Ingeniero en computación

Ingeniero en Software y tecnologías emergentes

Materia: Programación Estructurada / Clave **36276**

Alumno: Hernandez Ceseña Ivan Fernando

Matrícula: 373077

Maestro: Pedro Núñez Yépiz

Actividad No. : 12

Tema - Unidad : Archivos de Texto

Ensenada Baja California a 12 de noviembre del 2023

1. INTRODUCCIÓN

Se van a usar funciones para tener un programa con un operador múltiple (switch) y poder acceder por medio de este a la acción que se quiere ejecutar. acompañados de ciclos para automatizar selecciones y entradas de datos. Utilizando vectores y matrices.

Se implementa el uso de una librería (.h) hecha por el alumno. En ella se encontrarán las funciones que se vayan a usar durante el programa.

También se implementa el método de búsqueda para poder llenar vectores y matrices sin repetir valores.

el otro método que se implementa es el de ordenamiento en base al método de la burbuja, pero implementando mas funciones dentro del bloque de código de la función.

se usaron funciones como el fprintf y el sscanf para poder modificar y leer archivos de texto.

2. COMPETENCIA

usar funciones para hacer un código más claro fácil de entender, además de poner en práctica la optimización de código. utilizando vectores, matrices, cadenas y archivos de texto..

3. FUNDAMENTOS

- sintaxis de ciclos.
- funcionamiento de ciclos.
- proceso de ciclos.
- movilización de una función a otra.
- funcionamiento de vectores y matrices y cómo utilizarlas correctamente.
- declaraciones de vectores y matrices.
- metodos de ordenacion y busqueda
- uso de librerías propias
- archivos de texto.
- punteros

4. PROCEDIMIENTO

1.- Cargar Archivo : El programa deberá cargar el vector de registros desde el archivo de texto (solo podrá cargarse una sola vez el archivo)

2.- Agregar : El programa deberá ser capaz de agregar un 10 registros al arreglo y al final del archivo de texto. (Generar automáticamente los datos).

3.- Eliminar : El programa deberá buscar una matrícula en el vector por medio del método de búsqueda más óptimo. Utilizar banderas para escoger el método más adecuado., imprimir el registro y preguntar si se quiere eliminar el registro, (al cerrar el programa se deberar agregar al archivo borrados el registro o registros eliminados, asi se debera mantener dos archivos uno con datos validos y otro con los datos que se borraron)

4.- Buscar : El programa deberá buscar una matrícula en el vector por medio del método de búsqueda más óptimo. Utilizar banderas para escoger el método más adecuado. Mostrar los datos en forma de registro

5.- Ordenar : El programa deberá ordenar el vector por medio del método de ordenación más óptimo. Utilizar banderas para escoger el método más adecuado se ordenará por el campo llave (matrícula)

6.- Mostrar Todo: El programa deberá mostrar todos los registros del vector tal y como están en ese momento ordenado o desordenado. (mostrar en forma de tabla, y de 40 en 40)

7.- Generar Archivo : El programa deberá preguntar al usuario el nombre del archivo, solo nombre sin extensión, el programa generará un archivo con el nombre proporcionado por el usuario con extensión .txt los datos que pondrá en el archivo de texto serán idénticos a los contenidos en el Vector de registros. (ordenado o desordenado). El programa podrá generar múltiples archivos para comprobar las salidas.

8.- Cantidad de registros en archivo : El programa deberá llamar a un archivo externo, donde mande ejecutar el archivo y como parametros el nombre del archivo que se desea evaluar, el programa externo deberar ser capaz de retornar un valor enero que sea la cantida de registros que contiene el archivo en cuestion

9.- Mostrar Borrados: El programa deberá mostrar el archivo de texto tal y como se visualiza con la cantidad de registros que se eliminaron del archivo original y que fueron marcados en su momnto como registros eliminados

5. RESULTADOS Y CONCLUSIONES

en este apartado se declaran prototipos de funciones que vamos a usar para el programa

```
typedef struct _alumnos{
    int status;
    int matricula;
    char apellidoPaterno[30];
    char apellidoMaterno[30];
    char nombre[30];
    int edad;
    int sexo; } Talumns;


void menu(void);
void msg(void);

void registroAleatorio(Talumns reg[], int i, int numero);
void imprimirAleatorio(Talumns reg[], int posicion, int n);
void sexo(Talumns reg[], int i, int li, int ls);
void status(Talumns reg[], int i);
void matricula(Talumns reg[], int li, int ls, int i);
void apellidoPaterno(Talumns reg[], int i);
void apellidoMaterno(Talumns reg[], int i);
void nombre(Talumns reg[], int i);
void edad(Talumns reg[], int i, int li, int ls);
int existeValor(Talumns reg[], int n, int numero);
void eliminarRegistro(Talumns reg[], int posicion);
void buscarElemento(Talumns reg[], int posicion, int bandera);
void ordenarAscendente(Talumns reg[], int n);
int busquedaBinaria(Talumns reg[], int izquierda, int derecha);
void imprimirRegistro(Talumns reg[], int posicion);
void imprimirBuscado(Talumns reg[], int posicion);

void cambiar(Talumns reg[], int i, int j);
int particion(Talumns reg[], int menor, int mayor);
void quicksort(Talumns reg[], int menor, int mayor);
void archivoTexto(Talumns reg[], int posicion);
void numeroRegistros(void);
void generarTxt(Talumns reg[], int posicion);
int cargarArchivo(Talumns reg[], int posicion, char archivo[]);
void inactivos(Talumns reg[], int posicion);
```

menu (controla todo el programa)

```c
void menu()  // funcion menu
{
    int op, n=15, m=4, p=4;  // [op] que controla el switch, las demas variables definen tamano de vector y matriz
    int vector[n];  // declaracion de vector
    int matriz[m][p];  // declaracion de matriz
    int li, ls;   // limite inferior [li], limite superior [ls]
    int buscar, num;   // variables para el caso 6

    //  ciclo do while que controla el caso a ejecutar con ayuda de [op]
    do{
        msg();
        op= validacionNumero("elije una opcion", 0, 6);
        switch(op)
        {
            case 1:
            li=100, ls=200;
            vectorAleatorioSinRepetir(vector, n, li, ls);
            break;

            case 2:
            li=1, ls=16;
            matrizAleatoria(matriz, m, p, li, ls);
            break;

            case 3:
            imprimirVector(vector, n, "vector de numeros sin repetir");
            break;

            case 4:
            imprimirMatriz(matriz, m, p, "matriz de numeros sin repetir");
            break;

            case 5:
            ordenarVectorAscendente(vector, n);
            for(int i=0; i < n; i++)
            {
                printf("%d ", vector[i]);
            }
            printf("\n");
            break;

            case 6:
            li=100, ls=200;

            imprimirVector(vector, n, "busqueda de numero en vector");
            num = validacionNumero("dame el numero a buscar en el vector", li, ls);
            buscar = busquedaSecuencial(vector, n, num);

            if(buscar != -1)
            {
                printf("el numero %d se encuentra en la posicion %d\n", num, buscar);
            }
            else
            {
                printf("el numero no se encuentra en el vector\n");
            }
            break;

        }
    }while(op!=0);  //  condicion  que evalua si op [0] termina el programa
}
```

menu que controla todo el programa

```
            {
                printf("el registro esta lleno\n");
            }
            else
            {
                registroAleatorio(reg, posicion, numero);
                printf("llenado de 10 registros exitoso\n");
                posicion= posicion + numero;
                bandera = 0;
            }
            break;

            case 3:
            eliminarRegistro(reg, posicion);
            break;

            case 4:
            buscarElemento(reg, posicion, bandera);
            break;

            case 5:
            if(posicion > 500)
            {
                printf("quicksort\n");
                quicksort(reg, 0, posicion-1);
            }
            else
            {
                printf("bubblesort\n");
                ordenarAscendente(reg, posicion);
            }
            bandera = 1;
            break;

            case 6:
            printf("registro\n");
            imprimirRegistro(reg, posicion);
            break;

            case 7:
            generarTxt(reg, posicion);
            break;

            case 8:
            numeroRegistros();
            break;

            case 9:
            inactivos(reg, posicion);
            break;
        }
        system("pause");
    }while(op != 10);
}
```

funciones que interactuan con archivos txt

```c
int cargarArchivo(Talumns reg[], int posicion, char archivo[])
{
    FILE *fa;
    Talumns registro;
    int num;
    char sex[10];
    fa = fopen(archivo, "r");
    if(fa)
    {
        do
        {
            fscanf(fa, "%d.- %d %s %s %s %d %s", &num, &registro.matricula, registro.nombre, registro.apellidoPaterno, registro.apellidoMaterno, &registro.edad, sex);
            registro.status = 1;
            if(strcmp(sex,"HOMBRE")==0)
            {
                registro.sexo=1;
            }
            else
            {
                if(strcmp(sex,"MASCULINO")==0)
                {
                    registro.sexo=1;
                }
                else
                {
                    if(strcmp(sex,"MUJER")==0)
                    {
                        registro.sexo=2;
                    }
                    else
                    {
                        if(strcmp(sex,"FEMENINO")==0)
                        {
                            registro.sexo=2;
                        }
                    }
                }
            }
            reg[posicion] = registro;
            posicion++;
        } while (!feof(fa));
        fclose(fa);
        printf("archivo cargado\n");
    }
    else
    {
        printf("el archivo no existe\n");
    }
    return posicion-1;
}
```

```c
void numeroRegistros()
{
    int contador;
    char archivo[50];
    char parametros[50];

    validacionArchivo("ingresa el nombre del archivo que quieres evaluar:", archivo, 50);

    sprintf(parametros, "archivoExterno.exe %s", archivo);
    contador = system(parametros);

    if(contador != -1)
    {
        printf("el archivo %s tiene %d registros", archivo, contador);
    }
    else
    {
        printf("el archivo no se encontro");
    }
}

void generarTxt(Talumns reg[], int posicion)
{
    int i;
    FILE *fa;
    char archivo[30];

    validacionArchivo("escribe como quieres nombrar tu archivo", archivo, 30);

    strcat(archivo, ".txt");
    fa = fopen(archivo, "w");

    for(i =0; i<posicion; i++)
    {
        if(reg[i].status == 1)
        {
            fprintf(fa, "%6d.- %10d %10s %15s %15s %5d", i+1, reg[i].matricula, reg[i].nombre, reg[i].apellidoPaterno, reg[i].apellidoMaterno, reg[i].edad);
            if(reg[i].sexo == 1)
            {
                fprintf(fa, " %8s\n", "HOMBRE");
            }
            else
            {
                fprintf(fa, " %8s\n", "MUJER");
            }
        }
    }

    fclose(fa);
    printf("archivo generado\n");
}
```

```c
void inactivos(Talumns reg[], int posicion)
{
    int j;
    char archivo[30];
    FILE *fb;
    validacionArchivo("escribe como quieres nombrar tu archivo", archivo, 30);

    strcat(archivo, ".txt");
    fb = fopen(archivo, "w");

    for(j=0; j<posicion; j++)
    {
        if(reg[j].status == 0)
        {
            fprintf(fb, "%3d %10d %10s %15s %15s %4d", j+1, reg[j].matricula, reg[j].nombre, reg[j].apellidoPaterno, reg[j].apellidoMaterno, reg[j].edad);
            printf("%3d %10d %10s %15s %15s %4d", j+1, reg[j].matricula, reg[j].nombre, reg[j].apellidoPaterno, reg[j].apellidoMaterno, reg[j].edad);
            if(reg[j].sexo == 1)
            {
                fprintf(fb, " %8s ", "HOMBRE");
                printf(" %8s", "HOMBRE");
            }
            else
            {
                fprintf(fb, " %8s", "MUJER");
                printf(" %8s \n", "MUJER");
            }
            fprintf(fb, "\n");
            printf("\n");
            j++;
        }
    }
    fclose(fb);
}
```

El uso de una librería facilita mucho el entendimiento claro del código, también ayuda a que no se vea muy abultado. Esto nos ayuda a corregir errores de manera más rápida.

Los métodos de ordenamiento y búsqueda también son una herramienta bastante útil a la hora de usar funciones en las que se necesitan valores en órdenes específicos o que no se requieren valores repetidos.

El separar acciones específicas también ayuda a poder usarlas de mejor manera dentro de otros bloques de código y tener códigos más limpios y fáciles de entender.

6. ANEXOS

anexos en el otro archivo:

nombre del archivo: anexo_HCIF_RP12_PE

7. referencias

Diseño de algoritmos y su codificación en lenguaje C

Corona, M.A. y Ancona, M.A. (2011)..

España: McGraw-Hill.

ISBN: 9786071505712

Programación estructurada a fondo:implementación de algoritmos en C

:Pearson Educación.Sznajdleder, P. A. (2017)..

Buenos Aires,Argentina: Alfaomega

Como programar en C/C++

H.M. Deitel/ P.J. Deitel

Segunda edición

Editorial: Prentice Hall.

ISBN:9688804711

Programación en C.Metodología, estructura de datos y objetos

Joyanes, L. y Zahonero, I. (2001)..

España:McGraw-Hill.

ISBN: 8448130138