

Tipos de datos definidos por el programador

Antonio Espín Herranz

Tipos de datos definidos por el programador

- Estructuras y Uniones.
- Concepto de estructura.
- Manipulación de estructuras y uniones.
- Acceso a sus elementos.
- Matrices de estructuras y uniones.
- Enumeraciones.

Estructuras

- Estructura: es una colección de uno o mas tipos de elementos denominados miembros, cada uno de los cuales puede ser de un tipo diferente.
- Pueden contener cualquier número de miembros.

```
struct nombre_de_la_estructura {  
    tipo nombre_de_campo1;  
    tipo nombre_de_campo2;  
    tipo nombre_de_campoN;  
}
```

Definición de variables

1ª Forma:

```
struct info_libro {  
    char titulo[60];  
    char autor[30];  
    char editorial[30];  
    int anyo;  
} libro1, libro2, libro3;
```

2ª Forma:

```
struct info_libro libro1, libro2, libro3;
```

Uso de estructuras

- **Asignación:**

```
/* Soporta la asignación múltiple */  
libro1 = libro2 = libro3;
```

- **Inicialización:**

```
struct info_libro {  
    char titulo[60];  
    char autor[30];  
    char editorial[30];  
    int anyo;  
} libro1 = {"Manual C++", "Lucas", "Anaya", 2003};
```

- **Calcular el tamaño de la estructura:**

```
sizeof(libro1)
```

Acceso a estructuras

- **Acceso a los elementos de la estructura (operador .)**

```
struct info_libro libro1;  
strcpy(libro1.titulo, "Manual de C++");  
libro1.anyo = 2003;
```

- **Mediante punteros (operador →)**

```
struct info_libro libro1;  
struct info_libro *plibro1;  
plibro1 = &libro1;  
strcpy(plibro1→titulo, "Manual de C++");
```

Estructuras anidadas

- Ejemplo:

```
struct info_dir {  
    char direccion[25];  
    char ciudad[20];  
    char provincia[20];  
    long int cod_postal;  
}
```

```
struct empleado {  
    char nombre_emp[40];  
    struct info_dir direccion;  
    double salario;  
}
```

```
struct cliente {  
    char nombre_cliente[40];  
    struct info_dir direccion;  
    double saldo;  
}
```

Arrays de estructuras

- Se pueden crear arrays de estructuras como de otros tipos predefinidos.

- Definición:

```
struct info_libro libros[100];
```

- Inicialización:

```
struct info_libros libros[2] =  
{“C++”, “autor”, “Anaya”, 1999,  
“Cobol”, “Autor2”, “Anaya”, 2000};
```


Acceso con el array

```
/* Se indica la posición del array y luego el  
campo al que queremos acceder */
```

```
libros[0].anyo = 2002;
```

```
struct empleado empleados[20];
```

```
/* Por cada nivel de anidación usamos un . */
```

```
empleados[0].direccion.cod_postal = 28015;
```

Estructuras como parámetros

- Igual que el resto de las variables se pueden pasar por valor o por referencia:

```
struct info_libro libro;
```

```
/* Llamada a las funciones */  
leer_libro(&libro); imprime_libro(libro);
```

```
/* Definición de funciones */  
void imprime_libro(struct info_libro libro){  
    printf("titulo %s", libro.titulo);  
    ...  
}  
void leer_libro(struct info_libro *libro){  
    printf("Titulo del libro: ");    gets(libro->titulo);  
    ...  
}
```

Uniones

- Son similares a las estructuras, agrupan una serie de variables, pero la forma de almacenamiento es distinta.
- En la estructura todos los miembros ocupan posiciones contiguas de memoria.
- En las uniones, todos los miembros comparten un área común de memoria. Se reserva la memoria equivalente al campo que más ocupa.

Definición de uniones

```
union nombre_union {  
    tipo1 nombre1;  
    tipo2 nombre2;  
}
```

```
union PruebaUnion {  
    float item1;  
    int item2;  
} var1, *var2;
```

Acceso:

```
var1.item1 = 0.99;  
var2→item2 = 66;
```

Ejemplo: Campos de Bits

```
struct datos {  
    unsigned u0:1;  
    unsigned u1:1;  
    unsigned u2:1;  
    unsigned u3:1;  
    unsigned u4:1;  
    unsigned u5:1;  
    unsigned u6:1;  
    unsigned u7:1;  
};  
  
union v_datos {  
    char car;  
    struct datos d_car;  
};
```

```
v_datos var;  
  
var.car = 'X';  
  
printf("%u%u%u%u%u%u%u%u",  
        var.d_car.u7,  
        var.d_car.u6,  
        var.d_car.u5,  
        var.d_car.u4,  
        var.d_car.u3,  
        var.d_car.u2,  
        var.d_car.u1,  
        var.d_car.u0);
```

Uso de typedef

- Permite asignar sinónimos a los tipos.

- Ejemplo:

```
typedef unsigned char BYTE;  
BYTE byte;
```

```
typedef struct p {  
    char nombre[40];  
    char apellidos[60];  
} REGISTRO;
```

```
REGISTRO persona;
```

Enumeraciones

- **enum** es un tipo de datos definido por el usuario. Son constantes de tipo entero.
- Los valores de la **enum** se asocian con los valores 0, 1, 2, ... Se puede alterar si interesa.

```
enum dias_semana {LUNES, ..., DOMINGO};  
enum dias_semana dia;  
for (dia = LUNES ; dia <= DOMINGO; dia++){  
    ...  
}
```