Universidad Autónoma de Baja California

Facultad de Ingeniería, Arquitectura y Diseño

Ingeniero en Computación

Asignatura:

Programación Estructurada

Actividad 10:

FUNCIONES DE MÉTODO Y BÚSQUEDA

Brayan Arturo Rocha Meneses

Matricula:

371049

Ensenada Baja California 14 de Abril del

2024

Introducción:

En esta actividad nos enfocamos principalmente en aprender a usar las funciones de STRUCT y todo lo que implica, el uso correcto de typedef y como nos puede ayudar a la hora de que llamar a nuestras variables y estructuras sea menos tedioso y mas comprensible. También se repasaron temas previos como la ordenación y búsqueda de cadenas así como también los métodos de validación para estas.

Competencia:

Se espera que con esta actividad, nosotros como alumnos logremos hacer un uso adecuado de estos métodos, que podamos optimizar mejor nuestro código y que nos apegamos a las estructuras originales de C para que este se mantenga eficiente. Con el uso de la validación se espera que nuestro programa nunca se rompa y así lograr que estos estén mejor elaborados, seguirnos desarrollando como programadores y sobre todo usar estas nuevas funciones TYPEDEF y STRUCT con inteligencia y lógica.

Fundamentos:

- **Crear arreglos de nombres para mandar a llamar a la hora de generar los nombres aleatoriamente:**

```
// NOMBRES: Automaticos
char PriNomFem[10][10] = {"ANA", "MARIA", "MIA", "ELBA", "IRMA", "SONIA", "LUZ", "DULCE", "EVA",
"CARMEN"};
char SegNomFem[10][15] = {"VICTORIA", "FERNANDA", "CRISTINA", "CAROLINA", "ALEJANDRA", "ISABELA",
"DANIELA", "MARTINA", "SOFIA", "MARIANA"};
char PriNomMas[10][10] = {"LUIS", "HECTOR", "SAUL", "SAID", "IVAN", "JESUS", "ALDO", "PABLO",
"KEVIN", "OMAR"};
char SegNomMas[10][10] = {"FERNANDO", "ALEJANDRO", "RAFAEL", "ANTONIO", "FRANCISCO", "ALFREDO",
"JAZIEL", "MANUEL", "MIGUEL", "JAVIER"};
char Apellido1[20][10] = {"LOPEZ", "GOMEZ", "DIAZ", "OROZCO", "AGUILAR", "CAMPOS", "JIMENEZ",
"VAZQUEZ", "LOMA", "PEREZ", "ANDRADE", "PALACIOS", "GONZALEZ", "HERNANDEZ", "RODRIGUEZ",
"MARTINEZ", "CHAVEZ", "CRUZ", "SANCHEZ", "QUIJADA"};
char Apellido2[20][10] = {"LOPEZ", "GOMEZ", "DIAZ", "OROZCO", "AGUILAR", "CAMPOS", "JIMENEZ",
"VAZQUEZ", "LOMA", "PEREZ", "ANDRADE", "PALACIOS", "GONZALEZ", "HERNANDEZ", "RODRIGUEZ",
"MARTINEZ", "CHAVEZ", "CRUZ", "SANCHEZ", "QUIJADA"};
```

- **Mensajes a usuario y menu:**

```
// Menu Principal
int main()
{
srand(time(NULL));
    menu();
return 0;
}
// Menu para el usuario
int msges()
{
int op;
system ("CLS");
printf ("\n MENU DE LA ACTIVIDAD 10 \n");
    printf("1.- AGREGAR AUTOMATICO \n");
    printf("2.- AGREGAR MANUAL \n");
    printf("3.- ELIMINAR REGISTRO \n");
printf("4.- BUSCAR \n");
printf("5.- ORDENAR \n");
    printf("6.- IMPRIMIR \n");
    printf("0.- SALIR \n");
op = ValidarCadena("Escribe el numero de opcion que elegiste: \n", 0, 6);
// Retorno a opcion
    return op;
}
```

- **Casos segun la opción que elija el usuario:**

```
// Casos para el menu segun la opcion que elijan
void menu()
{
int op; int i
    = 0;
int EliminarMatricula; int
    MatriculaBuscada; int
    indice;
int TipoB;
int encontrado = 0; do{
system("CLS");
        op=msges();
        switch (op)
        {
```

- **Caso 1, manda a llamar a la función de GenerarAutom para crear 10 alumnos aleatoria mente y tiene una condición establecida para no excederse de 500:**

```
case 1:
// Limite total de 500 if (i + 10 <=
                500)
                {
for (int j = 0; j < 10; j++)
                    {
if (i < 500)
                        {
alumno[i] = GenerarAutom(); i++;
                        }
                    }
printf("10 alumnos agregados correctamente.\n");
                }
else
                {
printf("No es posible agregar 10 alumnos. Limite alcanzado.\n");
                }
TipoB = 0; system("PAUSE");
                break;
```

- **Caso 2, hace mas o menos lo mismo que el ca**

```
    Talumno GenerarAutom(void);
int AlumnosEliminados[500];
int CantidadEliminados = 0;
int rango(int ri, int rf);
int GenerarMatriculaRandom();
int GenerarEdadRandom();
char GenerarNombreRandom(char nombre[]);
int OrdenarMatricula(Talumno alumnos[], int cantidad);
int BuscarMatricula(Talumno alumnos[], int BuscarMatricula, int cantidad);
int BusquedaBinaria(Talumno alumnos[], int i, int d, int Matricula);
```

so 1 pero para la generación manual de los alumnos y esta va obviamente de 1 por 1:

```
case 2:
// Verifica si se puede agregar mas sin pasarnos if (i < 500)
                {
alumno[i] = GenerarManual();
```

```
i++;
printf("Alumno agregado correctamente.\n");
                }
else
                {
printf("No es posible agregar mas alumnos. Limite alcanzado.\n");
                }
TipoB = 0; system("PAUSE");
                break;
```

- **Caso 3, Elimina un alumno ingresado ya sea aleatoriamente o manual por matricula:**

```
case 3:
                EliminarMatricula = ValidarCadena("Ingresa la matricula del alumno a eliminar: ",
300000, 399999);
for (int j = 0; j < i; j++)
                {
if (alumno[j].matricula == EliminarMatricula)
                    {
if (alumno[j].status == 1)
                        {
alumno[j].status = 0; AlumnosEliminados[CantidadEliminados] = j;
                            CantidadEliminados++;
printf("Alumno eliminado correctamente.\n");
                        }
else
                        {
                            printf("El alumno con matricula %d ya ha sido eliminado
anteriormente.\n", EliminarMatricula);
                        }
encontrado = 1;
                    }
                }
if (!encontrado)
                {
printf("Alumno con matrícula %d no encontrado.\n", EliminarMatricula);
                }
TipoB = 0; system("PAUSE");
                break;
```

- **Caso 4, busca un alumno por matricula y tenemos la busqyeda secuencial y la binaria dependiendo si nuestro registro ya se encuentra ordenado o no:**

```
case 4:
if(TipoB == 0)
                {
```

```
                    MatriculaBuscada = ValidarCadena("Ingresa la matricula (entre 300000 y
399999): ", 300000, 399999);
                    indice = BuscarMatricula(alumno, MatriculaBuscada, i);
                    if (indice != -1)
                    {
                        printf("
                    \n");
                        printf("| %-7s | %-12s | %-40s | %-6s | %-9s |\n", "STATUS", "MATRICULA",
"NOMBRE", "EDAD", "SEXO");
                        printf("
                    \n");
                        if (alumno[indice].status == 1)
                        {
                            printf("| %-7d | %-12d | %-40s | %-6d | %-9s |\n",
alumno[indice].status, alumno[indice].matricula, alumno[indice].nombre, alumno[indice].edad,
(alumno[indice].sexo == 0 ? "Femenino" : "Masculino"));
                        }
                        printf("
                    \n");
                                }
                                else
                                {
                                                                printf("Alumno no
                                                                encontrado.\n");
                                }
                }
                else
                {

                                                MatriculaBuscada = ValidarCadena("Ingresa la
                                                matricula (entre 300000 y 399999):
", 300000, 399999);

                                                indice = BusquedaBinaria(alumno,
                                                0, i - 1, MatriculaBuscada); if
                                        (indice != -1)
                                                {
                        printf("
                        \n");
                        printf("| %-7s | %-12s | %-40s | %-6s | %-9s |\n", "STATUS", "MATRICULA",
"NOMBRE", "EDAD", "SEXO");
                        printf("
                        \n");
                        if (alumno[indice].status == 1)
                        {
                            printf("| %-7d | %-12d | %-40s | %-6d | %-9s |\n",
alumno[indice].status, alumno[indice].matricula, alumno[indice].nombre, alumno[indice].edad,
(alumno[indice].sexo == 0 ? "Femenino" : "Masculino"));
                        }
                        printf("
                        \n");
                    } else {
                        printf("Alumno no encontrado.\n");
                    }
                }
                system("PAUSE");
                break;
```

- **Caso 5, odena todos los alumnos que se hayan generado por matricula de la mas chica a la mas grande y lo imprime:**

```
            case 5:
                TipoB = OrdenarMatricula(alumno, i);
                printf("Alumnos ordenados por matricula:\n");
                printf("-------------------------------------------------------------------
--------------\n");
                printf("| %-7s | %-12s | %-40s | %-6s | %-9s |\n", "STATUS", "MATRICULA", "NOMBRE",
"EDAD", "SEXO");
                printf("-------------------------------------------------------------------
--------------\n");
                for (int j = 0; j < i; j++)
                {
                    if (alumno[j].status == 1)
                    {
                        printf("| %-7d | %-12d | %-40s | %-6d | %-9s |\n", alumno[j].status,
alumno[j].matricula, alumno[j].nombre, alumno[j].edad, (alumno[j].sexo == 0 ? "Femenino" :
"Masculino"));
                    }
                }
                printf("-------------------------------------------------------------------
--------------\n");
                system("PAUSE");
                break;
```

- **Caso 6, imprime el registro:**

```
            case 6:
                printf("-------------------------------------------------------------------
--------------\n");
                printf("| %-7s | %-12s | %-40s | %-6s | %-9s |\n", "STATUS", "MATRICULA", "NOMBRE",
"EDAD", "SEXO");
                printf("-------------------------------------------------------------------
--------------\n");
                for (int j = 0; j < i; j++)
                {
                    if (alumno[j].status == 1)
                    {
                        printf("| %-7d | %-12d | %-40s | %-6d | %-9s |\n", alumno[j].status,
alumno[j].matricula, alumno[j].nombre, alumno[j].edad, (alumno[j].sexo == 0 ? "Femenino" :
"Masculino"));
                    }
                }
                printf("-------------------------------------------------------------------
--------------\n");
                TipoB = 0;
                system("PAUSE");
                break;
```

- **Casos de salida e invalido:**

```
case 0:
printf("Saliendo del programa.\n");
```

```
break;
default:
printf("Opcion no valida.\n"); break;
        }
} while (op != 0);
}
```

- **Funciones utilizadas para generar automaticamente:**

```
// GENERAR AUTOMATICO
Talumno GenerarAutom(void)
{
Talumno alumno;
// Definir status
    alumno.status = 1;
alumno.matricula = GenerarMatriculaRandom();
// Almacena el sexo en una variable
char sexo = GenerarNombreRandom(alumno.nombre);
    alumno.edad = GenerarEdadRandom();
alumno.sexo = sexo;
// Retorno al alumno
    return alumno;
}
// GENERA NUMERO: Aleatoriamente
int rango(int ri, int rf) {
return ri + rand() % (rf - ri + 1);
}
// GENERAR NOMBRE COMPLETO: Aleatoriamente
char GenerarNombreRandom(char nombre[])
{
// Para poner el nombre temporalmente
    char PriNom[10];
char    SegNom[10];
    char PriApe[10];
    char SegApe[10];
    char sexo;
// Genera el sexo random if
    (rand() % 2 == 0)
    {
sexo = 0; // Femenino
strcpy(PriNom, PriNomFem[rango(0, 9)]);
if (rand() % 2 == 0)
        {
strcpy(SegNom, SegNomFem[rango(0, 9)]);
        }
else
        {
SegNom[0] = '\0';
        }
    }
else
    {
sexo = 1; // Masculino
```

```
strcpy(PriNom, PriNomMas[rango(0, 9)]);
if (rand() % 2 == 0)
        {
strcpy(SegNom, SegNomMas[rango(0, 9)]);
        }
else
        {
SegNom[0] = '\0';
        }
    }
// Generar los apellidos
strcpy(PriApe, Apellido1[rango(0, 19)]);
strcpy(SegApe, Apellido2[rango(0, 19)]);
// Componer el nombre completo
// Iniciar el nombre como una cadena que esta vacia
    nombre[0] = '\0';
strcat(nombre, PriNom); if
    (SegNom[0] != '\0')
    {
// Agregar espacio si hay un segundo nombre
        strcat(nombre, " ");
strcat(nombre, SegNom);
    }
// Agregar espacio entre los nombres y apellidos
    strcat(nombre, " ");
strcat(nombre, PriApe);
    strcat(nombre, " ");
    strcat(nombre, SegApe);
// Retorna el sexo
    return sexo;
}
// GENERAR MATRICULA: Aleatoriamente
int GenerarMatriculaRandom()
{
int ri = 300000; int
    rf = 399999;
return ri + rand() % (rf - ri + 1);
}
// GENERAR EDAD: Aleatoriamente
int GenerarEdadRandom()
{
int ri = 17; int
    rf = 59;
return ri + rand() % (rf - ri + 1);
}
```

- **Funciones utilizadas para generar manualmente:**

```
Talumno GenerarManual(void)
{
Talumno alumno; char
    PriNom[100];
```

```
        char
    SegNom[100]; char
    PriApe[100]; char
    SegApe[100];
int op;
// STATUS
alumno.status = 1;
// MATRICULA
    alumno.matricula = ValidarCadena("Ingresa la matricula (entre 300000 y 399999): ", 300000,
399999);
// NOMBRE: Ingresar el primer nombre do {
printf("Ingresa tu primer nombre: \n");
        fflush(stdin);
gets(PriNom); Mayusculas(PriNom);
op = ValidarCadenaTexto(PriNom); if (op
        != 1)
        {
printf("Nombre no valido.\n");
        }
} while (op != 1);
// Almacenar el primer nombre
    strcpy(alumno.nombre, PriNom);
// Preguntar si tiene segundo nombre
op = ValidarCadena("Tienes un segundo nombre? (SI = 0, NO = 1): \n", 0, 1); if
    (op == 0)
    {
printf("Ingresa tu segundo nombre: \n");
        fflush(stdin);
gets(SegNom); Mayusculas(SegNom);
op = ValidarCadenaTexto(SegNom); if (op
        != 1)
        {
printf("Nombre no valido.\n");
        }
strcat(alumno.nombre, " ");
        strcat(alumno.nombre, SegNom);
    }
// Ingresar el primer apellido do {
printf("Ingresa tu primer apellido: \n");
        fflush(stdin);
gets(PriApe); Mayusculas(PriApe);
op = ValidarCadenaTexto(PriApe); if (op
        != 1)
        {
printf("Apellido no valido.\n");
        }
} while (op != 1);
// Preguntar si tiene segundo apellido
op = ValidarCadena("Tienes un segundo apellido? (SI = 0, NO = 1): \n", 0, 1); if
    (op == 0)
```

```
        {
printf("Ingresa tu segundo apellido: \n");
        fflush(stdin);
gets(SegApe); Mayusculas(SegApe);
op = ValidarCadenaTexto(SegApe); if (op
        != 1)
        {
printf("Apellido no valido.\n");
        }
// Agregar un espacio antes
        strcat(alumno.nombre, " ");
        strcat(alumno.nombre, PriApe);
        strcat(alumno.nombre, " ");
        strcat(alumno.nombre, SegApe);
    }
// EDAD
alumno.edad = ValidarCadena("Ingresa la edad (entre 17 y 59): ", 17, 59);
// SEXO
alumno.sexo = ValidarCadena("Ingresa el sexo (0 para Femenino o 1 para Masculino): ", 0, 1);
// Retorno al alumno
    return alumno;
}
```

- **Búsquedas, eliminar y ordenación:**

```
// BUSQUEDA SECUENCIAL
int BuscarMatricula(Talumno alumnos[], int BuscarMatricula, int cantidad)
{
int i;
for (i = 0; i < cantidad; i++)
    {
if (alumnos[i].status == 1)
        {
if(alumnos[i].matricula == BuscarMatricula)
            {
// Se encontro la matricula y devuelve al indice return i;
            }
        }
    }
// No se encontro return
    -1;
}
// BUSQUEDA BINARIA
int BusquedaBinaria(Talumno alumnos[], int i, int d, int Matricula)
{
while (i <= d)
    {
int m = i + (d - i) / 2;
if (alumnos[m].matricula == Matricula)
        {
return m;
        }
```

```
if (alumnos[m].matricula < Matricula)
        {
i = m + 1;
        }
if (alumnos[m].matricula > Matricula)
        {
d = m - 1;
        }
    }
return -1;
}
// ORDENAR
int OrdenarMatricula(Talumno alumnos[], int cantidad)
{
int i, j; Talumno
    temp;
// Ciclo 1
for (i = 0; i < cantidad - 1; i++)
    {
// Ciclo 2 para comparar
for (j = 0; j < cantidad - i - 1; j++)
        {
// Comparacion
if (alumnos[j].status == 1)
           {
if(alumnos[j + 1].status == 1)
                {
if(alumnos[j].matricula > alumnos[j + 1].matricula)
                    {
// Intercambiar los elementos si estan fuera de orden temp = alumnos[j];
alumnos[j] = alumnos[j + 1]; alumnos[j + 1] = temp;
                    }
                }
            }
        }
    }
return -1;
}
```

PROGRAMA DE MI LIBRERIA PERSONALIZADA

eminem.h

```
// - LIBRERIAS - //
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>

/*-------------------------------------------*/
```

```c
// - FUNCIONES - //
// PARA VALIDAR:
int ValidarCadena(char mensj[], int ri, int rf);
int ValidarCadenaTexto(const char cadena[]);
/*-----------------------------------------------*/
// - VALIDACION - //
// CADENA: El valor del numero entre los rangos especificados.
int ValidarCadena(char mensj[], int ri, int rf)
{
    // Declarar variables
    int num;
    // Cadena que va a leer el mensaje ingresado
    char cadena[200];
    do
    {
        printf("%s", mensj);
        // Borrar basura
        fflush(stdin);
        gets(cadena);
        num = atoi(cadena);
    } while (num < ri || num > rf);
    // Retorna el valor qUe num, entre los rangos dados
    return num;
}
```

- **Esta es la que valida si esta vacía, si tiene espacios al inicio o al final, si tiene dobles espacios, si tiene acentos, si esta en mayusculas, si no tiene caracteres especiales y si tampoco tiene números:**

```c
// CADENA DE TEXTO
int ValidarCadenaTexto(const char cadena[])
{
int longitud = strlen(cadena);
    // CADENA VACIA
if (longitud == 0)
    {
return 0;
    }
// ESPACIO AL INICIO O FIN
if (cadena[0] == ' ' || cadena[longitud - 1] == ' ')
    {
return 0;
    }
// SOLO MAYUSCULAS Y ESPACIOS
for (int i = 0; cadena[i] != '\0'; i++)
    {
if (cadena[i] == ' ')
        {
// DOBLES ESPACIOS
if(cadena[i + 1] == ' ')
            {
return 0;
            }
```

```
            }
else
        {
// CARACTERES NO VALIDOS
if(cadena[i] < 'A' || cadena[i] > 'Z')
            {
return 0;
            }
        }
    }
// TODO AL 100
return 1;
}
```

- **Esta función convierte en mayusculas una cadena, esta me funciono para convertir en mayusculas el nombre cuando lo ingresan manualmente antes de validar:**

Procedimiento:

CURP

ACTIVIDAD 10

REALICE EL SIGUIENTE PROGRAMA QUE CONTENGA UN MENU

1.- AGREGAR (AUTOM 10 REGISTROS)
2.- AGREGAR MANUAL
3- ELIMINAR REGISTRO (lógico)
4.- BUSCAR
5- ORDENAR
6.- IMPRIMIR
0.- SALIR

UTILIZAR UN ARREGLO DE 500 REGISTROS

SE DEBERÁ **UTILIZAR ESTRUCTURAS** CON LOS DATOS BÁSICOS DE UN ALUMNO (status, Matricula, ApPat, ApMat, Nombre, Edad, Sexo)

Busqueda y Ordenacion por campo MATRICULA nota: usar librería propia

MENÚ

1.- AGREGAR (AUTOM 10 REGISTROS)
2.- AGREGAR MANUAL
3- ELIMINAR REGISTRO (lógico)
4.- BUSCAR
5- ORDENAR
6.- IMPRIMIR
0.- SALIR

UTILIZAR UN ARREGLO DE 500 REGISTROS
SE DEBERÁ **UTILIZAR ESTRUCTURAS** CON LOS DATOS BÁSICOS DE UN ALUMNO (status, Matricula, ApPat, ApMat, Nombre, Edad, Sexo)
Busqueda y Ordenacion por campo MATRICULA

nota: usar librería propia

Conclusiones:

1. Funciones: Las funciones son bloques de código que realizan una tarea específica y pueden ser reutilizadas, lo que ayuda a mantener el código limpio y eficiente.
2. Métodos de Ordenación: Los métodos de ordenación son algoritmos que organizan los registros de una tabla en algún orden secuencial de acuerdo a un criterio de ordenamiento. Estos métodos permiten organizar eficientemente grandes conjuntos de datos, lo que resulta en mejoras significativas en el rendimiento y la eficiencia de los algoritmos y programas.
3. Métodos de Búsqueda: Los métodos de búsqueda permiten localizar un nodo en particular si es que existe. La búsqueda puede ser la acción de recuperar datos o información, siendo una de las actividades que tiene más aplicaciones en los sistemas de información. Por ejemplo, la búsqueda secuencial o lineal es un método para encontrar un valor objetivo dentro de una lista, comprobando secuencialmente cada elemento de la lista para el valor objetivo hasta que es encontrado o hasta que todos los elementos hayan sido comparados.
4. Estructuras de Datos: Las estructuras de datos son formas de organizar y almacenar datos para que puedan ser utilizados de manera eficiente. Algunas estructuras de datos comunes incluyen listas, pilas, colas, árboles y grafos.
5. Librerías: Las librerías son conjuntos de funciones y procedimientos que realizan tareas comunes y que pueden ser utilizadas por diferentes programas. Las librerías ayudan a evitar la repetición de código y facilitan la programación al proporcionar funcionalidades ya desarrolladas.

Anexo: