UNIVERSIDAD DISTRITAL
FRANCISCO JOSE DE CALDAS

```
struct Complejo
{
        int real;               /* componente real del numero complejo */
        int imaginario;         /* componente imaginario */
} typedef Complejo;
```

Estructuras

2013
Transversal de Programación Básica
Proyecto Curricular de Ingeniería de Sistemas

Objetivos

- Asimilar el concepto de estructura
- Aprender a definir una estructura
- Utilizar estructuras para representar datos complejos

Introducción

En algunos lenguajes es posible agregar tipos de datos definidos por el programador, estos tipos de datos normalmente están constituidos de datos básicos (en el caso de C y C++ **int**, **char**, etc.) y su reunión tiene algún propósito de diseño.

Definición

Una estructura es un conjunto de una o más variables, de distinto tipo, agrupadas bajo un mismo nombre para que su manejo sea más sencillo. Su utilización más habitual es para la programación de bases de datos, ya que están especialmente indicadas para el trabajo con registros o fichas.

La sintaxis de su declaración es la siguiente:

```
struct tipo_estructura
{
    tipo_variable nombre_variable1;
    tipo_variable nombre_variable2;
    tipo_variable nombre_variable3;
};
```

Donde,

- **tipo_estructura** es el nombre del nuevo tipo de dato que hemos creado.
- **tipo_variable** y **nombre_variable** son las variables que forman parte de la estructura.

Para definir variables del tipo que acabamos de crear lo podemos hacer de varias maneras, aunque las dos más utilizadas son éstas:

• Una forma de definir la estructura:

```
struct trabajador
{
    char nombre[20];
    char apellidos[40];
```

```
    int edad;
    char puesto[10];
};
struct trabajador fijo, temporal;
```

- Otra forma:

```
struct trabajador
{
    char nombre[20];
    char apellidos[40];
    int edad;
    char puesto[10];
}
fijo, temporal;
```

En el primer caso declaramos la estructura, y en el momento en que necesitamos las variables, las declaramos. En el segundo las declaramos al mismo tiempo que la estructura. El problema del segundo método es que no podremos declarar más variables de este tipo a lo largo del programa. Para poder declarar una variable de tipo estructura, la estructura tiene que estar declarada previamente. Se debe declarar antes de la función **main**.

El manejo de las estructuras es muy sencillo, así como el acceso a los campos (o variables) de estas estructuras. La forma de acceder a estos campos es la siguiente:

```
variable.campo;
```

Donde **variable** es el nombre de la variable de tipo *estructura* que hemos creado, y **campo** es el nombre de la variable que forma parte de la estructura. Lo veremos mejor con un ejemplo basado en la estructura anterior.

```
temporal.edad=25;
```

Lo que estamos haciendo es almacenar el valor 25 en el campo **edad** de la variable **temporal** de tipo **trabajador**.

Otra característica interesante de las estructuras es que permiten pasar el contenido de una estructura a otra, siempre que sean del mismo tipo naturalmente:

```
fijo=temporal;
```

Al igual que con los otros tipos de datos, también es posible inicializar variables de tipo **estructura** en el momento de su declaración:

struct trabajador fijo={"Pedro","Hernández Suárez", 32, "gerente"};

Si uno de los campos de la estructura es un array de números, los valores de la inicialización deberán ir entre llaves:

```
struct notas
{
   char nombre[30];
   int notas[5];
};
struct notas alumno={"Carlos Pérez",{8,7,9,6,10}};
```

Estructuras y funciones

Podemos enviar una estructura a una función de las dos maneras conocidas:

1. Por valor: su declaración sería:

void visualizar(struct trabajador);

Después declararíamos la variable **fijo** y su llamada sería:

visualizar(fijo);

Por último, el desarrollo de la función sería:

void visualizar(struct trabajador datos)

```
/* Paso de una estructura por valor. */
#include <cstdlib>
#include <iostream>
using namespace std;
struct trabajador
{
   char nombre[20];
   char apellidos[40];
   int edad;
   char puesto[10];
};
void visualizar(struct trabajador);
int main() /* Rellenar y visualizar */
```

```
{
    struct trabajador fijo;
    cout<<("Nombre: ");
    gets(fijo.nombre);
    cout<<("\nApellidos: ");
    gets(fijo.apellidos);
    cout<<("\nEdad: ");
    char temp [10];
    gets(temp);
    fijo.edad = atoi(temp);
    cout<<("\nPuesto: ");
    gets(fijo.puesto);
    visualizar(fijo);
   system("PAUSE");
   return EXIT_SUCCESS;
}
void visualizar(struct trabajador datos)
{
    cout<<"Nombre: "<<datos.nombre;
    cout<<"\nApellidos: "<<datos.apellidos;
    cout<<"\nEdad: "<<datos.edad;
    cout<<"\nPuesto: "<<datos.puesto<<endl;
}
```

2. Por referencia: su declaración sería:

void visualizar(struct trabajador *);

Después declararemos la variable **fijo** y su llamada será:

visualizar(&fijo);

Por último, el desarrollo de la función será:

void visualizar(struct trabajador *datos)

En la función **visualizar**, el acceso a los campos de la variable **datos** se realiza mediante el operador **->**, ya que tratamos con un puntero. En estos casos siempre utilizaremos el operador **->**. Se consigue con el signo *menos* seguido de *mayor que*.

```
#include <cstdlib>
#include <iostream>
using namespace std;
/* Paso de una estructura por referencia. */
struct trabajador
{
    char nombre[20];
    char apellidos[40];
    int edad;
    char puesto[10];
};
void visualizar(struct trabajador *);
main() /* Rellenar y visualizar */
{
    struct trabajador fijo;
    cout<<("Nombre: ");
    gets(fijo.nombre);
    cout<<("\nApellidos: ");
    gets(fijo.apellidos);
    cout<<("\nEdad: ");
    char temp[10];
    gets(temp);
    fijo.edad=(atoi(temp));
    cout<<("\nPuesto: ");
    gets(fijo.puesto);
    visualizar(&fijo);
   system("PAUSE");
   return EXIT_SUCCESS;
}
void visualizar(struct trabajador *datos)
{
    cout<<"Nombre: "<<datos->nombre<<endl;
    cout<<"\nApellidos: "<<datos->apellidos<<endl;
    cout<<"\nEdad: "<<datos->edad<<endl;
    cout<<"\nPuesto: "<<datos->puesto<<endl;
}
```

Arrays de estructuras

Es posible agrupar un conjunto de elementos de tipo estructura en un array. Esto se conoce como *array de estructuras*:

```
struct trabajador
{
    char nombre[20];
    char apellidos[40];
    int edad;
};
struct trabajador fijo[20];
```

Así podremos almacenar los datos de 20 trabajadores.

Ejemplos sobre como acceder a los campos y sus elementos: para ver el nombre del cuarto trabajador, **fijo[3].nombre;**. Para ver la tercera letra del nombre del cuarto trabajador, **fijo[3].nombre[2];**. Para inicializar la variable en el momento de declararla lo haremos de esta manera:

```
struct trabajador fijo[20]={{"José","Herrero Martínez",29},{"Luis","García Sánchez",46}};
```

Typedef

Es posible agrupar un conjunto de elementos de tipo estructura en un array. Esto se conoce como *array de estructuras*: El lenguaje 'C' dispone de una declaración llamada **typedef** que permite la creación de nuevos tipos de datos. Ejemplos:

```
typedef int entero;        /* acabamos de crear un tipo de dato llamado entero */
entero a, b=3;                          /* declaramos dos variables de este tipo */
```

Su empleo con estructuras está especialmente indicado. Se puede hacer de varias formas:

- Una forma de hacerlo:

```
struct trabajador
{
    char nombre[20];
    char apellidos[40];
    int edad;
};
typedef struct trabajador datos;
datos fijo,temporal;
```

- Otra forma:

```
typedef struct
{
    char nombre[20];
    char apellidos[40];
    int edad;
}
datos;
datos fijo,temporal;
```

Ejercicios

1. Cree una estructura que represente un punto geométrico (x, y) y cree una función que permita calcular la distancia entre dos puntos.
2. Escribe un programa utilizando una estructura que lleve control de un inventario de cintas en una tienda de video. Asegúrate de que la estructura incluya el título de la cinta, la longitud de la cinta, el precio de renta, la fecha en que se vendió, y la clasificación de la cinta.
3. ¿Cómo es más conveniente declarar una estructura, en forma global o en forma local? Explique su respuesta.
4. Crear dos estructuras

```
struct distancia{
                 metros
                 centimetros
                }
struct volumen {
                longitud
                anchura
                altura
               }
```

representan las dimensiones de una habitación.
Escribir un programa que lea las dimensiones de una habitación en metros y centímetros y calcule su volumen.

Lectura de Profundización:

- Texto tomado de: http://www.josedomingo.org/web/pluginfile.php/1270/mod_resource/content/0/curso/ESTRUCTURAS.pdf

Imágenes
Las imágenes fueron tomadas de www.google.com

Fuentes:

- http://es.wikiversity.org/wiki/Fundamentos_de_C_-_Lecci%C3%B3n_3
- http://en.wikipedia.org/wiki/Struct_%28C_programming_language%29
- http://www2.ing.puc.cl/~iic11021/materia/misc/struct.htm
- http://www.cs.usfca.edu/~wolber/SoftwareDev/C/CStructs.htm
- http://manual.conitec.net/structs.htm