INGENIERIA EN COMPUTACION

Materia: Programación Estructurada

Alumno: Ayon Dimas Andres

Matricula: 366845

Profesor: Pedro Nunes Yepiz

Actividad: #10

Tema: Funciones, métodos de ordenación y búsqueda, estructuras y librerías

Ensenada B.C 17 de octubre del 2023

INTRODUCCION

Visto lo de prácticas anteriores llegamos hasta los **struct,** donde aplicaremos todo lo que usamos anteriormente ya que es esencial para poder realizar esta práctica, donde veremos cómo es que se pueden hacer los struct en c, así mismo que son y para que sirve, si importancia y del por qué se necesita en C. Utilizaremos funciones, ciclos y también nuestras propias librerías, que ya vimos en la práctica anterior como es que se hacían estas mismas.

COMPETENCIA

Es importante que sepamos el uso y la manera en la que se aplican los structs en c, aplicando lo anterior aprendido, como son las sentencias de lo ciclos, vectores, funciones, métodos de ordenación y búsqueda, librerías propias, etc.

Aquí aprenderemos porque son importantes en C y para que nos sirven, donde es importante su comprensión ya que estas nos van a servir para poder llevar registros de cualquier tipo y nosotros podremos buscar algún dato que desdemos para poder llevar ese orden que los structs nos permiten.

FUNDAMENTOS

Funciones en C

Una función es un bloque de código que realiza alguna operación. Una función puede definir opcionalmente parametron de entrada que permiten a los llamadores pasar argumentos a la función. Una función también puede devolver un valor como salida. Las funciones son útiles para encapsular las operaciones comunes en un solo bloque reutilizable, idealmente con un nombre que describa claramente lo que hace la función. La siguiente función acepta dos enteros de un autor de llamada y devuelve su suma; *a* y *b* son *parámetros* de tipo **int**.

La función puede ser invocada, o *llamada*, desde cualquier lugar del programa. Los valores que se pasan a la función son los *argumentos*, cuyos tipos deben ser compatibles con los tipos de los parámetros en la definición de la función.

No hay ningún límite práctico para la longitud de la función, pero el buen diseño tiene como objetivo las funciones que realizan una sola tarea bien definida. Los algoritmos complejos deben dividirse en funciones más sencillas y fáciles de comprender siempre que sea posible.

Las funciones definidas en el ámbito de clase se denominan funciones miembro. En C++, a diferencia de otros lenguajes, una función también pueden definirse en el ámbito de espacio de nombres (incluido el espacio de nombres global implícito). Estas funciones se denominan *funciones* gratuitas o *funciones* que no son miembro; se usan ampliamente en la biblioteca estándar.

Las funciones pueden ser *sobrecargadas*, lo que significa que diferentes versiones de una función pueden compartir el mismo nombre si difieren por el número y/o tipo de parámetros formales .

Métodos de búsqueda

La búsqueda es una operación que tiene por objeto la localización de un elemento dentro de la estructura de datos (arreglo). A menudo un programador estará trabajando con grandes cantidades de datos almacenados en arreglos y pudiera resultar necesario determinar si un arreglo contiene un valor que coincide con algún valor clave o buscado.

búsqueda secuencial

La **búsqueda secuencial** es un método de búsqueda utilizado en programación para encontrar un elemento específico en una lista o arreglo. En el contexto de C o cualquier otro lenguaje de programación, la búsqueda secuencial se realiza siguiendo estos pasos:

- Se comienza desde el primer elemento de la lista o arreglo.
- Se compara el elemento actual con el elemento que estás buscando.
- Si los elementos son iguales, la búsqueda se detiene y se devuelve la posición en la que se encontró el elemento.
- Si los elementos son diferentes, se pasa al siguiente elemento en la lista.
- Se repiten los pasos 2 a 4 hasta que se encuentre el elemento buscado o se llegue al final de la lista sin encontrarlo.
- La búsqueda secuencial es simple y fácil de implementar, pero puede ser ineficiente en listas grandes, ya que debe revisar cada elemento uno por uno. La cantidad de comparaciones en el peor caso es igual al tamaño de la lista.

búsqueda binaria

La búsqueda binaria, también conocida como búsqueda dicotómica, es un algoritmo de búsqueda eficiente utilizado en programación para encontrar un elemento específico en una lista o arreglo ordenado. La principal característica de la búsqueda binaria es que reduce a la mitad el espacio de búsqueda en cada iteración, lo que la hace mucho más rápida que la búsqueda secuencial en listas grandes. El proceso de búsqueda binaria en C (u otro lenguaje) se realiza de la siguiente manera:

- Se comienza con una lista ordenada (por ejemplo, en orden ascendente).

- Se establece un rango de búsqueda, generalmente utilizando dos índices, uno para el inicio y otro para el final del rango.
- Se calcula el índice del elemento medio en el rango actual (la mitad entre el índice de inicio y el índice de final).
- Se compara el elemento medio con el elemento que se busca.
- Si el elemento medio es igual al elemento buscado, la búsqueda se detiene, y se devuelve la posición del elemento medio como resultado.
- Si el elemento medio es menor que el elemento buscado, se ajusta el índice de inicio al índice del elemento medio + 1, descartando la mitad inferior del rango.
- Si el elemento medio es mayor que el elemento buscado, se ajusta el índice de final al índice del elemento medio - 1, descartando la mitad superior del rango.
- Se repiten los pasos 3 a 7 hasta que se encuentre el elemento buscado o hasta que el rango de búsqueda se reduzca a cero (lo que indica que el elemento no está en la lista).

La búsqueda binaria es especialmente eficiente en listas grandes, ya que reduce el espacio de búsqueda a la mitad en cada iteración. La cantidad máxima de comparaciones requeridas es logarítmica en el tamaño de la lista, lo que la hace mucho más rápida que la búsqueda secuencial, que puede requerir comparaciones lineales.

Métodos de ordenación

La ordenación o clasificación es el proceso de organizar datos en algún orden o secuencia específica, tal como creciente o decreciente, para datos numéricos, o alfabéticos, para datos de caracteres. Los métodos de ordenación más directos son los que se realizan en el espacio ocupado por el array.

Método de la burbuja

El método de ordenación de burbuja, también conocido como "bubble sort" en inglés, es un algoritmo simple de ordenación que se utiliza para ordenar elementos en una lista o arreglo en orden ascendente o descendente. Aunque es fácil de entender y de implementar, el ordenamiento de burbuja no es muy eficiente en comparación con otros algoritmos de ordenación, especialmente en listas grandes.

Es más adecuado para fines educativos y de aprendizaje que para aplicaciones prácticas en la mayoría de los casos. La idea principal detrás del ordenamiento de burbuja es iterar a través de la lista varias veces, comparando pares de elementos adyacentes y realizando un intercambio si es necesario para colocar los elementos en su posición correcta.

Método de selección

El método de ordenación por selección (Selection Sort) es un algoritmo de ordenación simple que funciona seleccionando repetidamente el elemento más pequeño (o más grande, dependiendo del orden deseado) de la lista no ordenada y colocándolo al principio (o al final) de la lista ordenada. Este proceso se repite hasta que toda la lista esté ordenada.

Librerías propias en C

En C, las "librerías propias" se refieren a bibliotecas de funciones y códigos escritos por el programador o el equipo de desarrollo específicamente para un proyecto o aplicación en particular. Estas bibliotecas contienen funciones personalizadas que se adaptan a las necesidades específicas del programa que se está desarrollando.

Las librerías propias pueden contener una variedad de funciones que realizan tareas específicas y que son utilizadas repetidamente en el código de la aplicación.

Struct

La estructura es una colección de variables, la cual puede poseer distintos tipos de datos (a diferencia de los arreglos que solamente pueden tener un solo tipo de dato). Es un tipo de dato definido por el usuario. Son también conocidas como Registros. Ayudan a organizar y manejar datos complicados en programas debido a que agrupan diferentes tipos de datos a los que se los trata como una sola unidad en lugar de ser vistas como unidades separadas

PROCEDIMIENTO

Funciones en C

```
#include <stdio.h>
#include <stdlib.h>

void sumaDosEnteros (int entero1, int entero2);

int main() {
   printf("Bienvenidos al programa\n");
   sumaDosEnteros(3,5);
   return 0; // Ejemplos aprenderaprogramar.com
}

void sumaDosEnteros (int entero1, int entero2) {
   int resultado = 0;
   resultado = entero1 + entero2;
   printf("Si sumamos %d y %d obtenemos %d\n", entero1, entero2, resultado);
   return;
}
```

Búsqueda secuencial

```c
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#define TAM 10

void main() {
int a[TAM], temp, i, j, num;

randomize(); //Inicializa el generador de numeros aleatorios
printf ("Llenando arreglo con números aleatorios\n");
for (i=0; i< TAM; i++)
    a[i]=random(100);
printf ("Numero a buscar? ");
scanf ("%d", &num);
for (i=0; i< TAM; i++)
    if (a[i] == num){
        printf ("\nValor encontrado");
        printf ("\nPosicion: %d", i);
    }
    else
        printf ("\nNo existe");
    printf ("El arreglo era:\n");
    for (i=0; i< TAM; i++)
        printf ("%d ", a[i]);
getch();
}
```

Búsqueda binaria

```c
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#define TAM 15

void main(){
    int a[TAM], busca, temp, bajo, alto, central;

printf("Llenando el arreglo con números aleatorios\n");
randomize(); //Inicializa el generador de aleatorios
for (int i=0; i< TAM; i++)
    a[i]=random(100);
        //Implementacion de Ordenamiento por burbuja de menor a mayor
        printf ("Ordenando arreglo...\n");
        for (int j=1; j <= TAM; j++)
            for (i=0; i< TAM-1; i++)
                if (a[i] > a[i+1]){
                    temp = a[i];
                    a[i] = a[i+1];
                    a[i+1] = temp;
                }

        //Implementacion de busqueda binaria
        printf ("\nIntroduce un numero a buscar: ");
        scanf ("%d", &busca);

        bajo = 0;
```

```
            alto = TAM-1;
            central = (bajo+alto)/2;
            while (bajo < alto && busca != a[central]){
                if(busca > a[central])
                    bajo = central+1;
                else
                    alto = central-1;
                central=(bajo+alto)/2;
            }

            if (busca == a[central])
                printf ("\n%d encontrado en posicion %d", busca, central);
            else
                printf ("\n%d no existe", busca);
            printf ("\n\nEl arreglo ordenado era\n\n");
            for (i=0; i< TAM; i++)
                printf ("%d ", a[i]);
    getch();
    }
```

Método de la burbuja

```
int
main(void)
{
    // El arreglo
    int arreglo[] = {30, 28, 11, 96, -5, 21, 18, 12, 22, 30, 97, -1, -40, -500};
    /*
    Calcular la longitud, puede ser definida por ti o calculada:
    https://parzibyte.me/blog/2018/09/21/longitud-de-un-arreglo-en-c/
    */
    int longitud = sizeof arreglo / sizeof arreglo[0];
    /*
    Imprimirlo antes de ordenarlo
    */
    printf("Imprimiendo arreglo antes de ordenar...\n");
    for (int x = 0; x < longitud; x++) {
     printf("%d ", arreglo[x]);
    }
    // Un salto de línea
    printf("\n");


    /*
    Invocar al método de la burbuja indicando todo el arreglo, desde 0 hasta el
    índice final
    */
    burbuja(arreglo, longitud);
    /*
    Imprimirlo después de ordenarlo
```

```
        */
        printf("Imprimiendo arreglo despues de ordenar...\n");
        for (int x = 0; x < longitud; x++)
          printf("%d ", arreglo[x]);
        return 0;
      }
```

Método de selección

```
void selection_sort(int *vector, int taille)
{
    int actual, mas_pequeno, j, temp;

    for (actual = 0; actual < taille - 1; actual++)
    {
        mas_pequeno = actual;
        for (j = actual 1; j < taille; j++)
            if (vector[j] < vector[mas_pequeno])
             mas_pequeno = j;
      temp = vector[actual];
         vector[actual] = vector[mas_pequeno];
      vector[mas_pequeno] = temp;
   }
}
```

Struct

```
#include <stdio.h>
#include <string.h>
```

```
// Definición de la estructura para representar un estudiante
struct Estudiante {
    char nombre[50];
    int edad;
    float calificacion;
};

int main() {
    // Declaración de una variable del tipo Estudiante
    struct Estudiante estudiante1;

    // Asignación de valores a los campos de la estructura
    strcpy(estudiante1.nombre, "Juan Perez");
    estudiante1.edad = 20;
    estudiante1.calificacion = 8.5;

    // Impresión de la información del estudiante
    printf("Nombre del estudiante: %s\n", estudiante1.nombre);
    printf("Edad del estudiante: %d\n", estudiante1.edad);
    printf("Calificación del estudiante: %.2f\n", estudiante1.calificacion);

    return 0;
}
```

1. **struct Estudiante**: Define una estructura llamada "Estudiante" que tiene tres campos: **nombre**, **edad** y **calificacion**.
2. **struct Estudiante estudiante1;**: Declara una variable **estudiante1** del tipo **Estudiante** para almacenar la información de un estudiante.
3. **strcpy(estudiante1.nombre, "Juan Perez");**: Copia el nombre "Juan Perez" en el campo **nombre** de la estructura **estudiante1**.
4. **estudiante1.edad = 20;**: Asigna el valor 20 al campo **edad** de **estudiante1**.
5. **estudiante1.calificacion = 8.5;**: Asigna el valor 8.5 al campo **calificacion** de **estudiante1**.

6. **printf**: Imprime en la consola la información del estudiante, incluyendo su nombre, edad y calificación.
7. **return 0;**: Devuelve 0 para indicar que el programa se ejecutó correctamente.

RESULTADOS Y CONCLUSIONES

En conclusión, podemos decir que las estructuras en el lenguaje de programación C nos ayudan para poder definir tipos de datos personalizados y combinar diferentes tipos de variables en una sola entidad cohesiva. A través de la declaración de una estructura, hemos sido capaces de agrupar datos relacionados de manera lógica, lo que facilita la manipulación y organización de información compleja.

Además, pudimos ver cómo las estructuras pueden ser utilizadas para representar entidades del mundo real, como estudiantes, empleados, o cualquier otra entidad con un conjunto específico de atributos. A través de la práctica, hemos aprendido a definir una estructura, declarar variables de ese tipo, y asignar valores a sus campos.

Asimismo, podido ver cómo acceder y manipular los datos dentro de una estructura utilizando operaciones como asignaciones y copias de cadenas. Es importante destacar que las estructuras son fundamentales en la programación en C, ya que proporcionan un enfoque organizado y eficiente para gestionar datos complejos. Esto através de la comprensión de las estructuras, hemos sentado las bases para manejar proyectos más complejos y hemos adquirido las habilidades necesarias para gestionar información de manera efectiva en aplicaciones más avanzadas.