Introducción a C

Juan Nehuen Gonzalez Montoro
Pablo Recabarren

Cátedra de Electrónica Digital III

15 de Agosto de 2013

Estructuras

Al contrario que los arrays, las estructuras nos permiten agrupar varios datos, que mantengan algún tipo de relación, aunque sean de distinto tipo, permitiendo manipularlos todos juntos, usando un mismo identificador, o cada uno por separado.

```
1 struct Persona {
2     char Nombre[65];
3     char Direccion[65];
4     int AnyoNacimiento;
5 } Fulanito;
```

Estructuras - Operador "."

El operador punto permite acceder a un atributo de una estructura y la forma de utilizarlo es la siguiente

```
6  struct Persona {
7      char Nombre[65];
8      char Direccion[65];
9      int AnyoNacimiento;
10 } Fulanito;
11
12 int main()
13 {
14         ...
15         ...
16
17         Fulanito.AnyoNacimiento = 2008;
18
19         ...
20 }
```

Punteros

Un puntero es un tipo especial de objeto que contiene la dirección de memoria de un objeto. Almacenada a partir de esa dirección de memoria puede haber cualquier clase de objeto: un char, un int, un float, un array, una estructura, una función u otro puntero. Seremos nosotros los responsables de decidir ese contenido, al declarar el puntero. Los punteros que apunten a tipos de objetos distintos, serán tipos diferentes. Por ejemplo, no podemos asignar a un puntero a char el valor de un puntero a int.

```
21
22 //Declaraci\'on de punteros
23 int *pEntero;
24 char *pCaracter;
25 struct stPunto *pPunto;
26 int* x, y;
```

Punteros

Los punteros apuntan a objetos, por lo tanto, lo primero que tenemos que saber hacer con nuestros punteros es asignarles direcciones de memoria válidas de objetos.
Para averiguar la dirección de memoria de cualquier objeto usaremos el operador de dirección (&), que leeremos como "dirección de".

```
27
28 //inicializar punteros
29 int A;
30 int *pA;
31
32 pA = &A;
```

Punteros

Para manipular el objeto apuntado por un puntero usaremos el operador de indirección, que es un asterisco (*).

```
33
34 int *pEntero;
35 int x = 10;
36 int y;
37
38 pEntero = &y;
39 *pEntero = x;
```

Punteros

Como todos los objetos, los punteros también contienen "basura" cuando son declarados. Es costumbre dar valores iniciales nulos a los punteros que no apuntan a ningún sitio concreto:

```
40 int *pEntero = 0; // Tambi n podemos asignar
       el valor NULL
41 char *pCaracter = 0;
```

Punteros y arrays


```
43 int vector[10];
44 int *puntero;
45
46 puntero = vector; /* Equivale a puntero = &
       vector[0]; (1)
47                    esto se lee como "
                         direcci n del primer
                          elemento de vector"
                      */
48 (*puntero)++;      /* Equivale a vector[0]++;
       (2) */
49 puntero++;         /* puntero equivale a
       asignar a puntero el valor &vector[1] (3)
       */
```

Punteros y estructuras


```
52
53 struct stEstructura {
54     int a, b;
55 } estructura, *e;
56
57 int main() {
58     estructura.a = 10;
59     estructura.b = 32;
60     e = &estructura;
61
62         estructura.a = estructura.a + e->b;
63
64 ...
65 }
```

Operadores de Referencia (&) e Indirección (*)

El operador de referencia (&) nos devuelve la dirección de memoria del operando.

```
66 int *punt;
67 int x = 10;
68
69 punt = &x;
```

El operador de indirección (*) considera a su operando como una dirección y devuelve su contenido.

```
70 int *punt;
71 int x;
72
73 x = *punt;
```

Operadores punto y flecha

permite acceder a objetos o campos dentro de una estructura.
Operador de selección (punto).
Operador de selección de objetos o campos para estructuras referenciadas con punteros(flecha).

```
74 struct punto {
75     int x;
76     int y;
77 };
78
79 punto p1;
80 punto *p2;
81
82 p1.x = 10;
83 p1.y = 20;
84 p2->x = 30;
85 p2->y = 40;
```

Parámetros por valor y por referencia


```
87 int funcion(int n, int &m);
88
89 int main() {
90     int a, b, c;
91     a = 10;
92     b = 20;
93     c = funcion(a,b);
94           ...
95 }
96
97 int funcion(int n, int &m) {
98     n = n + 2;
99     m = m - 5;
100    return n+m;
101 }
```

Parámetros por valor y por referencia


```
103 void funcion(int *q);
104
105 int main() {
106     int a;
107     int *p;
108
109     a = 100;
110     p = &a;
111
112     funcion(p);
113     funcion(&a);
114         ...
115 }
116
117 void funcion(int *q) {
118     *q += 50;
119     q++;
120 }
```

Sobrecarga de funciones


```
122
123 int mayor(int a, int b);
124 char mayor(char a, char b);
125 double mayor(double a, double b);
126
127 int main() {
128         int a;
129         char b;
130         double c;
131
132         b = mayor('a', 'f');
133         a = mayor(15, 35);
134         mayor(10.254, 12.452);
135         ...
136 }
```

Sobrecarga de funciones


```
139 int mayor(int a, int b) {
140     if(a > b) return a; else return b;
141 }
142
143 char mayor(char a, char b) {
144     if(a > b) return a; else return b;
145 }
146
147 double mayor(double a, double b) {
148     if(a > b) return a; else return b;
149 }
```

El preprocesador analiza el fichero fuente antes de la fase de compilación real, y realiza las sustituciones de macros y procesa las directivas del preprocesador. El preprocesador también elimina los comentarios. Una directiva de preprocesador es una lnea cuyo primer carácter es un #.

#define

La directiva #define, sirve para definir macros. Las macros suministran un sistema para la sustitución de palabras, con y sin parámetros.

```
151 #define mult1(a,b) a*b
152 #define mult2(a,b) ((a)*(b))
153 #define PI 3.14159265359
154
155 int main() {
156         int a,b,c,d;
157
158         a = mult1(4,5);
159         b = mult2(4,5);
160
161    // En este caso la primera macro no
162    // funciona, por qu ?: (2)
163     c = mult1(2+2,2+3);
164         d = mult2(2+2,2+3);
```

La directiva include, como ya hemos visto, sirve para insertar ficheros externos dentro de nuestro fichero de código fuente. Estos ficheros son conocidos como ficheros incluidos, ficheros de cabecera o "headers".

```
169 #define FICHERO "trabajo.h"
170
171 #include FICHERO
172
173 int main()
174 {
175 ...
176 }
```