Estructuras.

Programación I – Laboratorio I.
Tecnicatura Superior en Programación.
UTN-FRA

Autores: *Pablo Gil*
Hector Farina

Revisores: *Ing. Ernesto Gigliotti*
Lic. Mauricio Dávila

Versión : 1

Índice de contenido

1Estructuras.....3
1.1Definición.....3
1.2Declaración de una estructura.....3
1.3Acceso a los campos – Operador punto.....5
1.4Vector de estructuras.....6
1.5Propiedades.....6
1.5.1Asignación 6
1.5.2Comparación.....8

1 Estructuras

Imaginemos que alguien nos pide desarrollar un programa que nos permita administrar los datos de personas, por ejemplo Nombre, apellido, dirección, provincia, localidad, teléfono y fecha de nacimiento.
Con los conocimientos adquiridos hasta el momento, lo primero que se plantea es en qué tipo de variables va a guardar los datos que se ingresen. Puede pensar en principio en 7 arrays, uno para cada campo a guardar.

Las cosas asi planteadas, permiten resolver el problema, pero debido a la gran cantidad de datos a ingresar se torna denso el manejo del código.
Cuando compramos una agenda con índice telefónico y deseamos llenar los datos de personas (nombre, apellido, dirección etc...) no tienemos una hoja para los nombres otra para los apellidos otra para las direcciones, etc. Lo que tenemos es una hoja en la cual guardan todos los datos de una persona, por lo tanto necesitaremos tantas hojas como personas deseeamos ingresar.

En definitiva, la cantidad y tipo de datos son los mismos solo cambia la forma en la cuál se los agrupa. Hasta el momento solo sabemos agrupar datos que sean del mismo tipo (creando un array), la idea es que por una cuestión de practicidad y para facilitar el manejo de gran cantidad de información, se pueda agrupar de alguna forma datos que sean de diferentes tipos.
Esta forma de agrupar datos es conocida en el lenguaje C con el nombre de **estructuras de datos**.

1.1 Definición

Una estructura de datos es un conjunto de variables de distinto tipo a la cuál se hace referencia bajo un mismo nombre.

Siguiendo con el ejemplo mencionado en la introducción, podemos definir una estructura que contenga variables (cadenas) para guardar el nombre, apellido, dirección, localidad , telefono y 3 enteros para guardar el día mes y año de nacimiento. A todo este conjunto de variables lo llamamos agenda y éste será el nombre de referencia con el cuál se conocerá la estructura de datos.
Debemos tener en cuenta que la estructura es un tipo de dato definido por el usuario y por lo tanto puede tener la cantidad de variables que se crean convenientes.
A las variables dentro de la estructura se las suele llamar “miembro de la estructura” o “campo”.

1.2 Declaración de una estructura

Dado que una estructura es una variable definida por el programador, se hace necesario que el compilador conozca el formato que va a tener la o las estructuras que componen el programa. Para esto se describe fuera de todas las funciones y al comienzo del programa cual es el formato de la estructura. La forma genérica es:

```
struct nombre {
       tipo variable_1;
       tipo variable_2;
       // ...
       tipo variable_n;
};
```

Donde *struct* es la palabra reservada que le indica al compilador que se esta declarando una estructura.

Entre llaves se colocan todas las variables que se requieran en la estructura, terminadas por ;
Se debe tener en cuenta que lo que se hizo hasta el momento es solamente definir el formato de la estructura, no se declaró ninguna variable de este tipo.
Cuando se necesita definir una variable estructura para utilizarla dentro del programa, se escribirá lo siguiente:

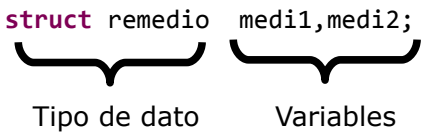
```
struct nombre  var_1,var_2,.............,var_n;
```

Donde "struct nombre" se refiere al formato de estructura y var_1, var_2 son las variables que se pueden usar dentro del programa.

Analicemos un ejemplo más concreto. Se necesitan guardar los datos de los medicamentos de una farmacia como ser nombre del laboratorio, nombre del medicamento y precio. De acuerdo a esto podemos definir una estructura con el siguiente formato:

```
struct remedio {
	char laboratorio[30];
	char nombre[20];
	float precio;
};
```

Cuando tengamos que declarar una variable solo tenemos que escribir:



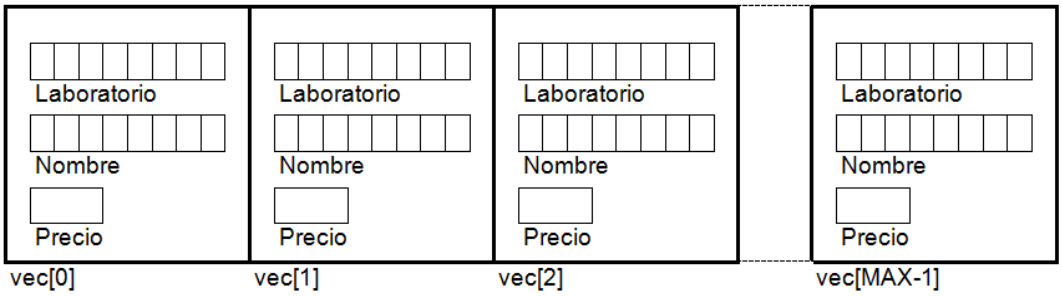
Para aclarar en qué parte del programa se escribe la declaración del formato de la estructura y la variable se verá un ejemplo.

```
#include <stdio.h>

struct remedio {
	char laboratorio[30];
	char nombre[20];
	float precio;
};

void funcion1(int a,float b);

void main(void)
{
	int x,y,z;
	struct remedio medic1,medic2;
	// ...
}
void funcion1(int a,float b)
{
	struct remedio x,vec[10];
	int suma;
	// ...
}
```

A pesar de trabajar con un nuevo tipo de variable (estructura de datos) se siguen manteniendo y aplicando los mismos conceptos que para variables comunes, es decir se pueden tener variables de tipo estructura que sean locales y globales.

En el ejemplo se agregó como variable un vector que es del tipo struct remedio, es decir, se tiene un vector de 10 estructuras.

A dicho vector se lo puede imaginar gráficamente de acuerdo al siguiente diagrama:



1.3 Acceso a los campos – Operador punto

Con la introducción del concepto de estructura lo que se está haciendo es empaquetar datos, pero en definitiva necesitamos cargar las variables que se encuentran dentro de la estructura, para ello se utiliza el operador punto. La forma genérica de usar este operador es la siguiente:

```
Nombre_de_variable_estructura.Nombre_del_campo
```

Si tomamos el ejemplo mencionado anteriormente, donde está la estructura remedio y la variable "medic1" y queremos cargar el precio, se debería escribir de la siguiente forma:

```
medic1.precio=10.25;
strcpy ( medic1.laboratorio , "Bayer" );
strcpy ( medic1.nombre , "Aspirina" );
```

Aquí cargamos la estructura completa asignando a cada campo un valor. En el caso en que se deba ingresar el dato desde el teclado (normalmente se da esta situación) se escribirá lo siguiente:

```
scanf ( "%f" , &medic1.precio );
gets (medic1.laboratorio);
scanf ( "%s" , medic1.nombre);
```

El uso del operador punto nos permite acceder a un campo de la estructura. Observamos que aunque se este trabajando con una estructura, en definitiva llegamos al campo por medio del operador punto y se esta trabajando con variables conocidas y anteriormente usadas. Lo que se pretende remarcar con esto es que la forma de trabajar sigue siendo la misma. Por ejemplo cuando se debia cargar una variable del tipo *float* como precio y se usaba *scanf* también se usaba el "&" precediendo al nombre de la variable, por lo tanto cuando utilicemos estructuras con uno de los campos que sea *float* tambien deberemos usar el "&" precediendo a la variable de la misma forma que se muestra en el ejemplo de arriba.

Cuando se cargaba una cadena de caracteres en un vector, no se usaba "&", por lo tanto cuando se tenga que cargar un vector que se encuentra dentro de una estructura, tampoco se debe usar el "&".

1.4 Vector de estructuras

En la mayoría de los programas que se realicen se va a necesitar ingresar una determinada cantidad de datos. Al igual que ocurría en los programas cuando se ingresaban datos para luego usarlos, se los guardaba en un vector. Con las estructuras ocurre lo mismo, si se necesita ingresar nombre y nota de 20 alumnos se utiliza un modelo de estructura que contenga un campo nombre y un campo nota y luego como variable se utiliza un vector de 20 estructuras para guardar los datos.
Para cargar dicho vector hay que situarse en cada uno de los elementos del vector y recien ahí cargar los campos de la estructura.
El código sería:

```
void main (void)
{
	struct alumno alu[20];
	for(i=0;i<20;i++)
	{
		printf("Ingrese nombre: ");
		gets(alu[i].nombre);
		printf("Ingrese nota: ");
		scanf("%d",&alu[i].nota);
		fflush(stdin);
	}
	// ...
}
```

La forma de cargar el vector es la misma que cuando se carga un vector de enteros, la diferencia es que por cada elemento del vector debo cargar todos los campos de la estructura.

1.5 Propiedades

1.5.1 Asignación

Se puede realizar la asignación de una estructura a otra siempre y cuando las 2 estructuras tengan el mismo formato.

Ejemplo 1
Cargar una variable estructura para luego copiar los datos a una segunda estructura

```
#include <stdio.h>

struct alumno {
	char nombre[20];
	int nota;
};

void main(void)
{
	struct alumno alu1 , alu2;

	printf("Ingrese nombre");
	gets(alu1.nombre);
	printf("Ingrese nota");
	scanf("%d",&alu1.nota);

	alu2=alu1;
}
```

Como se ve en el ejemplo se define una estructura y 2 variables de ese tipo que son alu1 y alu2.
Se carga la variable alu1 y luego se pretende asignar lo que esta cargado en alu1 a alu2. De acuerdo a la propiedad de asignación de estructuras se puede hacer dado que las 2 variables son del mismo tipo de estructura. Lo que realmente pasa es que se copian los datos desde la estructura alu1 hacia alu2 sin necesidad de tener que copiar campo por campo los datos.
Si no existiera esta propiedad se debería hacer lo siguiente:

```
strcpy (alu2.nombre , alu1.nombre);
alu2.nota=alu1.nota;
```

Imaginemos que el caso que estamos tratando no es complicado ya que solo se cuenta con 2 campos, pero si la estructura tuviese más campos, se debería asignar uno por uno y la tarea sería tediosa.

Ejemplo 2
Cargar los datos en una estructura para luego copiar los datos a otra.

```
#include <stdio.h>
#include <string.h>

struct fecha { int dia,mes,anio;};

struct gente {
	char nombre[20];
	struct fecha f_nacimiento;
};

void main(void)
{
	struct gente pers;
	struct fecha fn;

	printf("Ingrese nombre");
	gets(pers.nombre);
	printf("Ingrese dia de nacimiento");
	scanf("%d",&fn.dia);
	printf("Ingrese mes de nacimiento");
	scanf("%d",&fn.mes);
	printf("Ingrese a¤o de nacimiento");
	scanf("%d",&fn.anio);

	pers.f_nacimiento=fn;
}
```

En este ejemplo se trata de ver lo mismo que en el anterior pero usando estructuras anidadas. Se observa que se sigue cumpliendo la propiedad, es posible asignar siempre y cuando las estructuras tienen la misma forma y efectivamente pers.f_nacimiento no es más ni menos que la estructura fecha.

1.5.2 Comparación

No es posible realizar comparación entre 2 estructuras. Lo que se hace es comparar los campos del mismo tipo entre 2 estructuras. Imaginemos el ejemplo dado de asignación. ¿Cómo se hace para comparar las 2 variables, cómo sabe cual es mayor, menor o igual?. Solo se puede comparar entre 2 nombres o 2 notas.

Por lo tanto, se debe tener en cuenta que siempre que necesite realizar comparaciones entre 2 variables estructura será necesario establecer el criterio de comparación.
Todos los operadores relacionales (usados para comparar) que fueron utilizados hasta el momento son aplicables a la comparación entre los campos de la estructura. En definitiva lo que se está haciendo es comparar 2 variables simples, ya que con el operador punto se llega a la variable, ejemplos:

```
if ( alu1.nota == alu2.nota)

if (alu2.nota < alu1.nota )

if ( ¡ strcmp ( alu1.nombre , alu2.nombre))
```

Ejemplo: Se desarrollará el algoritmo para ordenar alfabéticamente un vector de estructuras de MAX elementos en el cuál se van a aplicar las 2 propiedades vistas.

```
#include <stdio.h>
#include <string.h>

#define MAX 10

struct alumno {
	char nombre[20];
	int nota;
};

void main(void)
{
	struct alumno pers[MAX],aux;
	// ...
	for(i=0;i<MAX-1;i++)
		for(j=i+1;j<MAX;j++)
			if((strcmp(pers[i].nombre,pers[j].nombre))<0)
			{
				aux=pers[i];
				pers[i]=pers[j];
				pers[j]=aux;
			}
	// ...
}
```

Notas:

- Los campos de una estructura pueden ser cualquiera de los tipos de variables conocidas (char , int , float , double) y tambien pueden ser vectores , matrices o punteros.

- No se puede tener como estructura anidada a la misma estructura que se esta definiendo