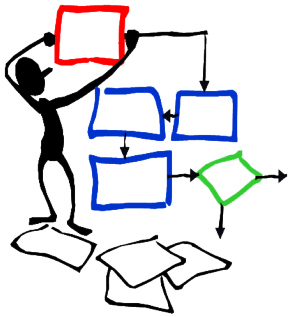
Instituto Politécnico Nacional

Escuela Superior de Cómputo



Algoritmia y programación estructurada

Tema 14: Tipos de datos estructurados en lenguaje C

M. en C. Edgardo Adrián Franco Martínez
http://www.eafranco.com
edfrancom@ipn.mx
@edfrancom edgardoadrianfrancom

Contenido

- Introducción
- Estructuras
- Declaración de una estructura
- Definición de variables de una estructura
- Inicialización de una estructura
- Crear una referencia a una estructura desde el método **`main`**
- Acceso a una estructura
- Estructuras anidadas
- Sinónimo de un tipo de dato

Introducción

- C proporciona cinco diferentes elementos para creación de tipos de datos propios, es decir, tipos de datos que permiten al programador crear aplicaciones más potentes:
  - *Estructuras*: es la agrupación de variables del diferente tipo, bajo un mismo nombre.
  - *Unión*: permite que la misma parte de memoria sea definida como dos o más tipos de variables diferentes.
  - *Campo de bits*: es un tipo especial de *estructura* o *unión* que permite el fácil acceso a bits individuales.
  - *Enumeración*: es una lista de constantes enteras con nombre.

NOTA: `typedef`: ayuda a definir un sinónimo para un tipo de dato ya existente.

Estructuras

- El uso de las estructuras o tipos de datos estructurados, permiten al usuario crear nuevos tipos de datos más versátiles y específicos, para crear aplicaciones más potentes y resolver problemas aun más complejos. E.g., diseñar elementos que contengan registros para una base de datos, creación de datos como: pilas, colas, árboles, etc.

Estructuras

- La potencia de las estructuras radica en que están constituidas por múltiples variables, que pueden ser de diferentes tipos de datos e incluso una estructura puede contener a otras estructuras como tipos de dato.
- Un arreglo es un tipo de dato estructurado, la diferencia entre un arreglo y una estructura radica en que: *un arreglo está constituido por un determinado numero de elementos, todos del mismo tipo, sin embargo, como ya se menciono, una estructura es homogénea, es decir, puede estar constituida por diferentes tipos de datos (datos simples, arreglos, estructuras, etc)*.

Estructuras

- Definición de estructura: ***Una estructura es una colección de variables que se referencian bajo un único nombre, proporcionando un medio conveniente de mantener junta información relacionada***.
- La ***declaración de una estructura*** forma una plantilla que puede utilizarse para crear objetos estructuras, es decir, generar múltiples variables que tengan el cuerpo de la estructura ya definida.
- A las variables que componen o se encuentran dentro de una estructura se les llama ***miembros, elementos o campos*** de la estructura.

Declaración de una estructura

- Como ya se menciono, un estructura es un tipo de dato definido por el programador y al igual que las demás variables se debe declarar antes de utilizarla.

La firma de una estructura es:

```
struct <identificador de la estructura>
{
  <tipo de dato miembro1> <identificador1>
  <tipo de dato miembro2> <identificador2>
  . . .
  <tipo de dato miembron> <identificadorn>
};
```

El fin de la declaración de la estructura debe terminar con punto y coma (;)

Declaración de una estructura

- Por ejemplo si quisiéramos declarar una estructura que almacene los datos de una persona, en este caso seria:

```
struct persona
{
  char nombre[20];
  char domicilio[100];
  int edad;
  char CURP[18];
  long int telefono;
  . . .
};
```

Definición de variables de una estructura

- Hasta el momento se ha declarado el cuerpo o la plantilla de una estructura, para poder utilizarla se debe definir una variable que haga referencia a los elementos de la estructura. Para declarar una variable de la estructura siguen los mismos pasos que cuando se declaran variables de tipos de datos simples, es decir:

```
struct <identificador de la estructura> <identificador de la variable de la estructura>;
```

Definición de variables de una estructura

Para definir una o varias variables de la estructura de tipo persona:

```
struct persona per1, per2, per3;
```

Donde:

per1, per2, per3 son diferentes variables que contienen los mismos elementos definidos en la estructura **persona**.

Otro ejemplo:

```
struct libro
{
  char titulo[200];
  char autor[100];
  char editorial[100];
  char ISBN[100];
  int numPaginas;
  . . .
};

struct libro book1, book2, book3;
```

Inicialización de una estructura

Se puede inicializar una estructura de dos formas:

1. Al momento de definir la plantilla de la estructura, se especifican los valores iniciales entre llaves, por ejemplo:

```
struct libro
{
  char titulo[200];
  char autor[100];
  char editorial[100];
  char ISBN[100];
  int numPaginas;
} book1 = {
    "Manual de referencia en C",
    "Herbert Schildt",
    "Mc Graw Hill",
    "84-481-2895-8",
    709
};
```

2. Dentro de la sección donde se va a utilizar la estructura, es decir, por asignación.

Inicialización de una estructura

Otros ejemplos para inicializar una estructura seria:

Ejemplo 2:

```
struct info_libro
{
  char titulo[200];
  char autor[100];
  char editorial[100];
  int anio;
} book1 = {"Maravilla del saber", "Lucas Garcia", "Mc Graw Hill", 1999};
```

Inicialización de una estructura

Ejemplo 3:

```
struct corredor
{
  char nombre[200];
  int edad;
  char sexo;
  char categoria[100];
  char club[100];
  float tiempoPromedio;
};

struct corredor v1 = {
  "Salvador Hernandez",
  29,
  'H',
  "Senior",
  "Independiente",
   0.0
};
```

También con terminación punto y coma (;)

Crear una referencia a una estructura desde el método **main**

```c
#include <stdio.h>

struct persona{
  char nombre[30];
  int edad;
  float altura;
  float peso;
};

int main( void ){
  struct persona p1;
  printf( "El tamanio de la estructura persona es: %d", sizeof(p1) );
  return 0;
}
```

Se define la variable de la estructura persona dentro de **main**. *Para este caso* **p1** *será una variable local a* **main**

Acceso a una estructura

- Para tener acceso a los elementos de una estructura, bien para modificar o recuperar su valor se utilizan dos operadores:
  1. El operador punto (`.`)
  2. Y el operador flecha (`->`)

Operador punto (.)

El operador punto es la forma más sencilla de tener acceso a los elementos de una estructura, para utilizarlo basta con definir una variable al tipo de estructura y utilizar esta variable para referirnos a los miembros de la estructura anteponiendo el operador punto. La sintaxis de acceso es:

```
<identificador_variable_a_la_estructura>.<nombreMiembro> = dato;

o

dato = <identificador_variable_a_la_estructura>.<nombreMiembro>;
```

El operador punto proporciona el camino directo al miembro correspondiente y los datos que se almacenan en cada miembro deben ser del mismo tipo con el que se declaro cada miembro.

Operador punto (.)

Refiriéndonos nuevamente a la estructura persona

```c
#include <stdio.h>

struct persona{
  char nombre[30];
  int edad;
  float altura;
  float peso;
};

int main( void ){
  struct persona p1;
  strcpy(p1.nombre,"Jacinto Dominguez");
  p1.edad = 25;
  p1.altura = 1.82;
  p1.peso = 76.3;
  printf( "El tamanio de la estructura persona es: %d", sizeof(p1) );
  return 0;
}
```

Se está teniendo acceso ó haciendo referencia a los elementos de la estructura **persona** a través del operador punto

Operador flecha (->)

El operador flecha se utiliza cuando se están manejando apuntadores a una estructura.

Hay que recordar que un apuntador puede hacer referencia a cualquier tipo de dato y una estructura no es la excepción.

Para definir un apuntador a una estructura, la sintaxis es la siguiente:

```
struct <identificador_de_la_estructura>
*<identificador_del_apuntador>;
```

Operador flecha (->)

```c
#include <stdio.h>

struct estudiante
{
  char nombre[30];
  long int matricula;
  float promedio;
  short int numMaterias;
};

int main( void ){
  struct estudiante e1;
  struct estudiante *ptr_e = &e1;

  strcpy(ptr_e->nombre,"Julian Arreaga");
  ptr_e->matricula = 2010568912;
  ptr_e->promedio = 9.06;
  ptr_e->numMaterias = 6;
  return 0;
}
```

Se declara el apuntador a la estructura **estudiante**

Se hace referencia a los elementos de la estructura **estudiante** a través del apuntador **ptr_e** con el operador flecha

Operador flecha (->)

- El operador flecha sirve para tener acceso a los elementos de una estructura a partir de un apuntador.
- El uso más general para utilizar apuntadores a estructuras es cuando se requiere pasar por referencia una estructura a una función o cuando se están trabajando listas, colas o generación de estructuras dinámicas, es decir, uso de *memoria dinámica*.

- Otro aspecto muy importante para pasar por referencia una estructura a una función radica en que si una estructura se pasa por valor (*por copia*) a una función o el tipo de retorno de dicha función es la misma estructura y si esta estructura consta de muchísimos miembros, entonces, copiar todos estos elementos de ida y de regreso en la llamada a la función degradaría mucho la ejecución del programa e incluso a niveles inaceptables, por lo tanto, es más conveniente pasar la referencia a dicha estructura a través de apuntadores, esto conlleva a que la llamada a la función sea muy rápida.

- La mayor diferencia de utilizar el operador punto (.) y el operador flecha (->) para tener acceso a los miembros de una estructura radica en:

  - Cuando se utiliza el operador punto, quiere decir que se están creando variables a estructuras de manera estática, es decir, cuando se declaran dichas variables el compilador les asigna memoria como a cualquier otra variable y esta memoria se mantendrá asignada todo el ciclo de vida de nuestra aplicación.

  - El operador flecha se utilizará mayormente cuando se estén creando estructuras de manera dinámica o cuando se pase por referencia una estructura a una función.

Estructuras anidadas

- Una estructura puede contener variables a otras estructuras llamadas *estructuras anidadas*. Usar estructuras anidadas nos permite ahorrar tiempo en la escritura de programas que utilizan estructuras con información similar, por ejemplo:

```
struct empleado
{
  int ID;
  char nombre[30];
  char direccion[30];
  char ciudad[30];
  long int CP;
  double salario;
};
```

```
struct cliente
{
  int ID;
  char nombre[30];
  char direccion[30];
  char ciudad[30];
  long int CP;
  double credito;
};
```

- Las dos estructuras anteriores contiene información que es muy similar, para ahorrarnos el código repetido, se podría generar una estructura que contenga los datos similares y anidarla dentro de las dos estructuras más generales, es decir:

```
struct info_persona
{
  int ID;
  char nombre[30];
  char direccion[30];
  char ciudad[30];
  long int CP;
};
```

```
struct empleado
{
  struct info_persona info_empleado;
  double salario;
};

struct cliente
{
  struct info_persona info_cliente;
  double credito;
};
```

- Y para tener acceso a cada uno de los elementos de cada estructura sería de la siguiente manera:

```c
#include <stdio.h>

int main( void ){
  struct empleado emp;
  struct cliente cli;
  //Para introducir los datos del empleado
  strcpy(emp.info_empleado.nombre,"Julian Arreaga");
  strcpy(emp.info_empleado.direccion,"Bugambilias #48");
  strcpy(emp.info_empleado.ciudad,"Tlalpan");
  emp.info_empleado.CP = 68643;
  emp.salario = 20345.123;

  //Para introducir los datos del cliente
  strcpy(cli.info_cliente.nombre,"Hernesto Juarez");
  strcpy(cli.info_cliente.direccion,"Matamoros S/N");
  strcpy(cli.info_cliente.ciudad,"Tlalpan");
  cli.info_cliente.CP = 68643;
  cli.credito = 10000000.000;
  return 0;
}
```

Sinónimo de un tipo de dato

- Existe en C una palabra reservada que nos permite definir sinónimos de un tipo de dato ya definido, esta palabra reservada es **`typedef`**.

La firma para utilizar **`typedef`** es:

```
typedef <identificador_datoDefinido> <nuevoNombre>;
```

Así por ejemplo si quisiéramos dar un nuevo nombre a los datos dobles sería:

```
typedef double dobles;
```

y su uso seria:

```
dobles a = 3.13473;
dobles b = 0.47293764;
dobles c = 947282.43232;
```

- Otros ejemplos de **typedef:**

```
typedef char* String;
typedef conts char* string = "Adios a todos…";

String nombre = "Luis Joyanes Aguilar";
```

typedef también se puede utilizar para definir nuevos nombres a una estructura, por ejemplo:

```
struct complejo
{
  float imaginario;
  float real;
};
typedef struct complejo complex;

//definición de variables de estructuras complejo, ya no es necesario
//colocar la palabra struct ya que va implícita en el nombre complex.
complex c1, c2, c3;
```