Unidad 07: Estructura de datos estáticos y almacenamiento lógico

7

Contenidos analíticos

Parte III: Estructura de datos estáticos y almacenamiento logico

Definición de tipo de dato derivado y estructura de dato. Operadores de acceso a cada miembro de la estructura. Estructuras de datos con almacenamiento lógico: Estructuras de posiciones contiguas de memoria de datos no homogéneos y estructuras de datos homogéneos. Combinaciones. Concepto de estructuras estáticas y estructuras con asignación dinámica en memoria. Concepto de estructuras con almacenamiento físico. Profundización de estudio de estructuras de posiciones contiguas no homogéneas en memoria. Introducción al concepto de abstracción de datos

Tipos de Datos

Como ya hemos visto, el objeto de estudio de Algoritmos y Estructura de datos esta centrado básicamente en eso, en los algoritmos, que refiere al "…*conjunto de finito de reglas, ordenadas de forma lógica y precisa para la solución de un problema, con utilización o no de un computador*…" y la estructura de datos. Los tipos de datos Identifican o determinan un dominio de valores y un conjunto de operaciones aplicables sobre esos valores.

1. Primitivos.
2. Derivados.
3. Abstractos.

Los algoritmos operan sobre datos de distinta naturaleza, por lo tanto los programas que implementan dichos algoritmos necesitan una forma de representarlos.
Tipo de dato es una clase de objeto ligado a un conjunto de operaciones para crearlos y manipularlos, un tipo de dato se caracteriza por

1. Un rango de valores posibles.
2. Un conjunto de operaciones realizadas sobre ese tipo.
3. Su representación interna.

Al definir un tipo de dato se esta indicando los valores que pueden tomar sus elementos y las operaciones que pueden hacerse sobre ellos.
Al definir un identificador de un determinado tipo el nombre del identificador indica la localización en memoria, el tipo los valores y operaciones permitidas, y como cada tipo se representa de forma distinta en la computadora los lenguajes de alto nivel hacen abstracción de la representación interna e ignoran los detalles pero interpretan la representación según el tipo.

Tipos de datos pueden ser.

1. **Estáticos:** Ocupan una posición de memoria en el momento de la definición, no la liberan durante el proceso solamente la liberan al finalizar la aplicación.

a. **Simples**: Son indivisibles en datos mas elementales, ocupan una única posición para un único dato de un único tipo por vez.
    i. **Ordinales**: Un tipo de dato es ordinal o esta ordenado discretamente si cada elemento que es parte del tipo tiene un único elemento anterior (salvo el primero) y un único elemento siguiente (salvo el ultimo).
        1. **Enteros**: Es el tipo de dato numérico mas simple.
        2. **Lógico** o booleano: puede tomar valores entre dos posibles: verdadero o falso.
        3. **Carácter**: Proporcionan objetos de la clase de datos que contienen un solo elemento como valor. Este conjunto de elementos esta establecido y normatizado por el estándar ASCII.
    ii. **No ordinales**: No están ordenados discretamente, la implementación es por aproximación
        1. Reales: Es una clase de dato numérico que permite representar números decimales.
b. **Cadenas**: Contienen N-1 caracteres imprimibles y son tratados como una única variable.
c. **Estructuras**: Tienen un único nombre para mas de un dato que puede ser del mismo tipo o de tipo distinto. Permiten acceso a cada dato particular y son divisibles en datos mas elementales.

   Una estructura es, en definitiva, un conjunto de variables no necesariamente del mismo tipo relacionadas entre si de diversas formas.

   Si los datos que la componen son todas del mismo tipo son homogéneas, heterogéneas en caso contrario.

   Una estructura es estática si la cantidad de elementos que contiene es fija, es decir no cambia durante la ejecución del programa
    i. **Registro**: Es un conjunto de valores que tiene las siguientes características:
       Los valores pueden ser de tipo distinto. Es una estructura heterogénea.
       Los valores almacenados se llaman campos, cada uno de ellos tiene un identificador y pueden ser accedidos individualmente.
       El operador de acceso a cada miembro de un registro es l operador punto**.**
       El almacenamiento es fijo.
    ii. **Arreglo**: Colección ordenada e indexada de elementos con las siguientes características:
       Todos los elementos son del mismo tipo, un arreglo es una estructura homogénea.
       Los elementos pueden recuperarse en cualquier orden, simplemente indicando la posición que ocupa dentro de la estructura, esto indica que el arreglo es una estructura indexada.
       El operador de acceso es el operador []
       La memoria ocupada a lo largo de la ejecución del programa es fija, por esto es una estructura estática.
       El nombre del arreglo se socia a un área de memoria fija y consecutiva del tamaño especificado en la declaración.
       El índice debe ser de tipo ordinal. El valor del índice puede verse como el desplazamiento respecto de la posición inicial del arreglo.
       Los arreglos pueden ser de varias dimensiones. Esta dimensión indica la cantidad de índices necesarias para acceder a un elemento del arreglo.
       El arreglo lineal, con un índice, o una dimensión se llama vector.
       El arreglo con 2 o mas índices o dimensiones es una matriz. Un grupo de elementos homogéneo con un orden interno en el que se necesitan 2 o mas índices para referenciar a un elemento de la estructura.
    iii. **Archivos**: Estructura de datos con almacenamiento físico en memoria secundaria o disco.
       Las acciones generales vinculadas con archivos son

Asignar, abrir, crear, cerrar, leer, grabar, Cantidad de elementos, Posición del puntero, Acceder a una posición determinada, marca de final del archivo, definiciones y declaraciones de variables.

Según su organización pueden ser secuenciales, indexados.

1. **Archivos de texto**: Secuencia de líneas compuestas por cero uno o mas caracteres que finalizan con un carácter especial que indica el final de la línea. Los datos internos son representados en caracteres, son mas portables y en general mas extensos.
2. **Archivos de tipo o binarios**: secuencia de bytes en su representación interna sin interpretar. Son reconocidos como iguales si son leídos de la forma en que fueron escritos. Son menos portables y menos extensos.

2. **Combinaciones de estructuras**
   a. struct con un miembro que sea una struct
   b. struct con un miembro que sea un array
   c. struct con miembros que sean array de struct
   d. struct con las combinaciones complejas entre datos simples, cadenas, struct, array, puntero
   e. Array de array→vector de vectores o matrices, de dos o mas dimensiones
   f. Array de una o mas dimensiones con datos de tipo struct con las combinaciones previas
   g. Estructuras enlazadas con cualquiera de las alternativas anteriores
3. **Dinámicos:** Ocupan direcciones de memoria en tiempo de ejecución y se instancian a través de punteros. Estas instancias pueden también liberarse en tiempo de ejecución. El tema de puntadores y estructuras enlazadas (estructuras relacionadas con este tipo de dato se analizan en detalle en capítulos siguientes). En caso de C/C++ haremos referencia a los conceptos de: Constructores (new), con los que se crearan instancias a los punteros en tiempos de ejecución generando lo que denominaremos variables anónimas; destructores (delete) con lo que se liberaran las instancias a punteros generadas en tiempo de ejecución. También abordaremos el tema de operadores: de direccioo (&), de indirección (*) y de acceso a un miembro de una struct apuntada por un puntero: (*puntero). Miembro o el equivalente puntero->miembro. Con esos puntero se pueden generar las estructuras que a continuación se detallan
   a. **Listas simplemente enlazadas**: cada elemento sólo dispone de un puntero, que apuntará al siguiente elemento de la lista o valdrá NULL si es el último elemento.
   b. Pilas: son un tipo especial de lista, conocidas como listas LIFO (Last In, First Out: el último en entrar es el primero en salir). Los elementos se "amontonan" o apilan, de modo que sólo el elemento que está encima de la pila puede ser leído, y sólo pueden añadirse elementos encima de la pila**.**
   c. **Colas**: otro tipo de listas, conocidas como listas FIFO (First In, First Out: El primero en entrar es el primero en salir). Los elementos se almacenan en fila, pero sólo pueden añadirse por un extremo y leerse por el otro.
   d. **Listas circulares**: o listas cerradas, son parecidas a las listas abiertas, pero el último elemento apunta al primero. De hecho, en las listas circulares no puede hablarse de "primero" ni de "último". Cualquier nodo puede ser el nodo de entrada y salida.
   e. **Listas doblemente enlazad**as: cada elemento dispone de dos punteros, uno a punta al siguiente elemento y el otro al elemento anterior. Al contrario que las listas abiertas anteriores, estas listas pueden recorrerse en los dos sentidos.
   f. **Árboles**: cada elemento dispone de dos o más punteros, pero las referencias nunca son a elementos anteriores, de modo que la estructura se ramifica y crece igual que un árbol.
   g. **Árboles binarios**: son árboles donde cada nodo sólo puede apuntar a dos nodos.
   h. **Árboles binarios de búsqueda** (ABB): son árboles binarios ordenados. Desde cada nodo todos los nodos de una rama serán mayores, según la norma que se haya seguido para ordenar el árbol, y los de la otra rama serán menores.
   i. **Árboles AVL**: son también árboles de búsqueda, pero su estructura está más optimizada para reducir los tiempos de búsqueda.

j. **Árboles B**: son estructuras más complejas, aunque también se trata de árboles de búsqueda, están mucho más optimizados que los anteriores.
k. **Tablas HASH**: son estructuras auxiliares para ordenar listas.
l. **Grafos**: es el siguiente nivel de complejidad, podemos considerar estas estructuras como árboles no jerarquizados.
m. **Diccionarios**.

Registros struct en C, C++

Registro: Es un conjunto de valores que tiene las siguientes características:

- Los valores pueden ser de tipo distinto, no necesariamente homogeneos.
- Se define como posiciones contiguas de memoria de tipos no homogéneos. Es, entonces, una estructura heterogénea.
- Los valores almacenados se llaman campos, cada uno de ellos tiene un identificador y pueden ser accedidos individualmente.
- El operador de acceso a cada miembro de un registro es l operador punto (.)
- El almacenamiento es fijo.

Declaracion Genérica

En C, su declaración es:

```
 struct NombreTipo {
        Tipo Identificador;
        Tipo Identificador;
}
struct TipoRegistro {
        int N;
        double Y;
};                          //      declara un tipo
TipoRegistro Registro; //   define una variable
```

Las estructuras pueden ser anidadas. Ejemplo de estructuras anidadas en C

```
struct TipoFecha {
        int     D;
        int     M;
        int     A;
};                          //      declara un tipo fecha

struct TipoAlumno {
        int             Legajo;
        string          Nombre;
        TipoFecha       Fecha
};                          //      declara un tipo Alumno con un campo de tipo Fecha

TipoAlumno Alumno;    //        define un identificador con la estructura declarada.
```

En el caso de la definición precedente, Alumno es un registro (struct para C) con tres miembros (campos) uno de los cuales es un registro (struct) de TipoFecha. El acceso es:

Nombre	Tipo dato	
Alumno	Registro	Registro total del alumno
Alumno.Legajo	Entero	Campo legajo del registro alumno que es un entero
Alumno.Nombre	Cadena	Campo nombre del registro alumno que es una cadena
Alumno.Fecha	Registro	Campo fecha del registro alumno que es un registro
Alumno.Fecha.D	Entero	Campo dia del registro fecha que es un entero
Alumno.Fecha.M	Entero	Campo mes del registro fecha que es un entero
Alumno.fecha.A	Entero	Campo anio del registro alumno que es un entero

Declaraciones de estructura

Designa un tipo y especifica una secuencia de valores variables (denominados “miembros” o “campos” de la estructura) que pueden tener diferentes tipos. Una variable de este tipo de estructura contiene la secuencia completa definida por el tipo. Las estructuras de C son similares a los tipos denominados "registros" en otros lenguajes.

La declaración de un tipo de estructura no reserva espacio para una estructura. Es solo una plantilla para declaraciones posteriores de variables de estructura.

Un miembro no se puede declarar para que tenga el tipo de la estructura en la que aparece. Sin embargo, un miembro puede declararse como un puntero al tipo de estructura en la que aparece siempre y cuando el tipo de estructura tenga una etiqueta. Esto permite crear listas de estructuras vinculadas. A estas estructuras se las denomina estructuras autoreferenciadas

Las estructuras siguen la misma definición de ámbito que otros identificadores.

Los identificadores de estructura deben ser distintos de otras etiquetas de estructura, unión y enumeración con la misma visibilidad.

```
struct a
{
    int x;
    struct b
    {
      int y;
    } var2;
} var1;
```

estas declaraciones son válidas:

```
struct a var3;
struct b var4;
```

Ejemplos

En estos ejemplos se muestran declaraciones de estructura:

CCopiar

```
struct employee   /* Defines a structure variable named temp */
{
    char name[20];
    int id;
    long class;
} temp;
```

La estructura `employee` tiene tres miembros: `name`, `id` y `class`. El miembro `name` es un array de 20 elementos, y `id` y `class` son miembros simples con **`int`** y **`long`**, respectivamente. El identificador `employee` es el identificador de la estructura.

```
struct employee student, faculty, staff;
```

En este ejemplo se definen tres variables de estructura: `student`, `faculty` y `staff`. Cada estructura tiene la misma lista de tres miembros. Los miembros se declaran para que tengan el tipo de estructura `employee`, definido en el ejemplo anterior.

```
struct
{
    float x, y;
} complex;
```

La estructura `complex` tiene dos miembros con el tipo **`float`**, `x` y `y`. El tipo de estructura no tiene ninguna etiqueta y, por tanto, es un tipo sin nombre o anónimo.

```
struct sample
{
    char c;
    float *pf;
    struct sample *next;
} x;
```

Los dos primeros miembros de la estructura son una variable **`char`** y un puntero a un valor **`float`**. El tercer miembro, `next`, se declara como un puntero al tipo de estructura que se va a definir (`sample`).

Las estructuras anónimas pueden ser útiles cuando no se necesita el nombre de etiqueta, como cuando una declaración define todas las instancias de estructura. Por ejemplo:

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

Las estructuras incrustadas suelen ser anónimas.

```
struct somestruct
{
    struct    /* Anonymous structure */
    {
        int x, y;
    } point;
    int type;
} w;
```

Declaracion de union

Especifica un conjunto de valores variable y, opcionalmente, una etiqueta que asigna un nombre a la unión. Los valores de variables se denominan "miembros" de la unión y pueden tener diferentes tipos. Las uniones son similares a los "registros de variante" en otros lenguajes.

Una variable con tipo **`union`** almacena uno de los valores definidos por ese tipo. Las mismas reglas rigen las declaraciones de estructura y de unión. Las uniones también pueden tener campos de bits.

Los miembros de las uniones no pueden tener un tipo incompleto, un tipo **`void`** o un tipo de función. Por lo tanto, los miembros no pueden ser una instancia de la unión pero pueden ser punteros al tipo de unión que se va a declarar.

Una declaración de tipo de unión es solo una plantilla. La memoria no se reserva hasta que se declara la variable.

Nota

Si se declara una unión de dos tipos y se almacena un valor, pero se obtiene acceso a la unión con el otro tipo, los resultados no son confiables. Por ejemplo, se declara una unión de **`float`** y **`int`** . Se almacena un valor **`float`** , pero el programa accede

posteriormente al valor como **int** . En esta situación, el valor dependería del almacenamiento interno de los valores **float** . El valor entero no sería confiable.

Ejemplos

```
union sign
{
    int svar;
    unsigned uvar;
} number;
```

En este ejemplo se define una variable de unión con el tipo `sign` y se declara una variable denominada `number` que tiene dos miembros: `svar`, un entero con signo, y `uvar`, un entero sin signo. Esta declaración permite que el valor actual de `number` se almacene como un valor con signo o sin signo. La etiqueta asociada a este tipo de unión es `sign`.

```
union
{
    struct
    {
      unsigned int icon : 8;
      unsigned color : 4;
    } window1;
    int screenval;
} screen[25][80];
```

La matriz `screen` contiene 2.000 elementos. Cada elemento de la matriz es una unión individual con dos miembros: `window1` y `screenval`. El miembro `window1` es una estructura con dos miembros de campos de bits, `icon` y `color`. El miembro `screenval` es **`int`**. En un momento dado, cada elemento de unión contiene el **`int`** representado por `screenval` o la estructura representada por `window1`.

Las uniones anidadas se pueden declarar de forma anónima cuando son miembros de otra estructura o unión. Aquí se muestra un ejemplo de una unión sin nombre:

```
struct str
{
    int a, b;
    union
    {
      char c[4];
      long l;
      float f;
   };
   char c_array[10];
} my_str;
```

Las uniones se suelen anidar dentro de una estructura que incluye un campo que proporciona el tipo de los datos contenidos en la unión en un momento concreto. Aquí se muestra un ejemplo de una declaración para dicha unión:

```
struct x
{
    int type_tag;
    union
    {
      int x;
      float y;
    }
}
```

Ejercicios con struct

1. Declare una estructura etiquetada estudiante que permita almacenar el nombre, apellido, numero de legajo, curso materia y turno. Seleccione los tipos de datos que considere mas adecuados y justifique su elección
2. Desarrollar una función que, utilizando la estructura previamente declarada permita completar los datos de un estudiante y la retorne
3. Desarrolle un programa en C/C++ que utilizando las declaraciones y definiciones anteriores permita almacenar los datos de un conjunto de estudiantes y mostrarlos en el orden en el que fueron ingresados.
4. Declarar una estructura para un trabajador con su nombre, sueldo, y fecha de ingreso, en una estructura anidada con dia, mes y año.
5. Desarrollar una función que permita cargar y retornar una estructura con los datos de un trabajador
6. Desarrollar un programa que utilizando lo descripto en los puntos 4 y 5 reciba una fecha anterior a la fecha actual y si el trabajador ingreso antes de esa fecha se le otorgue un aumento en su sueldo de 34 por ciento.
7. Declare una estructura persona con un campo de 35 caracteres para el nombre y un campo fecha con una estructura anidada con su fecha de nacimiento
8. Desarrolle una función que cargue los datos de una persona y lo retorne
9. Desarrolle un programa que utilizando lo requerido en los puntos 7 y 8 ingrese la fecha actual y los datos de cinco personas y, en el mismo orden de ingreso, en caso de la fecha corresponder al dia de su cumpleaños muestre su nombre y un mensaje de felicitación