Estructuras

Prof. Andrés Arcia
Departamento de Computación
Escuela de Ingeniería de Sistemas
Facultad de Ingeniería
Universidad de Los Andes

Definición.

Una estructura, o registro como se conoce en lenguajes como Pascal, es un conjunto de variables que pueden ser agrupadas y tratadas como si fuesen una sola. Este conjunto puede estar compuesto de variables de cualquier tipo.

Definición

Imagine que se desea almacenar los datos de un estudiante y sus notas. Todo esto puede ser agrupado en una estructura que podríamos llamar: *DatosEstudiante*, que de forma practica define un nuevo tipo de dato.

Ejemplo

```
Estructura RegistroEstudiante
{
   cadena cedula;
   cadena nombre;
   cadena telefono;
   entero notaPrimerParcial;
   entero notaSegundoParcial;
};
```

Ventajas

Mejor organización del código, más expresividad y en consecuencia un mejor código.

Acorta el tamaño del código para hacer asignaciones de estructuras completas. Ej: estudiante1 = estudiante2. Donde ambas variables son del tipo RegistroEstudiante.

Pueden ser pasadas por parámetro.

Sintaxis Algoritmica

La notación algorítmica de una estructura es:

```
Estructura <nombre_estructura>
{
    tipo1 dato1;
    tipo2 dato2;
    …
    tipon daton;
}
```

Una propiedad interesante del tipo de dato estructura es que pueden anidarse otras “Estructuras” indefinidamente.

Sintaxis C

```
struct punto {
int x;
int y;
};  // observe el punto y coma al final.
```

El nombre de la estructura es “punto” y tiene como miembros un par de variables enteras: “x” e “y”.

Sintaxis C

Podemos declarar una o mas variables del tipo estructura de la siguiente manera:

```
struct { int var1;
         float var2; } i, j, k;
```

Notese que *i, j* y *k* son declaradas análogamente como si lo hiciéramos con cualquiera de los tipos conocidos.

Sintaxis C

Una vez definida la estructura, pueden hacerse sucesivas declaraciones:

```
struct point   w;
```

Y también puede utilizarse la inicialización con llaves:

```
struct point h = {10, 20}; // coordenadas (10,20).
```

Operadores sobre Estructuras

Para trabajar con estructuras hace falta conocer sus operadores.

Operador de acceso a los miembros:

```
nombre_estructura.miembro = valor;
```

Según esta instrucción se esta accediendo a un miembro especifico de una estructura y se está cambiando de valor.

Anidamiento de Estructuras

Las estructuras pueden estar anidadas unas dentro de otras:

```
struct point {
int x;
int y;
};

struct triangle {
struct point p1, p2, p3;
} t;

int main()
{
// En algún lugar dentro del programa

t.p1 = {4, 6};
t.p2.x = 20;
t.p2.y = 32;
t.p3.x = t.p2.x - 10;
t.p3.y = t.p3.y + t.p3.x;

return 0;
}
```

Pase de Estructuras como Parámetros

Las estructuras al igual que las variables simples, pueden ser pasadas por parámetro a cualquier función.

```
struct recta {
float m; //pendiente
float b; // termino independiente
};

float area_triagulo (struct triangle ta)
{
   float base = longitud(ta.p1, ta.p2);
   struct recta r1 = ec_recta_perpendicular(ta.p1, ta.p2);
   struct recta r2 = ec_recta(ta.p1, ta.p2);
   struct point p_i = interseccion(r1, r2);
   float altura = longitud(p_i, p3);

   return (base * altura) / 2;
}
```

Arreglos de Estructuras

Una manera de resolver los arreglos de estructuras es a través de arreglos paralelos.

```
cadena nombres[n];
cadena apellidos[n];
…
entero primerParcial[n];
```

Arreglos de Estructuras

Sin embargo existe una manera más elegante y sencilla de hacerlo:

```
struct RegistroEstudiante listado[n];
```

Para el acceso a cada uno de los miembros se hace:

```
nombre_arreglo[indice].miembro
```

Punteros a Estructuras

Las estructuras también pueden ser trabajadas a través de su dirección de memoria.

```
struct RegistroEstudiante * pre;
struct RegistroEstudiante reg1;

reg1.nombre = "Fernando";
reg1.apellido = "Corbato";
pre = &reg1;

pre->notaPrimerParcial = 20;
// el operado de indirección -> sirve para acceder
// los miembros de una estructura a través de un
// puntero.
```