Estructura de Datos y Algoritmos
Departamento de electrónica
Universidad Técnico Federico Santa María

DEPARTAMENTO DE ELECTRONICA

Ayudantía 2: Arreglos, punteros y estructuras

Objetivo Principal: Conocer las ventajas y desventajas, la funcionalidad de los arreglos, punteros y estructuras.
Autor: Kevin Pizarro Aguirre

---

1. Arreglos

Los arreglos son una estructura de dato que consiste en la colección o agrupación de elementos del mismo tipo, donde además la ubicación en memoria para cada elemento es contigua. El acceso a los elementos es a través de un índice, tanto para poder obtener el valor que posee o para darle un nuevo valor. De hecho esa es una de las ventajas de los arreglos, al ser accedidos por índice permite que su tiempo sea constante en dicha operación, es el mejor tiempo o complejidad que podríamos tener en estructuras de datos.

Por otro lado, el tema del uso de memoria contigua puede ser un problema ya que si no existen secciones de memoria contigua que permitan almacenar el arreglo (por fragmentación u otras razones) simplemente no se podrá crear. A diferencia de otras estructuras de datos que no necesariamente serán asignados contiguamente en memoria, permitiendo crear nuevo espacios o eliminarlos sin muchos problemas.

A continuación un ejemplo de uso de arreglos.

```c
#include <stdio.h>
int main(void)
{
    int mi_arreglo [5];//Declaro arreglo de 5 elementos de tipo entero
    mi_arreglo[0] = 1; //Se asigna el valor 1 al elemento del indice 0 de mi_arreglo
    printf("%d.\n", mi_arreglo[4]);//Imprime el valor almacenado en el indice 4
}
```

Notar que en la posición 4 del arreglo no se le ha asignado un valor, pero si se imprimiese qué es lo que tiene almacenado entonces probablemente sea un valor aleatorio. Esto se debe a que es altamente probable que anteriormente esa sección de memoria haya sido ocupada por otro proceso y al liberar la sección de memoria no necesariamente reseteó su valor a 0.

Además, si se imprimieran las direcciones de memoria de todos los elementos del arreglo entonces se notaría la contigüidad de los elementos, separados en X bytes según el tipo de arreglo, expresado en el sistema hexadecimal. Por ejemplo, si el arreglo es de tipo int entonces estarán separados por 4 bytes.

Finalmente cabe mencionar que para crear un arreglo bidimensional basta con declararlo de la siguiente forma, donde el primer índice corresponde a la fila y el segundo a la columna.

```c
int matriz [2][2];
```

2. Punteros

Los punteros sin duda son una de las particularidades de C que son útiles y muchas veces indispensables. Su filosofía viene acompañada del nombre, son aquellos que apuntan a otras variables dentro de nuestro programa, pero ¿qué apuntan?. Apuntan a una dirección de memoria de la variable en cuestión. Esto es sumamente útil cuando se desea modificar "absolutamente" al elemento que está apuntando.

Su creación es a través del tipo al cual se desea apuntar y el operador *. Además es útil el operador & puesto que nos permite obtener la dirección de memoria de la variable a la cual se la apliquemos.

```
1 #include <stdio.h>
2 int main(void)
3 {
4     int numerito = 2; //declaramos una variable entera de valor 2
5     int * punterito;  //declaramos puntero a entero.
6     punterito = & numerito; //apuntamos a la direccion de memoria de la variable
7     *punterito = 3;   //el valor donde apunta el puntero ahora es 3, por ende
      numerito vale 3
8 }
```

Notar en la línea 5, en la declaración del puntero, aún no se le asigna una sección de memoria por lo que podría no estar apuntando a ningún lugar o apuntar a cualquier lugar.

Luego con este simple principio se pueden realizar programas interesantes, nos permiten fundar las bases de muchas estructuras de datos donde cada elemento apunta a otro formando una especie de "cadena" o lista que no necesariamente estará en secciones de memoria contigua. Además, permite fundar las bases del paso por referencia y paso por valor, donde el primero corresponde a pasar un puntero o la dirección de memoria como parámetro una función que definamos y el segundo tal como se ha trabajado hasta el momento, pasando el valor de la variable directamente.

Paso por valor y paso por referencia

```
1  void power(int * base, int * potencia, int * resultado) { //por referencia
2     *resultado = 1;
3     int i;
4     for (i = 1; i <= *potencia; i++){
5          *resultado = (*base) * (*resultado); //Para cada iteracion actualiza el
      valor de resultado
6     }
7  }
8
9  int power2(int base, int potencia){ //por valor
10    int resultado = 1;
11    int i;
12    for (i = 1; i <= potencia; i++){
13         resultado = base * resultado;
14    }
15    return resultado; //Es necesario retornar el valor, de lo contrario se perderia
      el valor de resultado
16 }
```

Si en algún caso se deseara mover el puntero en 1 posición hacia adelante basta con hacer **punterito ++;** (útil para recorrer strings).

3. Estructuras

Las estructuras son una colección o agrupación de variables, las que pueden ser iguales o diferentes, bajo un mismo nombre. En general, son utilizadas para llevar registros ordenados de un suceso o "objeto" en particular. Por ejemplo, en una biblioteca llevar un registro de libros, donde cada libro tiene un ID, un título, un autor y un año de publicación. Además, es la base para muchas estructuras de datos como las listas enlazadas, árboles binarios y tablas hash por mencionar algunas.

Su definición se puede realizar principalmente de dos formas, las veremos a continuación.

Forma 1. A través de **struct** <nombre_estructura>{variables agrupadas}, notando que al crear un libro en base a la "planilla" se realiza con struct Books <nombre_variable>.

```c
struct Books {
    int book_id;
    char title [50];
    char author [50];
    int pub_year;
};
int main(void){
    struct Books Book1;
    Book1.book_id = 1;
    return 0;
}
```

Forma 2. A través de una definición de tipo con **typedef**. Notar que ahora al crear un nuevo libro entonces no es necesario repetir el llamado de struct, basta llamarlo con el alias que se le ha dado (Libros en el ejemplo) puesto que se ha definido un nuevo tipo de variable.

```c
typedef struct Books {
    int book_id;
    char title [50];
    char author [50];
    int pub_year;
}Libros;
int main(void){
    Libros Libro1;
    Libro1.book_id = 1;
    return 0;
}
```

El acceso a los miembros o variables dentro de las estructuras es a través del operador **.**, por otro lado si se tuviera un puntero a una estructura el acceso es a través del operador $->$. Ver siguiente ejemplos del repositorio. Además también se pueden anidar estructuras, es decir, tener una estructura dentro de una estructura tal como si fuera otro tipo de dato.

4. Ejercicios

1. Escriba un programa que le permita encontrar el valor mínimo dentro de un arreglo.

2. Declare un arreglo de entero de tamaño 5. Solicite al usuario que ingrese los valores para cada uno. Finalmente ordene ascendentemente el arreglo. Sin realizar cálculos exactos, ¿cuánto tardaría si fueran 10, 100, 1000 o 10000 elementos en vez de 5?

3. Escriba un programa que calcule el determinante de una matriz 2x2. ¿Cómo lo implementaría para una matriz de 3x3?

4. Usando punteros, escriba un programa que cuente las vocales de un string recibido como entrada. Imprima su resultado. ¿Qué modificaría para que además pueda contar las consonantes?

5. Escriba una función que intercambie los valores de dos variables enteras usando paso por referencia.

6. Usando estructuras, defina un sistema de inventario donde los miembros corresponden al ID, cantidad del objeto, descripción del objeto y el precio del objeto. Defina un arreglo de 10 elementos para almacenar los objetos, una función para añadir un nuevo objeto y eliminar dado un ID.

Nota: Soluciones en el repositorio.