Repaso de Estructuras

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

Definición.

Una estructura o registro 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, inclusive de otras estructuras. Una estructura, generalmente, define un nuevo tipo de dato.

Definición.

Semánticamente, una estructura agrupa datos que tienen alguna relación según un cierto criterio.

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 práctica define un nuevo tipo de dato.

Ejemplo

```
struct DatosEstudiante
{
   char cedula[12];
   char nombre[32];
   char telefono[10];
   int notaPrimerParcial;
   int notaSegundoParcial;
};
```

Forma de una Estructura

- Qué forma tiene una estructura?
  - Tiene la forma de todos sus componentes pero *serializados*. La serialización consiste en la juntura de todos los contenidos de las variables de un registro, uno detrás del otro.
  - Hay un caso especial que distinguir: cuando hay punteros en la estructura. Ve por qué? Piense en el contenido de una variable puntero.

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 y ocurren copias, inclusive de arreglos, al menos con el compilador de GNU.

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, ta.p3);
   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.
```

Ejemplo de Punteros a Estructura

- Suponga el siguiente caso de una estructura que tiene un apuntador a una estructura de su mismo tipo.

```
struct bloque {
   int a;
   float b;
   struct bloque * s};
```

- Observe que *s* es un apuntador a estructura del tipo *struct bloque*.

Ejemplo de Punteros a Estructura

- Como se obtendrian un par de estructuras enlazadas?

```
struct bloque bl1;

struct bloque bl2;

bl2.s = &bl1;
```

- Luego, que puedo hacer con *bl1* desde *bl2*?

```
bl2.s->a = 145; // a pertenece a bl1

cout << bl1.a; // imprime 145...
```

Estructuras y funciones

```
struct trickyStruct
{ int longArray[10000]; };

void f1(trickyStruct ts);
void f2(trickyStruct *ts);

void f3()
{ struct trickyStruct p;
  // cualquier operacion para llenar p
  f1(p);  // copia todo
  f2(&p); // solamente pasa la direccion, nos ahorramos 9999
   copias...
}
```

Cargando Estructuras en el Heap

- El Heap es una región de memoria que se destina para guardar datos globales o imperecederos durante la vida de un programa.
- Para posicionar variables en el Heap se hace a través del constructo *malloc*.
- La interfaz de *malloc:*
  - *void * malloc(int);*

Ejemplo

```
struct DataInHeap
{ int a;
  float b;
  char c[10]; };

struct DataInHeap * locateInHeap(int _a, float _b, char * _c)
{
   struct DataInHeap * pstr;
   pstr = (struct DataInHeap *)malloc(sizeof(struct DataInHeap));
   if (pstr != NULL)
     {  pstr->a = _a;
        pstr->b = _b;
        strcpy(pstr->c, _c); }
```

Despues de *malloc*

- Despues de haber apartado la memoria del Heap y haberla utilizado, hay que devolverla.
- El constructo para devolver la memoria del Heap es *free.*
- Sintaxis de free:
  - free(void * var);

Ejercicios

- Haga un programa que ordene una lista dinamica de estructuras a través de diferentes campos, por supuesto, uno a la vez.
- Modele:
  - La estructura para un Estudiante.
  - La ecuación de 2do grado.
  - Un laberinto.
  - Un empleado (utilice estructuras anidadas).