Capítulo 2

Estructuras de Datos

1. Estructuras

Una estructura puede contener cualquier número de miembros que tendrán un nombre único, además la estructura debe declararse antes de su utilización con el siguiente formato:

```
struct <nombre de la estructura>
{
    <tipo de datos miembro1> <nombre miembro1>
    <tipo de datos miembro2> <nombre miembro2>
    ...
    <tipo de datos miembroN> <nombre miembroN>
};
```

La palabra reservada "struct" indica que estamos declarando una estructura que debemos nombrar, que va a contener N miembros siendo esto un nuevo tipo de dato complejo que al ser definido produce y crea un área en memoria donde los datos se almacenan de acuerdo con el formato declarado. Asimismo, se puede inicializar una estructura de dos formas: Dentro de la sección de código de su programa y como parte de la definición.

El uso de memoria resulta vital para el uso de tipos de datos estructurados, el operador sizeof se puede aplicar para determinar el tamaño que ocupa una estructura. Pudiendo acceder también utilizando el operador punto (.) o el operador flecha (a).

Una estructura se puede pasar como parámetro de una función sin embargo imprimirla es más complejo que una de tipo básico. La solución a esto es la creación de una función para imprimir mediante paso de parámetros por valor y oír referencia. Por último, el concepto de casting básicamente no tiene sentido y por tanto no existe en los datos estructurados por ello es necesario crear un método que transforme de un tipo de estructura a otro.

Siguiendo el concepto de eficiencia desde el punto de vista del uso de memoria existe la opción para los datos estructurados relacionando punteros con estructuras.

2. Uniones

Hemos visto que, en una estructura, cada campo (o miembro) tiene un espacio en memoria para almacenar su valor, pero hay situaciones especiales en la que las estructuras de datos pueden desperdiciar memoria. Para manejar estas situaciones con eficiencia, existen las uniones.

```
Union <nombre de la union>
{
<tipo de datos miembro1> <nombre miembro1>;
<tipo de datos miembro2> <nombre miembro2>;
...
<tipo de datos miembroN> <nombre miembroN>;
};
```

La cantidad de memoria reservada para la unión es igual a la anchura de la variable más grande. En el tipo unión, todos los miembros ocupan la misma posición de memoria, la del miembro de mayor tamaño.

3. Enumeraciones

Existen situaciones en las que es necesario utilizar valores de constantes cuyo valor realmente no importa y lo que importa son los casos que pueden denotar. Por ejemplo, aquellas constantes que denoten el estado civil de una persona: soltero, casado, viudo o divorciado. En este caso, lo importante no es el valor de la constante, sino que el estado sea uno de los valores válidos. Para manejar estas situaciones, existen las enumeraciones.

Las enumeraciones son un tipo de dato estándar utilizado para asignar valores enteros a una colección de nombres siendo por tanto un tipo definido por el usuario con constantes de tipo entero, su principal utilidad es la de contar ordenar y seleccionar variables. Pero a la enumeración se le puede asignar un nombre que se comportara como un nuevo tipo de dato que solo puede contener valores especificados en la enumeración.

Asimismo, en la declaración del tipo enum puede asociarse a los identificadores valores constantes en vez de asociar por defecto con el formato:

```
enum nombre
{
enumerador1 = expresión_constante1,
enumerador2 = expresión_constante2,
...,
enumeradorN = expresión_contanteN
};
```

Los identificadores del enumerado no se pueden introducir ni se pueden escribir en pantalla.

Las enumeraciones facilitan el manejo de los datos ya que es más fácil recordar un nombre que un valor entero. Las declaraciones pueden hacerse dentro o fuera de la función main(). Es habitual encontrar las enumeraciones como herramienta para generar menús.

De esta forma se asigna a cada identificador (que son las opciones para el usuario) un número entero. Las enumeraciones admiten como máximo 256 elementos.

4. Typedef

Con este código podemos usar un tipo enumerado para generar un menú para el usuario. Existe una sola cuestión algo incómoda pero técnicamente correcta. Se trata de los valores que deben tener los enteros de cada elemento de la enumeración. Ya que estamos usando la función getchar() para detectar la opción seleccionada por el usuario del 1 al 4, debemos hacer que el valor de cada elemento de la enumeración sea equivalente al valor del carácter ASCII que el usuario ingresa.

Por ejemplo, el número "1" se encuentra en la posición 49 en la lista de caracteres ASCII. El número "2" es la posición 50, y así sucesivamente. Pero al margen de esta cuestión técnica, veremos cómo los valores del menú tienen su contrapartida en los distintos casos del switch, donde el primer caso es "alta" y no el número 49 (equivalente al "1" en ASCII). La misma lógica aplica al resto de opciones del menú.

Un typedef nos permite crear un determinado alias de un dato definido por el usuario o de un tipo ya existente, de esta forma podemos extender el lenguaje del que partimos con nuevas palabras reservadas. Con esta línea de código usaremos la palabra reservada tipo entero para declarar enteros, pero también podremos crear atajos a tipos de datos compuestos.

```
typedef unsigned int tipoEnteroSinSigno;
```

De igual forma usaremos la nueva palabra para declarar enteros sin signo, su función es similar a la de las macros de tipo define que sustituyen una palabra reservada por un valor, siendo su principal diferencia que las macros son solo reglas de sustitución mientras que typedef hereda todas las características del tipado. Como último punto uno de los usos más comunes es el