
        }//Fin de main()
```

Para mostrar que estos números suceden aproximadamente con la misma probabilidad, simulamos *6000* lanzamientos de los dados. Cada entero del *1* al *6* debe aparecer aproximadamente ***1000*** **veces**.

Ejemplo 13.2

El siguiente programa: ***FRECUENCIA.CPP***, ilustra la idea anterior.

```
/* El siguiente programa: FRECUENCIA.CPP, ilustra el lanzamiento de un dado 6000 veces */

#include <iostream.h>          //Para cout y cin
#include <iomanip.h>           //Para setw()
#include <stdlib.h>            //Para rand()

void main(void)
        {
                int     frecuencia1 = 0, frecuencia2 = 0,
                        frecuencia3 = 0, frecuencia4 = 0,
                        frecuencia5 = 0, frecuencia6 = 0,
                        cara;

                for(int lanzamiento = 1; lanzamiento <= 6000; lanzamiento++)
                        {
                                cara = 1 + rand() % 6;
```

```
                    switch(cara)
                        {
                            case 1:
                                ++frecuencia1;
                                break;

                            case 2:
                                ++frecuencia2;
                                break;

                            case 3:
                                ++frecuencia3;
                                break;

                            case 4:
                                ++frecuencia4;
                                break;

                            case 5:
                                ++frecuencia5;
                                break;

                            case 6:
                                ++frecuencia6;
                                break;

                            default:
                                cout << "¡Nunca deberá llegar aquí";
                        }//Fin de switch()
                }//Fin de for()

        cout << "               Cara" << setw(13) << "Frecuencia"
             << "\n               1" << setw(13) << frecuencia1
             << "\n               2" << setw(13) << frecuencia2
             << "\n               3" << setw(13) << frecuencia3
             << "\n               4" << setw(13) << frecuencia4
             << "\n               5" << setw(13) << frecuencia5
             << "\n               6" << setw(13) << frecuencia6 << endl;
}//Fin de main()
```

Como lo muestra la salida del programa, ***escalando*** y ***desplazando*** hemos utilizado la función ***rand()*** para simular de manera realista el lanzamiento de un dado. Observe que el programa nunca debería llegar al caso ***default*** incluido con la estructura ***switch***; no obstante, lo indicamos por ser una buena práctica.

Al volver a ejecutar el programa observe que se imprime la misma secuencia de valores. **¿Cómo pueden ser aleatorios estos números**? Es irónico que esta capacidad de repetición es una característica importante de la función ***rand()*** Al depurar un programa, esta repetición es importante para probar que las correcciones al programa funcionan de manera apropiada.

La función ***rand()*** de hecho genera ***números pseudoaleatorios***. La ***llamada repetida*** de ***rand()*** produce una secuencia de números que parece ser aleatoria. Sin embargo, la secuencia se repite cada vez que se ***ejecuta*** **el** ***programa***. Una vez que se ha depurado el programa, puede acondicionarse para que produzca una secuencia diferente de números aleatorios con cada ejecución. A esto se le llama ***aleatorización*** y se logra mediante la función ***srand()*** de la biblioteca estándar. La función ***srand()*** toma un argumento entero ***unsigned*** (conocido como ***semilla***) y ***siembra*** la función ***rand()*** para que genere una secuencia diferente de números aleatorios cada vez que se ejecute el programa.

Ejemplo 13.3

El siguiente programa: ***ALEATORIZACION.CPP***, ilustra el uso de la función ***srand()***

```
/* El siguiente programa: ALEATORIZACION.CPP, ilustra el uso de la función srand() */

#include <iostream.h>          //Para cout y cin
#include <iomanip.h>           //Para setw()
#include <stdlib.h>            //Para srand() y rand()

void main(void)
        {
                unsigned semilla;

                cout << "Teclee la semilla: ";
                cin >> semilla;
                cout << endl;
                srand(semilla);

                for(int i = 1; i <= 10; i++)
                        {
                                cout << setw(10) << 1 + rand() % 6;

                                if(i % 5 == 0)
                                        cout << endl;
                        }//Fin de for
        }// Fin de main()
```

La función ***srand()*** toma como argumento un valor ***unsigned int***. El prototipo de la función ***srand()*** está en el archivo de encabezado <***stdlib.h***> (***cstdlib*** en el nuevo estándar de ***C++***)

Ejecute el programa varias veces y observe los resultados. Cada vez que se ejecute el programa se obtiene una secuencia de números aleatorios diferentes, siempre y cuando se le proporcione una semilla diferente.

Si deseamos aleatorizar sin tener que sembrar un número cada vez, podemos dar una instrucción como:

```
srand(time(NULL));
```

Con esto la computadora lee su reloj, obteniendo de ahí el valor para la semilla. La función ***time()*** (con argumento ***NULL***, como se indica en la instrucción anterior) devuelve en segundos la ***hora calendario actual***. Este valor se convierte a un entero sin signo y se utiliza como la semilla para el generador de números aleatorios. El prototipo de función de ***time()*** está en <***time.h***> (***ctime()*** en el nuevo estándar de *C++*)

Los valores que ***rand()*** produce directamente siempre están en el rango:

$$0 \leq rand() \leq RAND\_MAX$$

Previamente demostramos cómo escribir una sola instrucción para simular el lanzamiento de un dado mediante la instrucción:

```
cara = 1 + rand() % 6;
```

que siempre le asigna a la variable ***cara*** un entero (al azar) dentro del rango:

$$1 \leq cara \leq 6$$

Note que la amplitud de este rango (es decir, el número de enteros consecutivos dentro del mismo) es *6* y el número inicial del rango es *1*. Haciendo referencia a la instrucción anterior, vemos que la amplitud del rango la determina el número que escala *rand()* con el operador de módulo (es decir, *6*) y el número inicial del rango es igual al número (es decir, *1*) que se suma a *rand() % 6*. Podemos generalizar este resultado como sigue:

n = a + rand() % b;

donde *a* es el *valor de desplazamiento* (que es igual al primer número del rango de enteros consecutivos) y *b* es el *factor de escalamiento* (que es igual a la amplitud del rango de enteros consecutivos deseados).

JUEGO DE AZAR

Uno de los juegos de azar más populares se conoce como ***dados*** y se juega en los casinos y en los callejones de todo el mundo. Las reglas del juego son sencillas:

*Un jugador lanza dos dados. Cada dado tiene seis caras. Estas caras tienen 1, 2, 3, 4, 5 y 6 puntos. Una vez que los dados se detienen, se calcula la suma de los puntos en las dos caras superiores. Si la suma es 7 u 11 en el primer tiro, el jugador gana, si es 2, 3 o 12 en el primer tiro, el jugador **pierde** (es decir, la casa gana) Si la suma es 4, 5, 6, 8, 9 o 10 en el primer tiro, entonces la suma se vuelve el **punto** a lograr del jugador. Para ganar, el jugador debe continuar lanzando los dados hasta que llegue a su **punto**. El jugador **pierde** al lanzar un 7 antes de llegar al **punto**.*

Ejemplo 13.4

El siguiente programa: ***JUEDADOS.CPP***, simula el juego de **dados**.

```
/* El siguiente programa: JUEDADOS.CPP, simula el juego de dados. */

#include <iostream.h>          //Para cout y cin
#include <stdlib.h>            //Para rand() y srand()
#include <time.h>              //Para time()

int lanzamiento(void);         //Prototipo de función

void main(void)
        {
                enum estado {Continuar, Ganar, Perder};
                estado estadoJuego;
                int suma, miPunto;

                srand(time(NULL));
                suma = lanzamiento();          //Primer lanzamiento de los dados

                switch(suma)
                        {
                                case 7:
                                case 11:       //Gana al primer lanzamiento
                                        estadoJuego = Ganar;
                                        break;

                                case 2:
                                case 3:
                                case 12:       //Pierde al primer lanzamiento
                                        estadoJuego = Perder;
                                        break;
```

```
                    default:        //Conserva el punto
                            estadoJuego = Continuar;
                            miPunto = suma;
                            cout << endl << "El puntaje es " << miPunto << endl << endl;
                            break;  //Este enunciado es opcional
                }//Fin de switch

            while(estadoJuego == Continuar) //Sigue lanzando
                {
                    suma = lanzamiento();

                    if(suma == miPunto)         //Gana al llegar al punto
                        estadoJuego = Ganar;
                    else
                        if(suma == 7)           //Pierde al tirar 7
                            estadoJuego = Perder;
                }//Fin de while()

            if(estadoJuego == Ganar)
                cout << endl << "El jugador gana" << endl;
            else
                cout << endl << "El jugador pierde" << endl;
        }//Fin de main()

int lanzamiento(void)
        {
            int dado1, dado2, suma;

            dado1 = 1 + rand() % 6;
            dado2 = 1 + rand() % 6;
            suma = dado1 + dado2;

            cout << "Tirada del jugador " << dado1 << " + " << dado2
                 << " = " << suma << endl;

            return suma;
        }
```

Note que el jugador debe tirar dos dados con el primer tiro y debe hacerlo también con todos los lanzamientos subsecuentes. Definimos una función ***lanzamiento()*** que lanza los dados y calcula e imprime su suma. Dicha función se define una vez, pero se llama desde dos lugares del programa. La función ***lanzamiento()*** devuelve la suma de los dos dados, así que lo correcto es indicar un tipo de devolución ***int*** en el encabezado de la función.

El juego requiere de cierta cantidad de razonamiento. El jugador puede ganar o perder en el primer tiro, o ganar y perder en cualquier tiro subsecuente. La variable ***estadoJuego*** sirve para llevar la cuenta de esto. ***estadoJuego*** se declara como de tipo ***estado***. La línea

enum estado {Continuar, Ganar, Perder}

crea un ***tipo definido por el usuario*** llamado ***enumerado***. Según hemos visto un ***tipo enumerado*** indicado por la palabra clave ***enum*** y seguida por un ***nombre de tipo*** (en este caso ***estado***) es un conjunto de constantes enteras representadas por identificadores. Los valores de estas ***constantes de enumeración*** inician en ***0***, a menos que se indique otra cosa, y se incrementan en ***1***. En la enumeración previa, a ***Continuar***, ***Ganar*** y ***Perder*** se le asigna los valores ***0***, ***1*** y ***2***, respectivamente. Los identificadores de un ***enum*** deben ser únicos, pero las constantes de enumeración separadas pueden tener el mismo valor entero.

A la variable del tipo ***estado*** definido por el usuario sólo se les puede asignar alguno de los tres valores declarados en la enumeración. Cuando se gana un juego, ***estadoJuego*** está establecida a ***Ganar***. Cuando se pierde, se establece a ***Perder***. De otra manera ***estadoJuego*** se establece a ***Continuar***, de modo que se pueden lanzar los dados nuevamente.

Si después del primer lanzamiento se gana el juego, se omite el cuerpo de la estructura ***while***, pues ***estadoJuego*** no es igual que ***Continuar***. El programa procede con la estructura ***if/else***, que imprime ***El jugador gana*** si ***estadoJuego*** es igual a ***Ganar*** y ***El jugador pierde*** si es igual que ***Perder***.

Si después del primer lanzamiento no se termina el juego, ***suma*** se guarda en ***miPunto***. La ejecución procede con la estructura ***while***, pues ***estadoJuego*** es igual a ***Continuar***. Con cada ciclo a través del ***while*** se invoca a ***lanzamiento()*** para generar una ***suma*** nueva. Si ***suma*** es igual que ***miPunto***, ***estadoJuego*** se establece a ***Ganar***, la prueba del ***while*** falla, la estructura ***if/else*** imprime ***El jugador gana*** y la ejecución termina. Si ***suma*** es igual que **7**, ***estadoJuego*** se establece a ***Perder***, la prueba del ***while*** falla, la instrucción ***if/else*** imprime ***El jugador pierde*** y la ejecución termina.

Observe el interesante uso de los distintos mecanismos de control del programa que hemos explicado. El programa del juego de dados utiliza dos funciones (***main()*** y ***lanzamiento()***) y las estructuras ***switch***, ***while***, ***if/else*** e ***if*** anidados.

FUNCIONES DE LA BIBLIOTECA FECHA Y HORA (*time.h*)

LA FUNCIÓN time()

A menudo sus programas requerirán la ***fecha actual*** y la ***hora***. La mayoría de los compiladores ***C++*** proporcionan varias funciones que devuelven la ***fecha*** y la ***hora*** en diferentes formatos. Una de tales funciones es ***time()***, la cual devuelve la ***fecha actual*** y la ***hora*** en segundos a partir del ***00:00*** del ***1*** de ***enero*** de ***1970***. La función devuelve un valor del tipo ***time_t***, como se muestra a continuación:

```
#include <time.h>
time_t  time(time_t  *fechaHora);
```

Si no desea pasar un ***argumento*** a ***time()***, invoque la función con ***NULL***.

```
horaActual = time(NULL);
```

Ejemplo 13.5

El siguiente programa: ***RETARDAR5.CPP***, ilustra el uso de la función ***time()***.

```
/* El siguiente programa: RETARDAR5.CPP, utiliza la función time() para implementar
   un retardo en la ejecución del programa de 5 segundos.
*/

#include <iostream.h>                    //Para cin y cout
#include <time.h>                        //Para time()

void main(void)
        {
                time_t horaActual;
                time_t horaInicial;

                cout << "Retardo de 5 segundos en la ejecución de un programa" << endl;

                time(&horaInicial);      //Hora inicial en segundos.
```

```
        do
                {
                        time(&horaActual);
                }while((horaActual - horaInicial) < 5);

        cout <<"Retardo efectuado" << endl;
}//Fin de main()
```

LA FUNCIÓN *ctime()*

La función ***ctime()***, convierte el formato en segundos de ***time()***, a una ***cadena de caracteres*** con el siguiente formato:

"Sat Jan 08 09:28:36 2000\n"

Ejemplo 13.6

El siguiente programa: ***TIEMPOC.CPP***, ilustra el uso de la función ***ctime()***

```
/* El siguiente programa: TIEMPOC.CPP, ilustra el uso de la función ctime(), para mostrar
   la fecha y hora en el formado: "Sat Jan 08 09:28:36 2000\n"
*/

#include <iostream.h>                   //Para cout y cin
#include <time.h>                       //Para ctime()

void main(void)
        {
                time_t horaActual;

                time(&horaActual);      //Obtiene la hora en segundos.

                cout << "La fecha actual y la hora es : " << ctime(&horaActual);
        }//Fin de main()
```

AJUSTE DE HORARIO (VARIABLE GLOBAL *_daylight*)

Varias de las funciones que se presentarán en esta lección toman en cuenta el ***ajuste de horario***. Para realizar este proceso, varios compiladores ***C++*** declaran una variable global llamada ***_daylight***. Si la variable ***daylight*** contiene un **1** si está activa o ***0*** en caso contrario. Las funciones ***tzset()***, ***localtime()*** y ***ftime()*** modifican el valor de esta variable. El siguiente fragmento de código utiliza la variable ***_daylight*** para determinar si se encuentra o no activa.

```
if(_daylight)
        cout << "El ajuste de horario se encuentra activo";
else
        cout << "El ajuste de horario no se encuentra activo";
```

LA FUNCIÓN *delay()*

Algunos compiladores ***C++*** ofrecen la función ***delay()*** para retardar un cierto número de milisegundos la ejecución de un programa. Su formato es el siguiente:

```
#include <dos.h>
void delay(unsigned miliSegundos);
```

Ejemplo 13.7

El siguiente programa: ***RETARDAR5B.CPP***, ilustra el uso de la función ***delay()***

```
/* El siguiente programa: RETARDAR5B.CPP, utiliza la función delay() para implementar
   un retardo de 5 segundos.

  Nota: Borland C++ v5.02 no contiene la función delay(), en cuyo caso puede usar uno o más ciclos
        for para implementar un retardo.

*/

#include <iostream.h>          //Para cin y cout
#include <dos.h>               //Para delay()

void main(void)
        {
                cout << "Retardo de 5 segundos en la ejecución de un programa" << endl;

                delay(5000);

                cout <<"Retardo efectuado" << endl;
        }//Fin de main()
```

LA FUNCIÓN *clock()*

Cuando elabora las diferentes etapas de su programa, en ocasiones deseará medir la cantidad de tiempo que consumen cada una de ellas. De esta manera puede determinar la parte del programa que está consumiendo más tiempo. Como regla, deberá empezar a optimizar la parte del programa que consume mayor tiempo del procesador. Para ayudarle a determinar el tiempo de proceso de las diferentes partes, ***C++*** proporciona la función ***clock()***, la cual regresa el número de *tictacs* del reloj que requirió la tarea para ejecutar su trabajo. (El número de *tictacs* que existe en un segundo se encuentra en la variable global ***CLK_TCK***) El formato de la función ***clock()*** es el siguiente:

```
#include <time.h>
clock_t clock(void);
```

Como hemos dicho, la función ***clock()*** *regresa* el número de *tictacs* que tardó en realizar un proceso. Para convertir estos *tictacs* a segundos, puede dividir el resultado por la constante ***CLK_TCK***, la cual se encuentra definida en el archivo de cabecera ***time.h***.

Ejemplo 13.8

El siguiente programa: ***RELOJ.CPP***, utiliza la función ***clock()*** para visualizar el tiempo en segundos que duró un proceso.

```
/* El siguiente programa: RELOJ.CPP, ilustra el uso de la función clock. */

#include <iostream.h>          //Para cout y cin
#include <time.h>              //Para clock()
#include <dos.h>               //Para delay()
#include <iomanip.h>           //Para setw()

void main(void)
        {
                time_t horaActual;
                time_t horaInicial;

                cout << "Tiempo consumido de procesador: " << clock() /(long) CLK_TCK << endl << endl;

                //    delay(2000); por no existir esta función lo simularemos con un ciclo do
                 time(&horaInicial);

                do
                        {
                                time(&horaActual);
                        }while((horaActual - horaInicial) < 2);

                cout << "Tiempo consumido de procesador: " << (long)clock() /(long) CLK_TCK << endl << endl;

                //    delay(3000); por no existir esta función lo simularemos con un ciclo do
                time(&horaInicial);

                do
                        {
                                time(&horaActual);
                        }while((horaActual - horaInicial) < 3);

                cout << "Tiempo consumido de procesador: " << (long)clock() /(long) CLK_TCK << endl << endl;
 }//Fin de main()
```

LA FUNCIÓN *difftime()*

La función ***difftime()*** devuelve la diferencia entre dos tiempos como un valor en punto flotante. Su formato es el siguiente:

```
float difftime(time_t horaFinal, time_ t horaInicial);
```

Ejemplo 13.9

El siguiente programa: ***DIFTIEMPO.CPP***, utiliza la función ***difftime()*** para **retardar la ejecución del programa durante 5 segundos.**

```
/* El siguiente programa: DIFTIEMPO.CPP, ilustra el uso de la función difftime */

#include <iostream.h>              //Para cout y cin
#include <time.h>                  //Para difftime()

void main(void)
        {
                time_t horaInicial;
                time_t horaActual;

                time(&horaInicial);

                cout << "Este programa provoca un retardo de 5 segundos" << endl;
                do
                        {
                                time(&horaActual);
                        }while(difftime(horaActual, horaInicial) < 5.0);
        }//Fin de main()
```

LA FUNCIÓN *_strdate()*

Mientras que la función ***ctime()*** requiere que se ejecute la función ***time()***, la función ***_strdate()***, nos da directamente una cadena con la fecha actual. Su formato es el siguiente:

```
#include <time.h>
char *_strdate(char *bufferFecha);
```

El buffer que contendrá la fecha debe ser capaz de recibir ***9*** caracteres (***ocho*** para **mm/dd/aa** y ***uno*** más para el **elemento nulo** de la cadena(**/0**))

Ejemplo 13.10

El siguiente programa: ***FECHACAD.CPP***, utiliza la función ***_strdate()*** para visualizar la fecha actual del sistema:

```
/* El siguiente programa: FECHACAD.CPP, ilustra el uso de la función _strdate() */

#include <iostream.h>          //Para cout y cin
#include <time.h>              //Para _strdate()

void main(void)
        {
                char fecha[9];

                _strdate(fecha);

                cout << "La fecha actual es: " << fecha << endl;
        }//Fin de main()
```

LA FUNCIÓN *_strtime()*

Mientras que la función ***ctime()*** requiere que se ejecute la función ***time()***, la función ***_strtime()***, nos da directamente una cadena con la hora actual. Su formato es el siguiente:

```
#include <time.h>
char *_strtime(char *bufferHora);
```

El buffer que contendrá la hora debe ser capaz de recibir ***9*** caracteres (***ocho*** para **hh:mm:ss** y ***uno*** más para el **elemento nulo** de la cadena(/0)).

Ejemplo 13.11

El siguiente programa: ***TIEMPOCAD.CPP***, utiliza la función ***_strtime()*** para visualizar la hora actual del sistema:

```
/* El siguiente programa: TIEMPOCAD.CPP, ilustra el uso de la función _strtime() */

#include <iostream.h>              //Para cout y cin
#include <time.h>                  //Para _strdate()

void main(void)
        {
                char tiempo[9];

                _strtime(tiempo);

                cout << "La hora actual es: " << tiempo << endl;
        }//Fin de main()
```

EL RELOJ DEL *BIOS*

El ***BIOS*** de toda ***PC*** tiene construido un reloj que da cierto número de ***tictacs*** por segundo. El número de ***tictacs*** por segundo viene dado en la variable global ***CLK_TCK***. Dentro de la memoria, el ***BIOS*** almacena el número de ***tictacs*** que han ocurrido desde medianoche. Anteriormente, muchos programadores utilizaban este reloj para retardar su programa un cierto número de ***tictacs***. La función ***delay()*** ofrece un tiempo más preciso (en milisegundos) El reloj del ***BIOS*** es útil para generar una ***semilla*** que se utilizará en la generación de números aleatorios. Muchos compiladores ***C++*** proporcionan dos funciones que le permiten controlar el reloj del ***BIOS***: ***biostime()*** y ***_bios_timeofday()***

LA FUNCIÓN *biostime()*

La función ***biostime()*** le permite acceder el número de ***tictacs*** que han ocurrido desde medianoche.

El formato de la función ***biostime()*** es el siguiente:

```
#include <bios.h>
long biostime(int operación, long tiempoNuevo);
```

El parámetro ***operación*** le permite especificar si lo que desea es ***leer*** o ***establecer*** la hora del ***BIOS***.

Valor	Significado
0	***Leer** el valor actual del reloj.*
1	***Establecer** el tiempo al nuevo valor definido en tiempoNuevo.*

La función devuelve el número actual de *tictacs*.

LA FUNCIÓN *_bios_timeofsay()*

La función ***_bios_timeofday()*** le permite también ***leer*** o ***establecer*** el reloj del ***BIOS***.

```
#include <bios.h>
long _bios_timeofday(int operación, long *ticTacs);
```

El parámetro ***operación*** especifica si desea ***leer*** o ***establecer*** el reloj del ***BIOS***.

Valor	Significado
_TIME_GETCLOCK	***Leer** el valor actual del reloj.*
_TIME_SETCLOCK	***Establecer** el tiempo al valor en tictacs definido en ticTacs.*

La función ***_bios_timeofday()*** devuelve el valor regresado y almacenado en el registro ***AX*** por ***BIOS***.

Ejemplo 13.12

El siguiente programa: ***TIEMPOBIOS.CPP***, utiliza ambas funciones para leer el número de *tictacs* del ***BIOS***.

```
/* El siguiente programa: TIEMPOBIOS.CPP, ilustra el uso de biostime() y _bios_timeofday() */

#include <iostream.h>
#include <bios.h>
#include <time.h>

void main(void)
        {
                long ticTacs;

                ticTacs = biostime(0, ticTacs);

                cout << "El número de tictacs desde medianoche es        : " << ticTacs << endl;

                _bios_timeofday(_TIME_GETCLOCK, &ticTacs);

                cout << "El número de segundos transcurridos desde medianoche: " << ticTacs /CLK_TCK << endl;
        }//Fin de main()
```

LA FUNCIÓN *localtime()*

La función ***localtime()*** convierte la ***hora*** en ***segundos*** a un tipo de ***estructura tm*** como se muestra a continuación:

```
struct tm
        {
                int tm_sec;         // 0 a 59 segundos
                int tm_min;         // 0 a 59 minutos
                int tm_hour;        // 0 a 24 horas
                int tm_mday;        // 1 a 31 días
                int tm_mon;         // 0 a 11 meses
                int tm_year;        // Año - 1900
                int tm_wday;        // 0 domingo hasta 6 sábado
                int tm_yday;        // 1 a 365 días del año
                int tm_isdst;       // No cero si hay cambio de horario(ahorro de luz del día)
        };
```

El formato de la función ***localtime()*** es el siguiente:

```
#include <time.h>
struct tm *localtime(const time_t *horaSegundos);
```

La función ***localtime()*** utiliza las variables globales ***_timezone*** y ***_daylight*** para ajustar la hora, dependiendo de la zona.

Ejemplo 13.13

El siguiente programa: ***TIEMPOLOCAL.CPP***, ilustra el uso de la función ***localtime()***

```
/* El siguiente programa: TIEMPOLOCAL.CPP, ilustra el uso de la función localtime() */

#include <iostream.h>           //Para cout y cin
#include <time.h>               //Para localtime()

void main(void)
        {
                struct tm *horaActual;

                time_t segundos;
                time(&segundos);

                horaActual = localtime(&segundos);

                cout << "La fecha actual (dd-mm-aa) es: " << horaActual -> tm_mday
                     << "-" << horaActual -> tm_mon + 1 << "-" << horaActual -> tm_year
                     << endl << endl;

                cout << "La hora actual (hh:mm:ss)es: " << horaActual -> tm_hour
                     << ":" << horaActual -> tm_min << ":" << horaActual -> tm_sec << endl;
        }//Fin de main()
```

LA FUNCIÓN *gmtime()*

La función ***gmtime()*** convierte la ***hora*** en ***segundos*** con respecto al meridiano de **Greenwich** a un tipo de ***estructura tm*** como se muestra a continuación:

```
struct tm
        {
                int tm_sec;             // 0 a 59 segundos
                int tm_min;             // 0 a 59 minutos
                int tm_hour;            // 0 a 24 horas
                int tm_mday;            // 1 a 31 días
                int tm_mon;             // 0 a 11 meses
                int tm_year;            // Año - 1900
                int tm_wday;            // 0 domingo hasta 6 sábado
                int tm_yday;            // 1 a 365 días del año
                int tm_isdst;           // No cero si hay cambio de horario(ahorro de luz del día)
        };
```

El formato de la función ***gmtime()*** es el siguiente:

```
#include <time.h>
struct tm *gmtime(const time_t *horaSegundos);
```

La función ***gmtime()*** utiliza las variables globales ***_timezone*** y ***_daylight*** para ajustar la hora, dependiendo de la zona.

Ejemplo 13.14

El siguiente programa: ***TIEMPOMG.CPP***, ilustra el uso de la función ***gmtime()***

```
/* El siguiente programa: TIEMPOMG.CPP, ilustra el uso de la función gmtime()
    para obtener la hora de acuerdo al meridiano de Greenwich
*/

#include <iostream.h>           //Para cout y cin
#include <time.h>               //Para gmtime()

void main(void)
        {
                struct tm *horaActual;

                time_t segundos;
                time(&segundos);

                horaActual = gmtime(&segundos);

                cout << "La fecha actual (dd-mm-aa) es: " << horaActual -> tm_mday
                     << "-" << horaActual -> tm_mon + 1 << "-" << horaActual -> tm_year
                     << endl << endl;

                cout << "La hora actual (hh:mm:ss)es: " << horaActual -> tm_hour
                     << ":" << horaActual -> tm_min << ":" << horaActual -> tm_sec << endl;
        }//Fin de main()
```

LA FUNCIÓN *gettime()*

Si esta utilizando ***DOS***, su programa puede obtener la hora del sistema del ***DOS*** utilizando la función ***gettime()*** La función asigna la ***hora actual*** a una ***estructura de tipo time***, como se muestra a continuación:

```
struct time
        {
                unsigned char ti_min;           //Minutos: 0 a 59
                unsigned char ti_hour;          //Horas: 0 a 23
                unsigned char ti_hund;          //Centésimo de Segundo: 0 a 99
                unsigned char ti_sec;           //Segundos: 0 a 59
        };
```

El formato de la función ***gettime()*** es el siguiente:

```
#include <dos.h>
void gettime(struct time *horaActual);
```

Ejemplo 13.15

El siguiente programa: ***TIEMPODOS.CPP***, ilustra el uso de la función ***gettime()*** para visualizar la hora del sistema.

```
/* El siguiente programa: TIEMPODOS.CPP, ilustra el uso de la función gettime()
   para obtener la hora de sistema operativo DOS.
*/

#include <iostream.h>           //Para cout y cin
#include <dos.h>                //Para gettime()

void main(void)
        {
                struct time horaActual;

                gettime(&horaActual);

                cout << "La hora actual (hh:mm:ss.ss)es: " << int(horaActual.ti_hour)
                     << ":" << int(horaActual.ti_min) << ":" << int(horaActual.ti_sec)
                     << "." << int(horaActual.ti_hund) << endl;
        }//Fin de main()
```

LA FUNCIÓN *_dos_gettime()*

Varios de los compiladores basados en el sistema operativo ***DOS*** proporcionan la función ***_dos_gettime()***, el cual devuelve una ***estructura del tipo dostime_t***, como se muestra a continuación:

```
struct dostime_t
        {
                unsigned char hour;             //Horas: 0 a 23
                unsigned char minute;           //Minutos: 0 a 59
                unsigned char second;           //Segundos: 0 a 59
                unsigned char hsecond;          //Centésimo de Segundo: 0 a 99
        };
```

El formato de la función *_dos_gettime()* es la siguiente:

```
#include <dos.h>
void _dos_gettime(struct dostime_t *horaActual)
```

LA FUNCIÓN getdate()

Si esta utilizando *DOS*, su programa puede obtener la fecha del sistema del *DOS* utilizando la función *getdate()* La función asigna la *hora actual* a una *estructura de tipo date*, como se muestra a continuación:

```
struct date
        {
                int da_year;            //Año actual
                char da_day;            //Día actual: 1 a 31
                char da_mon;            //Mes actual: 1 a 12
        };
```

El formato de la función *getdate()* es el siguiente:

```
#include <dos.h>
void getdate(struct date *fechaActual);
```

Ejemplo 13.16

El siguiente programa *FECHADOS.CPP*, ilustra el uso de la función *getdate()* para visualizar la fecha del sistema.

```
/* El siguiente programa: FECHADOS.CPP, ilustra el uso de la función getdate()
   para obtener la fecha de sistema operativo DOS.
*/

#include <iostream.h>           //Para cout y cin
#include <dos.h>                //Para gedate()

void main(void)
        {
                struct date fechaActual;

                getdate(&fechaActual);

                cout << "La fecha actual (dd-mm-aaaa) es: " << int(fechaActual.da_day)
                     << "-" << int(fechaActual.da_mon) << "-" << (fechaActual.da_year)<< endl;
        }//Fin de main()
```

LA FUNCIÓN *_dos_getdate()*

Varios de los compiladores basados en el sistema operativo *DOS* proporcionan la función *_dos_getdate()*, la cual devuelve una *estructura del tipo dosdate_t*, como se muestra a continuación:

```
struct dosdate_t
        {
                unsigned char day;          //Día: 1 a 31
                unsigned char month;        //Mes: 1 a 12
                unsigned int year;          //Año: 1980-2099
                unsigned char dayofweek;    //0 para domingo hasta 6 para sábado
        };
```

El formato de la función *_dos_getdate()* es la siguiente:

```
#include <dos.h>
void _dos_getdate(struct dosdate_t *fechaActual);
```

LA FUNCIÓN *settime()*

Si esta utilizando el ***DOS***, su programa puede establecer la hora del sistema utilizando la función ***settime()***, tal como si estuviera utilizando el ***comando TIME*** del ***DOS***. Para utilizar ***settime()***, asigna el tiempo a una ***estructura de tipo time*** como se muestra a continuación:

```
struct time
        {
                unsigned char ti_min;       //Minutos: 0 a 59
                unsigned char ti_hour;      //Horas: 0 a 23
                unsigned char ti_hund;      //Centésimo de Segundo: 0 a 99
                unsigned char ti_sec;       //Segundos: 0 a 59
        };
```

El formato de la función ***settime()*** es el siguiente:

```
#include <dos.h>
void settime(struct time *horaActual);
```

Ejemplo 13.17

El siguiente programa: ***DEFHORA.CPP***, utiliza la función ***settime()*** para establecer la hora actual del sistema a 22:47.

```
/* El siguiente programa: DEFHORA.CPP, ilustra el uso de la función settime() */

#include <iostream.h>           //Para cout y cin
#include <dos.h>                //Para settime()

void main(void)
        {
                struct time horaDeseada;

                horaDeseada.ti_hour     = 22;
                horaDeseada.ti_min      = 47;

                settime(&horaDeseada);
        }//Fin de main()
```

LA FUNCIÓN *_dos_settime()*

Varios de los compiladores basados en el sistema operativo ***DOS*** proporcionan la función ***_dos_settime()***, el cual establece la hora del sistema utilizando una ***estructura de tipo dostime_t***, como se muestra a continuación:

```
struct dostime_t
        {
                unsigned char hour;             // 0 a 23
                unsigned char minute;           // 0 a 59
                unsigned char second;           //0 a 59
                unsigned char hsecond;          // 0 a 99
        };
```

El formato de la función ***_dos_settime()*** es la siguiente:

```
#include <dos.h>
void _dos_settime(struct dostime_t *horaActual);
```

LA FUNCIÓN *setdate()*

Si esta utilizando ***DOS***, su programa puede establecer la fecha del sistema del ***DOS*** utilizando la función ***setdate()*** La función asigna la ***fecha actual*** a una ***estructura de tipo date***, como se muestra a continuación:

```
struct date
        {
                int da_year;                    //Año actual
                char da_day;                    //Día actual: 1 a 31
                char da_mon;                    //Mes actual: 1 a 12
        };
```

El formato de la función ***setdate()*** es el siguiente:

```
#include <dos.h>
void setdate(struct date *fechaActual);
```

Ejemplo 13.18

El siguiente programa: ***DEFFECHA.CPP***, ilustra el uso de la función ***setdate()*** para establecer la fecha actual del sistema a ***8 de diciembre de 1999***.

```
/* El siguiente programa: DEFFECHA.CPP, ilustra el uso de la función setdate() */

#include <iostream.h>           //Para cout y cin
#include <dos.h>                //Para setdate()

void main(void)
        {
                struct date fechaDeseada;

                fechaDeseada.da_day = 8;
                fechaDeseada.da_mon = 12;
                fechaDeseada.da_year = 1999;

        setdate(&fechaDeseada);
  }//Fin de main()
```

LA FUNCIÓN *_dos_setdate()*

Varios de los compiladores basados en el sistema operativo ***DOS*** proporcionan la función ***_dos_setdate()***, el cual establece la fecha del sistema utilizando una ***estructura de tipo dosdate_t*** como se muestra a continuación:

```
struct dosdate_t
        {
                unsigned char day;          //Día: 1 a 31
                unsigned char month;        //Mes: 1 a 12
                unsigned int year;          //Año: 1980-2099
                unsigned char dayofweek;    //0 para domingo hasta 6 para sábado
        };
```

El formato de la función ***_dos_setdate()*** es el siguiente:

```
#include <dos.h>
unsigned _dos_setdate(struct dosdate_t *fecha):
```

LA FUNCIÓN *dostounix()*

Si desea convertir una fecha y hora del estilo ***DOS*** al formato ***UNÍX***, puede utilizar la función ***dostounix()*** Esta función convierte ***estructuras del tipo fecha y tiempo*** a segundos desde la medianoche del ***1 de enero de 1970***, como se muestra a continuación:

```
#include <dos.h>
long dostounix(struct date *fechaDOS);
```

Ejemplo 13.19

El siguiente programa: ***DOSUNIX.CPP***, utiliza la función ***dostounix()*** para convertir la **fecha** y **hora** actual del sistema al formato correspondiente en ***UNÍX***.

```
/* El siguiente programa: DOSUNIX.CPP, ilustra el uso de la función dostounix() */

#include <iostream.h>           //Para cout y cin
#include <dos.h>                //Para dostounix()
#include <time.h>               // Para getdate() y gettime()

void main(void)
        {
                struct time horaDos;
                struct date fechaDos;

                time_t formatoUnix;
                struct tm *local;

                getdate(&fechaDos);
                gettime(&horaDos);

                formatoUnix = dostounix(&fechaDos, &horaDos);
                local = localtime(&formatoUnix);
                cout << "hora UNIX: " << asctime(local);
        }//Fin de main()
```

LA FUNCIÓN *tzset()*

Para determinar la información sobre la zona horaria, se utiliza la función ***tzset()***, la cual tiene el siguiente formato:

```
#include <time.h>
void tzset(void);
```

La función ***tzset()*** utiliza las entradas ***TZ*** del ambiente de la computadora para determinar la información acerca de la zona. Esta función también asigna valores apropiados a las variables globales ***_timezone***, ***_daylight*** y ***_tzname***.

Para determinar la diferencia en segundos entre la hora de ***GREENWICH*** y la hora local, ***C++*** proporciona la función ***tzset()*** (utiliza la entrada ***TZ*** del ambiente ***DOS*** para determinar la hora local de la zona) y la variable global ***_timezone***(donde se encuentra la diferencia de horarios en segundos)

Ejemplo 13.20

El siguiente programa: ***TIEMPOZONA.CPP***, utiliza la función ***tzset()*** y la variable global ***_timezone*** para visualizar la diferencia de horarios en la hora local y la de **Greenwich**, expresado en horas.

```
/* El siguiente programa: TIEMPOZONA.CPP, ilustra el uso de la variable global _timezone
   para determinar la diferencia de horario local con respecto a Greenwich en segundos.
*/

#include <iostream.h>          //Para cout y cin
#include <time.h>              //Para tzset()

void main(void)
        {
                tzset();
                cout << "La diferencia de horas entre la hora local y la de GMT es de: "
                     << _timezone/3600 << endl;
        }//Fin de main()
```

Para ayudar en sus programas a determinar la hora actual de la zona, muchos compiladores proporcionan la función ***tzset()*** que ya hemos antes descrito. La misma activa la variable global ***_tzname***. Esta contiene dos apuntadores: ***_tzname[0]***, la cual apunta a los tres caracteres que indican el nombre de la zona, y ***_tzname[1]*** que apunta a los tres caracteres que indican el nombre del ahorro de luz de la zona.

Ejemplo 13.21

El siguiente programa: ***NOMBRETZ.CPP***, utiliza el variable global ***_tzname*** para visualizar el nombre de la zona y del ahorro de luz.

```
/* Este programa: NOMBRETZ.CPP, muestra el nombre horario de la zona, así como el nombre
    ahorro de luz de la zona.
*/

#include <iostream.h>              //Para cout y cin
#include <time.h>                  //Para tzset()

void main(void)
        {
                tzset();

                cout << "El nombre horario de la zona es: " << _tzname[0] << endl << endl;

                if(_tzname[1])
                        cout << "La zona de ahorro de luz es: " << _tzname[1];
                else
                        cout << "No esta definida la zona de ahorro de luz";
        }//Fin de main()
```

LA ENTRADA *TZ* DEL AMBIENTE *DOS*

El comando ***SET*** del ***DOS*** asigna un valor a la entrada *TZ*. Su formato es el siguiente:

TZ = SSS[+/-]h[h][DDD]

Donde ***SSS*** contiene el nombre horario de la zona (tal como ***EST*** o ***PST***), ***[+/-]h[h]*** especifica la diferencia en horas entre la zona local y la de ***GMT***, y ***DDD*** especifica el nombre de ahorro de luz de la zona (tal como ***PDT***) La entrada siguiente establece la zona horaria para la costa oeste, se encuentra activada el ahorro de luz.

```
C:\> SET TZ=PST8PDT <enter>
```

Cuando no se encuentra activo el ahorro de luz, omita su nombre, tal como se muestra a continuación:

```
C:\> SET TZ=PST8 <enter>
```

Experimente con la entrada de ambiente *TZ* para ver los diferentes nombres en los programas antes elaborados.

Nota: *Si no especifica una entrada TZ, el valor por omisión es EST5EDT*

LA FUNCIÓN putenv()

Se puede definir la variable de medio ambiente *TZ* por programa, mediante la función ***putenv()***

Ejemplo 13.22

El siguiente programa: ***ESTABLETZ.CPP***, utiliza la función ***putenv()*** para establecer la zona horaria. Posteriormente se imprimen estos valores utilizando la variable global ***_tzname[]***

```
/* El siguiente programa: ESTABLETZ.CPP, ilustra el uso de la función putenv() */

#include <iostream.h>     //Para cout y cin
#include <stdlib.h>       //Para putenv()
#include <time.h>                 //Para tzser()

void main(void)
        {
                putenv("TZ=PST8PDT");

                tzset();

                cout << "El nombre horario de la zona es: " << _tzname[0] << endl << endl;

                if(_tzname[1])
                        cout << "El nombre horario ahorro de luz es: " << _tzname[1];
                else
                        cout << "No se definió la zona ahorro de luz";
  }//Fin de main()
```

LA FUNCIÓN ftime()

Una de las funciones más útiles que puede utilizar en sus programas para obtener información acerca de la zona horaria es ***ftime()*** Su formato es el siguiente:

```
#include <sys\timeb.h>
void ftime(struct timeb *horaZona);
```

El parámetro ***horaZona*** de la función ***ftime()*** es un apuntador a una ***estructura de tipo timeb***, como se muestra a continuación:

```
struct timeb
        {
                long time;
                short millitm;
                short timezone;
                short dstflag;
        };
```

El campo ***time*** contiene el número de segundos desde el ***1 de enero de 1970*** (***GMT***) El campo ***millitm*** contiene la parte fraccionaria de segundos expresada en milisegundos. El campo ***timezone*** contiene la diferencia de horas entre la hora local y la de ***Greenwich***. Finalmente, el campo ***dstflag*** especifica si se encuentra o no activo el ahorro de luz.

Ejemplo 13.23

El siguiente programa: ***FTIME.CPP***, utiliza la función ***ftime()*** para visualizar la información referente a la zona horaria.

```
/* El siguiente programa: FTIME.CPP, ilustra el uso de la función ftime() */

#include <iostream.h>              //Para cout y cin
#include <time.h>                  //Para tzset()
#include <sys\timeb.h>             //Para ftime()

void main(void)
        {
                struct timeb horaZona;

                tzset();
                ftime(&horaZona);

                cout << "Número de segundos desde el 1 de enero de 1970 (GMT): "
                     << horaZona.time << endl;

                cout << "Fracción de segundos: " << horaZona.millitm << endl;

                cout << "Diferencia en horas entre GMT y la hora local: "
                     << horaZona.timezone/60 << endl;

                if(horaZona.dstflag)
                        cout << "Se encuentra activo el ahorro de luz";
                else
                        cout << "El ahorro de luz no se encuentra activo";
        }//Fin de main()
```

LA FUNCIÓN stime()

La función ***stime()*** establece la hora del sistema expresado en segundos desde la medianoche del ***1 de enero de 1970***. Su formato es el siguiente:

```
#include <time.h>
int stime(time_t *segundos);
```

La función ***stime()*** siempre devuelve un ***0***.

Ejemplo 13.24

El siguiente programa: ***STIME.CPP***, utiliza la función ***stime()*** para establecer la fecha exactamente un día después (con la misma hora)

```
/* El siguiente programa: STIME.CPP, ilustra el uso de la función stime() */

#include <time.h>                       //Para stime()

void main(void)
        {
                time_t segundos;

                time(&segundos);        //Obtiene la hora actual

                segundos += (time_t)60 * 60 * 24;

                stime(&segundos);
        }//Fin de main()
```

LA FUNCIÓN *mktime()*

La función ***mktime()***, muestra el número de segundos que hay des el ***1 de enero de 1970***, a una fecha propuesta por el programador. Su formato es el siguiente:

```
#include <time.h>
time_t mktime(struct *fechaPropuesta);
```

Si la ***fechaPropuesta*** es válida, la función devuelve el número de segundos. Si ocurre algún error, la función devuelve ***–1***. El parámetro ***fechaPropuesta*** es un apuntador a una ***estructura de tipo tm***, como se muestra a continuación:

```
struct tm
        {
                int tm_sec;
                int tm_min;
                int tm_hour;
                int tm_mday;
                int tm_mon;
                int tm_year;
                int tm_wday;
                int tm_yday;
                int tm_isdat;
        };
```

Ejemplo 13.25

El siguiente programa: ***TIEMPOMK.CPP***, utiliza la función ***mktime()*** para determinar el número de segundos entre la medianoche del ***1 de enero de 1970*** a la medianoche del ***13 de enero de 1999***.

```
/* El siguiente programa: TIEMPOMK.CPP, ilustra el uso de la función mktime() */

#include <iostream.h>          //Para cout y cin
#include <time.h>              //Para mktime()

void main(void)
        {
                time_t segundos;

                struct tm camposHora;

                camposHora.tm_mday = 13;
                camposHora.tm_mon  = 01;
                camposHora.tm_year = 99;
                camposHora.tm_hour = 0;
                camposHora.tm_min  = 0;
                camposHora.tm_sec  = 0;

                segundos = mktime(&camposHora);

                cout << "El número de segundos entre el 13-01-99 y 1-1-70 es de: "
                     << segundos << endl;
        }//Fin de main()
```

Cuando en un programa utiliza la función *mktime()* y no indica todos los campos de la *estructura tm*, el compilador llena los campos restantes. Por ejemplo, el *campo tm_yday*, es llenado por el compilador con la *fecha juliana*.

Ejemplo 13.26

El siguiente programa: ***JULIANA.CPP***, utiliza la función *mktime()* para determinar la *fecha juliana* para el *13 de enero de 1999*.

```
/* El siguiente programa: JULIANA.CPP, determina la fecha juliana para 1l 13-01-99 */

#include <iostream.h>              //Para cout y cin
#include <time.h>                  //Para mktime()

void main(void)
        {
                time_t segundos;

                struct tm camposHora;

                camposHora.tm_mday      = 13;
                camposHora.tm_mon       = 1;
                camposHora.tm_year      = 99;

                if(mktime(&camposHora) == -1)
                        cout << "Error al realizar la conversión de los campos" << endl;
                else
                        cout << "La fecha juliana para el 13-01-99 es: " << camposHora.tm_yday << endl;
  }//Fin de main()
```

LA FUNCIÓN *strftime()*

Como hemos visto, las funciones ***_strdate*** y ***_strtime*** regresan la fecha y hora actual en *formato cadena de caracteres*. Para proporcionarle un mejor control sobre el formato de la fecha y la hora, muchos compiladores proporcionan la función ***strftime()*** El formato de esta función es el siguiente:

```
#include <time.h>
size_t strftime(char *cadena, size_t  longitudMaxima, const char *formato, const struct tm *fechaHora);
```

El parámetro ***cadena*** es la cadena de caracteres formateada. El parámetro ***longitudMaxima*** especifica el número máximo de caracteres que pueden escribirse en la cadena. El parámetro ***formato*** especifica el formato deseado utilizando los caracteres de formato ***%letra***, muy semejante al utilizado por la función ***printf()*** La ***tabla 13.2*** lista los caracteres válidos que puede colocar en el parámetro ***formato***. Finalmente, el parámetro ***fechaHora*** es un apuntador a una *estructura de tipo* ***tm*** que contiene los campos de la fecha y hora. La función regresa un contador con el número de caracteres asignados al *buffer* o **0** si el *buffer* fue sobrecargado.

Especificador del formato	Significado
%%	Carácter %
%a	Día de la semana abreviado
%A	Día de la semana completo
%b	Nombre del mes abreviado
%B	Nombre del mes completo
%c	Fecha y hora
%d	Día del mes con dos dígitos: de 01 al 31
%H	La hora con dos dígitos: del 00 al 23
%I	La hora con dos dígitos: del 01 al 12
%j	Día juliano con tres dígitos
%m	El mes expresado en dígitos: del 1 al 12
%M	Dos dígitos para los minutos: del 00 al 59
%p	Los caracteres AM o PM
%S	Dos dígitos para los segundos: del 00 al 59
%U	Dos dígitos para la semana: del 00 al 53 con domingo como el primer día de la semana
%w	Día de la semana (0 = domingo, 6 = sábado)
%W	Dos dígitos para la semana: del 00 al 53, con lunes como el primer día de la semana.
%x	Fecha
%X	Hora
%y	Dos dígitos para el año: del 00 al 99
%Y	Cuatro dígitos para el año
%Z°	Nombre de la zona horaria

Tabla 13.2. Especificadores de formato para la función strftime()

Ejemplo 13.27

El siguiente programa: ***TIEMPOSTR.CPP***, ilustra el uso de la función ***strftime()***

```
/* El siguiente programa: TIEMPOSTR.CPP, ilustra el uso de la función strftime()
   para formatear la salida de la fecha y hora.
*/

#include <iostream.h>          //Para cout y cin
#include <time.h>              //Para strftime()

void main(void)
        {
                char buffer[128];

                struct tm *fechaHora;

                time_t horaActual;

                tzset();
```

```
        time(&horaActual);
        fechaHora = localtime(&horaActual);

        strftime(buffer, sizeof(buffer), "%x %X", fechaHora);
        cout << "Utilizando %x %X la fecha es: " << buffer << endl;

        strftime(buffer, sizeof(buffer), "%A %B %m, %Y", fechaHora);
        cout << "Utilizando %A %B %m, %Y la fecha es: " << buffer << endl;

        strftime(buffer, sizeof(buffer), "%I:%M%p", fechaHora);
        cout << "Utilizando %I:%M%p la fecha es: " << buffer << endl;
}//Fin de main()
```

TIPOS DE RELOJES DE LA *PC*

Varios de las funciones utilizados en esta lección sirven para darnos la fecha y la hora. Para comprender estas funciones mejor, necesita saber que la ***PC*** utiliza cuatro tipos de relojes: El reloj de la ***CPU***, ***timer***, el reloj en ***tiempo real*** y el reloj ***CMOS***. El reloj de la ***CPU*** controla la velocidad a la que se ejecuta su programa. Cuando un usuario dice que está utilizando un sistema con **500 MHz**, se refiere al reloj de la ***CPU***. El reloj ***timer*** es un *chip* dentro de la ***PC*** que genera una interrupción cierto número de veces por segundo. Cada vez que ocurre un ***tictac***, la ***PC*** genera una interrupción **8**. Mediante la captura de esta interrupción, los programas residentes en memoria pueden activarse en intervalos específicos. El reloj en ***tiempo real*** da la fecha y la hora actual. En la mayoría de los casos el reloj de ***tiempo real*** tiene el mismo valor que el reloj ***CMOS***.

ARCHIVOS DE ENCABEZADO

Cada biblioteca estándar tiene un ***archivo de encabezado*** correspondiente que contiene los prototipos de todas las funciones de dicha biblioteca y las definiciones de varios tipos datos y constantes necesarios para tales funciones. En la ***tabla 13.3*** se listan algunos archivos de encabezado de la biblioteca estándar de ***C++*** que podrían incluirse en los programas en ***C++***

Archivo de encabezado	Explicación
<***assert.h***>	Contiene **macros e información** que agrega diagnósticos para la depuración de programas. La nueva versión de este archivo de encabezado es <***cassert***>.
<***ctype.h***>	Contiene **prototipos de función** que prueban ciertas propiedades de los caracteres y prototipos de función para convertir letras minúsculas a mayúsculas y viceversa. La nueva versión de este archivo de encabezado es <***cctype***>.
<***float.h***>	Contiene los **límites de tamaño** de los números de punto flotante del sistema. La nueva versión de este archivo de encabezado es <***cfloat***>.
<***limits.h***>	Contiene los **límites de tamaño** de los números enteros del sistema. La nueva versión de este archivo de encabezado es <***climits***>.
<***math.h***>	Contiene los **prototipos de las funciones** de la biblioteca matemática. La nueva versión de este archivo de encabezado es <***math***>.
<***stdio.h***>	Contiene los **prototipos de las funciones** de la biblioteca estándar de entrada/salida e información que utilizan. La nueva versión de este archivo de encabezado es <***cstdio***>.
<***stdlib.h***>	Contiene **prototipos de función** para conversión de números a texto, texto a números, asignación de memoria, números aleatorios y varias otras funciones de utilería. La nueva versión de este archivo de encabezado es <***cstdlib***>.

Archivo de encabezado	Explicación
<*string.h*>	Contiene **prototipos de función** para procesamiento de cadenas estilo *C*. La nueva versión de este archivo de encabezado es <***cstring***>.
<*time.h*>	Contiene **prototipos de función** y tipos para manipular la hora y la fecha. La nueva versión de este archivo de encabezado es <***ctime***>.
<*iostream.h*>	Contiene **prototipos de función** de entrada y salida estándar. La nueva versión de este archivo de encabezado es <***iostream***>.
<*iomanip.h*>	Contiene **prototipos de función** para los manipuladores de flujo que permiten formatear flujos de datos. La nueva versión de este archivo de encabezado es <***iomanip***>.
<*fstream.h*>	Contiene **prototipos para funciones** que efectúan entrada y salida de archivos en disco. La nueva versión de este archivo de encabezado es <***fstream***>.

Tabla 13.3. Archivos de encabezado de la biblioteca estándar

El programador puede crear archivos de encabezado personalizados. Los archivos de encabezado definidos por el programador deben terminar con ***.h***. Para incluir un archivo de encabezado definido por el programador, se utiliza la directiva de preprocesador ***#include***. Por ejemplo, el archivo de encabezado ***cuadrado.h*** puede incluirse en nuestro programa por medio de la directiva

#include "cuadrado.h"

al principio del programa.

LO QUE NECESITA SABER

Antes de continuar con la siguiente lección, **asegúrese de haber comprendido los siguientes conceptos:**

- La biblioteca de ***C++*** es una colección de funciones que el compilador le proporciona para utilizarlos en sus programas.
- Cada biblioteca estándar tiene un archivo de encabezado correspondiente, el cual contiene los prototipos de función de todas sus funciones, así como las definiciones de varias constantes simbólicas requeridas por dichas funciones.
- Los programadores pueden y deben crear e incluir sus propios archivos de encabezado.
- Para utilizar una función de una librería en tiempo de ejecución, debe especificar un prototipo de función.
- La mayoría de los compiladores ***C++*** le proporcionan archivos de cabecera que contienen los prototipos de funciones para cada función de las librerías en tiempo de ejecución.
- La función rand() genera un entero entre 0 y RAND_MAX, el cual está definido para que sea cuando menos 32767.
- Los prototipos de función de rand() y srand() están contenidos en <stdlib.h>.
- Los valores generados por rand() pueden ser escalados y desplazados, con el fin de generar valores dentro de un rango específico.
- Para aleatorizar un programa, emplee la función srand() de la biblioteca estándar.
- La instrucción srand() generalmente se inserta en los programas hasta que han sido depurados a fondo. Durante la depuración, es mejor omitir srand() Con esto se asegura la repetición de la información, que es esencial para probar que las correcciones a un programa de generación de números aleatorios funcionan correctamente.

- Para aleatorizar sin necesidad de sembrar una semilla cada vez, podemos indicar srand(time(0)). La función time() normalmente devuelve en segundos la hora calendario. El prototipo de la función time() se encuentra en el encabezado <time.h>
- La ecuación general para escalar y desplazar un número aleatorio es:

$$n = a + rand() \% b$$

donde *a* es el valor de desplazamiento (que es igual al primer número del rango deseado de enteros consecutivos) y *b* es el factor de escala (que es igual a la amplitud del rango deseado de enteros consecutivos).

PREGUNTAS Y PROBLEMAS

PREGUNTAS

1. La función _____ sirve para **generar números aleatorios**.

2. La función _____ sirve para **sembrar el número que servirá como semilla** para generar números aleatorios en un programa.

3. **Escriba** un programa que **pruebe si los ejemplos de las llamadas de funciones de la biblioteca matemática** mostrada en la ***figura 13.1*** efectivamente producen los resultados indicados.

4. **Conteste** las siguientes preguntas.

a. *¿**Qué significa** **seleccionar números aleatorios**?*
b. *¿**Por qué es útil** la función **rand()** para **simular juegos de azar**?*
c. *¿**Por qué necesitaría** aleatorizar un programa mediante **srand()** ¿**Bajo qué circunstancias no es recomendado aleatorizar**?*
d. *¿**Por qué** con frecuencia **es necesario escalar y/o desplazar los valores** generados por **rand()**?*
e. *¿**Por qué** es una técnica útil la **simulación computarizada de situaciones reales**?*

PROBLEMAS

1. **Muestre** el valor de ***x*** tras la ejecución de las siguientes instrucciones:

a) `x = fabs(7.5);`
b) `x = floor(7.5);`
c) `x = fabs(0.0);`
d) `x = ceil(0.0);`
e) `x = fabs(-6.4);`
f) `x = ceil(-6.4);`
g) `x = ceil(-fabs(-8 + floor(-5.5)));`

2. Una aplicación de la función ***floor()*** es redondear un valor a su entero más próximo. La instrucción:

```
y =floor(x + 0.5);
```

redondeará al número ***x*** al entero más próximo y asignará el resultado a ***y***. Escriba un programa que lea varios números y emplee la instrucción previa para redondearlos al entero más próximo. **Por cada número procesado imprima tanto el original como la cifra redondeada**.

3. La función ***floor()*** también sirve para redondear números a cierta posición decimal. La instrucción:

```
y = floor(x * 10 + 0.5) /10;
```

redondea ***x*** a la posición de décimas (la primera posición a la derecha del punto decimal). La instrucción:

```
y = floor(x * 100 + 0.5) / 100;
```

redondea *x* a la posición de centésimas (es decir, la segunda posición a la derecha del punto decimal). **Escriba un programa que defina cuatro funciones que redondeen *x* de las siguientes maneras**:

a. ***redondearEntero(número)***
b. ***redondearDecimas(número)***
c. ***redondearCentecimas(número)***
d. ***redonderMilesimas(número)***

Por cada valor que lea, su programa deberá imprimir la cifra original y el número redondeado al entero más próximo, a la décima más próxima, a la centésima más próxima y a la milésima más próxima.

4. **Escriba** instrucciones que asignen a la variable ***n*** **enteros al azar** dentro de los siguientes rangos:
   a. ***1 <= n <= 2***
   b. ***1 <= n <= 100***
   c. ***0 <= n <= 9***
   d. ***1000 <= n <= 1112***
   e. ***-1 <= n <= 1***
   f. ***-3 <= n <= 11***
5. Para cada uno de los conjuntos de enteros siguientes **escriba** una **sola instrucción** que **imprima** al azar alguna de las cifras del conjunto.
   a. ***2, 4, 6, 8, 10***
   b. ***3, 5, 7, 9, 11***
   c. ***6, 10, 14, 18, 22***
6. **Escriba** un **programa que simule el lanzamiento de una moneda**. Por cada lanzamiento, el programa deberá imprimir ***Cara*** o ***Cruz***. Haga que el programa lance la moneda ***100*** veces y cuente la cantidad de veces que aparece cada lado de la moneda. Imprima el resultado. El programa deberá llamar a una función independiente ***flip()*** que no acepte argumentos y que devuelva ***0*** si es ***cruz*** y ***1*** si es ***cara***. ***Nota:*** si el programa simula de manera realista el lanzamiento de monedas, entonces cada lado debe aparecer aproximadamente la mitad de las veces.
7. Las computadoras juegan cada vez más un papel importante en la educación. **Escriba** un **programa que ayuda a un estudiante de primaria a aprender a multiplicar**. Mediante ***srand()*** genere dos enteros de un dígito. Luego deberá **desplegar** una pregunta como:

   ¿Cuánto es 6 veces 7?

   El estudiante deberá teclear la respuesta. Su programa deberá revisar su respuesta. Si es correcta, imprima ***¡Muy bien!*** y presente otro problema. Si está equivocada, imprima ***No. Por favor trata de nuevo*** y permita que el estudiante responda la misma pregunta hasta que esté correcta.
8. El uso de las computadoras en la educación se conoce como ***CAI*** (Enseñanza Asistida por Computadora) Uno de los problemas que se presenta en los entornos ***CAI*** es la fatiga del estudiante. Esta puede eliminarse cambiando los diálogos de la computadora, a fin de conservar la atención del estudiante. **Modifique** el programa del **ejercicio *7*** para que se impriman distintos comentarios para las respuestas **correctas** e **incorrectas**, según sigue:

   Respuestas **correctas**

   ¡Muy bien!
   ¡Excelente!
   ¡Buen trabajo!
   ¡Sigue haciéndolo bien!

Respuestas **incorrectas**

No. Por favor trata de nuevo.
Incorrecto. Trata una vez más.

¡No te rindas!
No. Trata de nuevo.

Mediante el generador de números aleatorios, selecciones un número entre ***1*** y ***4*** para desplegar un mensaje diferente para cada respuesta. **Presente las respuestas mediante una estructura *switch***.

9. Los sistemas de instrucción asistido por computadora más refinados supervisan el desempeño del estudiante a través de un periodo. La decisión de iniciar un tema nuevo se base con frecuencia en el éxito del estudiante con los temas previos. **Modifique** el programa del **ejercicio *8*** para que cuente la cantidad de respuestas correctas e incorrectas que el estudiante haya introducido. Una vez que el estudiante haya introducido ***10*** **respuestas**, su **programa deberá calcular el porcentaje de respuestas correctas**. Si el porcentaje es menor que ***75%***, el programa deberá imprimir ***Por favor pídele ayuda a tu instructor*** y termina.

10. **Escriba** un programa que juegue ***adivina el número*** como sigue: el programa selecciona el número a adivinar eligiendo al azar un entero entre ***1*** y ***1000***. Luego debe presentar:

Tengo un número entre 1 y 1000.
¿Puede adivinar mi número?
Por favor introduzca su primer intento.

Después el jugador teclea algún número. El programa responde con uno de los siguientes mensajes:

1. ***¡Excelente! ¡Adivinó el número!***
   ¿Desea jugar otra vez (S o N)?
2. ***Muy bajo. Intente de nuevo.***
3. ***Muy alto. Intente de nuevo.***

Si el número que el jugador indicó es incorrecto, el programa deberá entrar en un ciclo hasta que su número sea el correcto. Luego continuará indicándole al jugador ***Muy alto*** o ***Muy bajo*** para ayudarle a dar con el número. ***Nota***: la técnica de búsqueda que se utiliza en este problema se llama búsqueda binaria.

11. **Modifique** el programa del ***ejercicio 10*** de modo que cuente la cantidad de veces que le llevó al usuario adivinar el número. Si el jugador adivina el número en ***10*** intentos o menos, imprime ***¡Sabes el secreto o tuviste mucha suerte!*** Si lo adivina en ***10*** intentos, imprime ***¡Ahah! ¡Sabes el secreto!*** Si lo hace en más de ***10*** intentos, imprime ***¡Puedes hacerlo mejor!***

12. **Modifique** el programa de dados cuya solución dimos en el programa ***JUEDADOS.CPP***, para que permita apuesta. Empaque como función la parte del programa que ejecuta el juego de dados. Inicialice la variable ***capitalApostar*** a ***10000.00*** pesos. Pídale al jugador que indique una apuesta. Utilice un ciclo ***while*** para comprobar que las apuestas introducidas en ***apuesta*** son **menores o iguales** que ***capitalApostar*** y, si no, pídale al usuario que vuelva a introducir ***apuesta*** hasta que su valor sea válido. Tras introducir una apuesta correcta, ejecute un juego de dados. Si el jugador gana, aumente ***capitalApostar*** en la cantidad que contiene ***apuesta*** e imprima el nuevo valor de ***capitalApostar***. Si el jugador pierde, disminuya ***capitalApostar*** en la cantidad que contenga ***apuesta***, imprima el nuevo ***capitalApostar***, revise si ***capitalApostar*** ha llegado a cero y, de ser así, imprima ***Lo siento. ¡Quebraste!.*** A medida que avance el programa, presente distintos mensajes para generar un poco de ***parloteo***, como ***Oh, ¿vas directo a la quiebra, eh?*** o ***¡Vamos, date una oportunidad!*** o ***Suficiente. Es tiempo de retirarte.***