ESTRUCTURAS

- Es una colección de elementos relacionados, posiblemente de diferentes tipos de datos, los cuales tienen un nombre común que los agrupa (nombre de la estructura).

- Son tipos de datos que se generan con combinaciones de otros tipos de datos, incluyendo otras estructuras.

DEFINICION DE UNA ESTRUCTURA

```
struct Time
{
        int hour;
        int minute;
        int second;
};  //end of structure Time
```

donde:

- **Nombre de la estructura:** Time
- **Número de miembros:** 3
- **Nombre de los miembros:** hour, minute, second (dentro de la estructura estos deben ser nombres únicos)
- **Tipo de dato de los miembros:** en este caso, int, pero puede ser cualquier tipo de dato. Una estructura puede tener miembros con diferentes tipos de datos, incluso puede tener un miembro que sea un puntero al mismo tipo de estructura ("self-referential structure")

NOTA:

La estructura anterior NO RESERVA ESPACIO DE MEMORIA, sino que crea un NUEVO TIPO DE DATO que esta ahora disponible para definir variables con él tal y como se hace con variables de otros tipos.

ACCESAR LOS MIEMBROS DE UNA ESTRUCTURA

A) Una vez definida la estructura, hemos generado un nuevo tipo de dato que es una colección de los tipos de datos que conocíamos anteriormente. Para hacer uso de la estructura, debemos usarla para definir variables de este nuevo tipo, como sigue:

1. Time t_object; // Define un objeto o variable de tipo Time, llamado t_object

2. Time t_array[10]; // Define un arreglo de 10 elementos de tipo Time (t_array)

3. Time *t_ptr = &t_object; // Define un puntero (t_ptr) que apunta a un objeto de tipo Time,
   // y que se inicializa con la dirección de t_object

4. Time &t_Ref = t_object; // Define una referencia (o alias) a un objeto de tipo Time
   // (t_Ref), y esta referencia se inicializa con t_object. A partir
   // de ahora nos podemos referir a t_object con su alias t_Ref

B) Para accesar los miembros de una estructura, usamos 2 operadores de acceso a los miembros:

1. **operador punto ("dot operador") (.)**: Accesa la estructura o miembro de la clase usando el nombre de variable para el objeto (t_object) ó la referencia al objeto (t_Ref). (casos 1, 2 y 4 definidos anteriormente). También puede usarse con un puntero (si se usa la sintaxis adecuada). Ejemplos de cómo accesar un miembro de una estructura con los objetos definidos anteriormente:

   t_object.hour

   t_array[0].hour

   t_Ref.hour

   (*t_ptr).hour

2. **operador flecha ("arrow operador") (->)**: Accesa la estructura o miembro de la clase usando un puntero a un objeto (t_ptr) solamente. (caso 3 definido anteriormente). Ejemplo:

   t_ptr -> hour, lo cual equivale a (*t_ptr).hour.

```
// Fig. 6.1: fig06_01.cpp
// Create a structure, set its members, and print it.
#include <iostream>

using std::cout;
using std::endl;

#include <iomanip>

using std::setfill;
using std::setw;

// structure definition
struct Time {
   int hour;     // 0-23 (24-hour clock format)
   int minute;   // 0-59
   int second;   // 0-59
}; // end struct Time

void printUniversal( const Time & );  // prototype
void printStandard( const Time & );   // prototype

int main()
{
   Time dinnerTime;  // variable of new type Time

   dinnerTime.hour = 18;  // set hour member of
                          // dinnerTime
   dinnerTime.minute = 30;  // set minute member
                            // of dinnerTime
   dinnerTime.second = 0;  // set second member
                           // of dinnerTime

   cout << "Dinner will be held at ";
   printUniversal( dinnerTime );
   cout << " universal time,\nwhich is ";
   printStandard( dinnerTime );
   cout << " standard time.\n";

   dinnerTime.hour = 29; // set hour to invalid
                         // value
   dinnerTime.minute = 73;  // set minute to
                            // invalid value

   cout << "\nTime with invalid values: ";
   printUniversal( dinnerTime );
   cout << endl;

   return 0;
} // end main



// print time in universal-time format
void printUniversal( const Time &t )
{
   cout << setfill( '0' ) << setw( 2 ) << t.hour << ":"
        << setw( 2 ) << t.minute << ":"
        << setw( 2 ) << t.second;

} // end function printUniversal



// print time in standard-time format
void printStandard( const Time &t )
{
   cout << ( ( t.hour == 0 || t.hour == 12 ) ?
           12 : t.hour % 12 ) << ":" << setfill( '0' )
        << setw( 2 ) << t.minute << ":"
        << setw( 2 ) << t.second
        << ( t.hour < 12 ? " AM" : " PM" );

} // end function printStandard
```

```
// Fig. 6.1 modified: fig06_01.cpp
// Create a structure, set its members, and print it.
// printUniversal is now using a pointer notation to
// access the structure members.
#include <iostream>

using std::cout;
using std::endl;

#include <iomanip>

using std::setfill;
using std::setw;

// structure definition
struct Time {
   int hour;     // 0-23 (24-hour clock format)
   int minute;   // 0-59
   int second;   // 0-59
}; // end struct Time

void printUniversal( const Time * );  // prototype
void printStandard( const Time & );   // prototype

int main()
{
   Time dinnerTime;         // variable of new type

   // set hour member of dinnerTime
   Time dinnerTime.hour = 18;

   // set minute member of dinnerTime
   dinnerTime.minute = 30;

   // set second member of dinnerTime
   dinnerTime.second = 0;

   cout << "Dinner will be held at ";
   printUniversal( &dinnerTime );
   cout << " universal time,\nwhich is ";
   printStandard( dinnerTime );
   cout << " standard time.\n";

   // set hour to invalid value
   dinnerTime.hour = 29;

   // set minute to invalid value
   dinnerTime.minute = 73;

   cout << "\nTime with invalid values: ";
   printUniversal( &dinnerTime );
   cout << endl;

   return 0;

} // end main


// print time in universal-time format
void printUniversal( const Time *t )
{
   cout << setfill( '0' ) << setw( 2 ) << t->hour
        << ":"<< setw( 2 ) << t->minute << ":"
        << setw( 2 ) << t->second;

} // end function printUniversal

// print time in standard-time format
void printStandard( const Time &t )
{
   cout << ( ( t.hour == 0 || t.hour == 12 ) ?
        12 : t.hour % 12 ) << ":" << setfill( '0' )
      << setw( 2 ) << t.minute << ":"
      << setw( 2 ) << t.second
      << ( t.hour < 12 ? " AM" : " PM" );

} // end function printStandard

/**************************************************
 (C) Copyright 1992-2002 by Deitel &
     Associates, Inc. and Prentice Hall.
     All Rights Reserved.
 **************************************************/

Dinner will be held at 18:30:00 universal time,
which is 6:30:00 PM standard time.

Time with invalid values: 29:73:00
Press any key to continue
```