ESCUELA POLITÉCNICA NACIONAL

ESCUELA DE FORMACIÓN DE TECNÓLOGOS

ALGORITMOS Y ESTRUCTURAS DE DATOS

ASIGNATURA:	Algoritmos y Estructuras de Datos
PROFESOR:	Ing. Loarte Byron
PERÍODO ACADÉMICO:	2019 - B

PROYECTO FINAL

TÍTULO:

Sistema de Gestión de Calificaciones

ESTUDIANTES

Stephanie Muñoz, Mateo Borja, Christian Soledispa, Diana Almeida

FECHA DE REALIZACIÓN: 2 / 3 / 2020

FECHA DE ENTREGA: 3 / 3 / 2020

CALIFICACIÓN OBTENIDA:

FIRMA DEL PROFESOR:

1 PROPÓSITO DEL PROYECTO

Se busca poner en práctica todos los conocimientos adquiridos durante el semestre en la materia de AED, elaborando un sistema de gestión de calificaciones, el cual puede ser muy útil en el entorno actual.

2 OBJETIVOS ESPECÍFICOS

- Familiarizarse con los diferentes algoritmos de búsqueda y ordenamiento que existen.
- Aprender a programar de forma más compacta y organizada mediante la creación y uso de librerías.
- Utilizar archivos de texto como medio para guardar los resultados de un programa y así evitar la pérdida de estos.

3 DESARROLLO Y RESULTADOS DE LA PRÁCTICA

Se pide implementar un sistema de gestión de calificaciones que permita a un docente ingresar sus datos e ir registrando las notas de sus estudiantes para, posterior a ello, ordenarlas utilizando cualquiera de los algoritmos de ordenamiento vistos o buscar un dato mediante los diversos algoritmos de búsqueda. Los resultados deben, además, guardarse en un archivo de texto.
Este programa fue escrito en C++ utilizando CodeBlocks, a continuación, se explicará el funcionamiento del código.

La ejecución de este programa empieza en el archivo principal el cual contiene la función "main".

```
#include <iostream>
#include <fstream>
#include <stdlib.h> //Libreria para utilizar la funcion system
#include <direct.h> //Libreria para crear la carpeta colegio


using namespace std;

#include "ingresar.h"
#include "archivo_calificaciones.h"
#include "ordenar.h"
#include "archivo_ordenar.h"
#include "busqueda.h"
#include "archivo_busqueda.h"

int main()
{
```

Fig. 1 Incluyendo las librerías creadas

Sin embargo, como se observa en la imagen anterior, antes de llegar a la función main es necesario incluir todas las librerías a usarse, tanto aquellas globales (con su espacio de nombres respectivo) como las que fueron creadas para este proyecto (nótese la diferencia al momento de incluir). Cada librería esta almacenada en un archivo de encabezado (.h) y contiene las funciones necesarias para procesar los datos según se pide, a continuación, cada librería se mostrará con sus contenidos.

ingresar.h

Esta biblioteca contiene una única función que permite ingresar cada estudiante con su nombre y su calificación, esto se pasando cada estudiante como elemento de un array. El ingreso de notas cuenta además con protección por si la nota que ingresa es negativa o mayor 20.

```
void ingresar_estudiantes(int n,string nombres[],double notas[])
{

    for(int i=1; i<=n; i++)
    {
        cin.ignore();
        cout<<"\nIngrese el nombre del estudiante: ";
        getline(cin,nombres[i]);
        do
        {
            cout<<"\nIngrese su calificaci□n: ";
            cin>>notas[i];
        }
        while((notas[i]<1)|(notas[i]>20));
    }

}
```

Fig. 2 Función ingresar estudiantes

ordenar.h

En esta biblioteca se encuentran implementadas todas las funciones para realizar los diferentes algoritmos de ordenamiento vistos, los cuales son:

- **Ordenamiento por Burbuja**
  Recorre los elementos de un array por parejas y los intercambia de lugar si el primero es mayor que el segundo, el proceso continúa hasta que el array esté ordenado

```
void burbuja (double notas[], int n,string nombres[])
{
    for (int i = 1; i <= n; i++)
    {
        for(int j = i+1; j <= n; j++)
        {
            if (notas[i] > notas[j])
            {
                double aux = notas[i];
                string auxn=nombres[i];
                notas[i] = notas[j];
                nombres[i]=nombres[j];
                notas[j] = aux;
                nombres[j]=auxn;
            }
        }

    }
}
```

Fig. 3 Función ordenamiento burbuja

- **QuickSort**
  Encuentra un elemento pivote y procede a dividir el array en elementos menores y mayores que el pivote, este proceso se repite recursivamente en las subdivisiones hasta que se tenga el conjunto de elementos ordenados. En la mayoría casos se toma el elemento medio como pivote, pero, dependiendo de la complejidad de la lista a ordenarse, existen formas de hallar un pivote más eficiente.

```
double mitad (double notas[],int pinicial,int pfinal)
{
    return notas[(pinicial+pfinal)/2];

}
void ordenar_quicksort(double notas[],int pinicial,int pfinal,string nombres[])
{
    string aux;
    int i=pinicial;
    int j=pfinal;
    double temp;
    double piv=mitad(notas,pinicial,pfinal);

    do
    {
        while(notas[i]<piv)
        {
            i++;
        }
        while(notas[j]>piv)
        {
            j--;
        }
        if(i<=j)
        {
            temp=notas[i];
            aux=nombres[i];
            notas[i]=notas[j];
            nombres[i]=nombres[j];
            notas[j]=temp;
            nombres[j]=aux;
            i++;
            j--;
        }
    }
    while(i<=j);

    if(pinicial<j)
    {
        ordenar_quicksort(notas,pinicial,j,nombres);
    }
    if(i<pfinal)
    {
        ordenar_quicksort(notas,i,pfinal,nombres);
    }

}
```

Fig. 4 Función QuickSort y su función mitad

- **Ordenamiento por Inserción**
  Este algoritmo utiliza dos sub-arrays, uno ordenado y otro sin ordenar; el array ordenado se construye reinsertando cada elemento no ordenado en el lugar que le corresponde en el array ordenado, al terminar, el array ordenado tendrá el total de elementos.

```
void ordenar_insercion(double notas[],int n,string nombres[])
{
    string auxn;
    int i,pos;
    double aux;
    for(i=1; i<=n; i++)
    {
        pos=i;
        aux=notas[i];
        auxn=nombres[i];
        while((pos>0)&&(notas[pos-1])>aux)
        {
            notas[pos]=notas[pos-1];
            nombres[pos]=nombres[pos-1];
            pos--;
        }
        notas[pos]=aux;
        nombres[pos]=auxn;
    }
}
```

Fig. 5 Función ordenamiento por Inserción

- **Ordenamiento por Selección**
  En este algoritmo se con un elemento menor del array, el cual se asume que es el primero, se realiza un recorrido por el resto de los elementos y, si se encuentra un elemento menor que el primero, ambos se intercambian; El proceso continúa hasta que todos los elementos estén ordenados.

```
void ordenar_seleccion(double notas[], int n,string nombres[])
{
    string auxn;
    int min;
    double aux;
    for(int i=1; i<=n; i++)
    {
        min=i;
        for(int j=i+1; j<=n; j++)
        {
            if(notas[j]<notas[min])
            {
                min=j;
            }
        }
        aux=notas[i];
        auxn=nombres[i];
        notas[i]=notas[min];
        nombres[i]=nombres[min];
        notas[min]=aux;
        nombres[min]=auxn;
    }
}
```

Fig. 6 Función ordenamiento por selección

- **HeapSort**
  El “Heap” o montículo es un espacio en la memoria reservado para las variables creadas durante la ejecución de un programa, este algoritmo almacena los elementos del array en el heap en forma de árbol binario completo, donde cada nodo padre será mayor que sus hijos (si no lo es, intercambiar); luego, se intercambia la raíz del árbol con su menor elemento, el mayor elemento pasa a ser el mayor del array y el menor pasa a ser la nueva raíz con la cual se repetirá el proceso.

```
void HeapSort(double b[], int n,string nombres[])
{
    double valor,temp;
    string valn,tempn;
    int a;

    for(int i=n; i>0; i--)
    {
        for(int j=1; j<=i; j++)
        {
            valor=b[j];
            valn=nombres[j];
            a=j/2;
            while(a>0 && b[a]<valor)
            {
                b[j]=b[a];
                nombres[j]=nombres[a];
                j=a;
                a=a/2;
            }
            b[j]=valor;
            nombres[j]=valn;
        }
        temp=b[1];
        tempn=nombres[1];
        b[1]=b[i];
        nombres[1]=nombres[i];
        b[i]=temp;
        nombres[i]=tempn;
    }

}
```

Fig. 7 Función HeapSort

- **MergeSort**
  Divide el array en mitades recursivamente hasta tener cada elemento por separado, posteriormente se van reintegrando los sub-arrays ubicando cada elemento de modo que esté en orden hasta tener el array completo ordenado.

```
void ordenar_mergesort(double notas[],int pinicial,int pfinal,int medio,int n,string nombres[])
{
    int i,j,k;
    double temp[pfinal-pinicial+1];
    i=pinicial;
    k=0;
    j=medio+1;
    string aux[pfinal-pinicial+1];

//Tomar los elementos y ordenar el arreglo temp

    while(i<=medio && j<=pfinal)
    {
        if(notas[i]<notas[j])
        {
            temp[k]=notas[i];
            aux[k]=nombres[i];
            k++;
            i++;
        }
        else
        {
            temp[k]=notas[j];
            aux[k]=nombres[j];
            k++;
            j++;
        }
    }

    //Almacenar los valors del arreglo sub-derecho

    while(i<=medio)
    {
        temp[k]=notas[i];
        aux[k]=nombres[i];
        k++;
        i++;
    }

    // Almacenar los valores del arreglo sub-izquierdo

    while(j<=pfinal)
    {
        temp[k]=notas[j];
        aux[k]=nombres[j];
        k++;
        j++;
    }
    for(i=pinicial; i<=pfinal; i++)
    {
        notas[i]=temp[i-pinicial];
        nombres[i]=aux[i-pinicial];
    }
}

void dividir(double notas[],int inicial,int final,int n,string nombres[])
{
    int mitad;
    if(inicial<final)
    {
        mitad=(inicial+final)/2;

        dividir(notas,inicial,mitad,n,nombres);
        dividir(notas,mitad+1,final,n,nombres);
        ordenar_mergesort(notas,inicial,final,mitad,n,nombres);
    }
}
```

Fig. 8 Función MergeSort y su función recursiva para dividir

busqueda.h

Terminando con los algoritmos de ordenamiento, esta biblioteca contiene la implementación de los algoritmos de búsqueda vistos, los cuales son:

- **Búsqueda Lineal**
  Recorre el array secuencialmente hasta encontrar el elemento buscado, cuando lo encuentra devuelve la posición en la que este se encuentra en el array.

```
bool busqueda_lineal(double notas[],int n,double dato,string nombres[])
{
    bool c;
    c=false;
    string aux;
    for(int i=1; i<=n; i++)
    {
        if(notas[i]==dato)
        {
            c=true;
            aux=nombres[i];

        }

    }

}
```

Fig. 9 Función Búsqueda Lineal

- **Búsqueda Binaria**
  Requiere de un array ordenado para funcionar, pues compara el elemento buscado con el elemento medio del array y según el resultado, se repite el proceso en el sub-array menor, mayor o bien el elemento buscado es el elemento medio.

```
void busqueda_binaria(double notas[],int n, double dato,string nombres[])
{
    bool c=false;
    string aux;
    int mitad, inferior, superior;
    inferior=1;
    superior=n;
    while(inferior<=superior)
    {
        mitad=(superior+inferior)/2;
        if(notas[mitad]==dato)
        {
            c=true;
            aux=nombres[mitad];
            break;
        }
        if(notas[mitad]>dato)
        {
            superior=mitad-1;
        }
        if(notas[mitad]<dato)
        {
            inferior=mitad+1;
        }
    }
}
```

Fig. 10 Función Búsqueda Binaria

- **Búsqueda por Interpolación**
  Igual que la búsqueda binaria, se encarga de buscar el elemento en sub-arrays mayores o menores que un elemento pivote, el cual ya no es el elemento medio, sino el resultado de la siguiente ecuación:

$$mid = primero + ((num - A[primero]) * (ultimo - primero))/(A[ultimo] - A[primero])$$

```
void busqueda_interpolacion(double notas[],int n,double num,string nombres[])
{
    int primero=0;
    int ultimo=n-1;
    int medio;
    int cont=0;

    while(notas[primero]!=num && cont<n)
    {
        medio=primero+((num-notas[primero])*(ultimo-primero)/(notas[ultimo]-
notas[primero]));

        if(notas[medio]>num)
        {
            ultimo=medio-1;
        }
        else if(notas[medio]<num)
        {
            primero=medio+1;
        }
        else
        {
            primero=medio;
        }
        cont++;
        break;
    }
}
```

Fig, 11 Función Búsqueda por interpolación

Librerías de Archivos

Estas tres librerías restantes son responsables de crear los diferentes archivos de resultados y agrupar en ellos la información de la forma en la que se pide. Son las siguientes:

- **archivo_busqueda.h**
  La función incluida en esta librería guarda el resultado de cualquier búsqueda realizada, el formato que se mostrará en el archivo de texto concuerda con el requerido.

```
void archivo_busqueda(double notas[],int n, double notas_buscar,string nombres[],int menu,string profesor, long int cedula)
{
    ofstream archivo3;
    bool c=false;
    string aux;
    archivo3.open("colegio//b�squeda.txt",ios::out);

    archivo3<<"                                    COLEGIO MEJIA"<<endl;
    archivo3<<"                                REPORTE DE CALIFICACIONES"<<endl;
    archivo3<<endl;
    archivo3<<"B�squeda de calificaciones "<<endl;
    archivo3<<"\nALGORITMO: ";
```

Fig. 12 Formato de salida de la función archivo_busqueda

La función además muestra un mensaje con el nombre del algoritmo seleccionado y, para cada caso, muestra un mensaje si el elemento buscado no se logró encontrar.

- **archivo_ordenar.h**
  Al igual que la función de la biblioteca anterior, esta función puede reconocer que tipo de algoritmo se usó e imprimir un mensaje según sea el caso, esto es posible mediante una sencilla cadena de condicionales, como se muestra a continuación.

```
if(m==1)
    {
        archivo2<<"Burbuja "<<endl;
    }
    if(m==2)
    {
        archivo2<<"Selecci□n "<<endl;
    }
    if(m==3)
    {
        archivo2<<"Merge Sort "<<endl;
    }
    if(m==4)
    {
        archivo2<<"Quick Sort "<<endl;
    }
    if(m==5)
    {
        archivo2<<"Inserci□n "<<endl;
    }
    if(m==6)
    {

        archivo2<<"Heapsort "<<endl;

    }
```

Fig. 13 condicionales múltiples que evalúan el algoritmo usado

- **archivo_calificaciones.h**
  Esta función registra el registro de calificaciones como tal, proporcionando el numero de aprobados y suspensos según las notas de cada estudiante, tal y como se había pedido.

```
        }
        archivo1<<i+1<<"                    "<<nombres[i]<<"                    "<<notas[i];
        archivo1<<endl;
    }
    archivo1<<endl;
    promedio=suma/n;
    archivo1<<"RESUMEN"<<endl;
    archivo1<<endl;
    archivo1<<"Promedio del curso:          "<<promedio<<endl;
    archivo1<<endl;
    archivo1<<"TOTAL                            Aprobados (14-20)       "<<apro<<endl;
    archivo1<<"                                 Suspenso (09-13)        "<<sus<<endl;
    archivo1<<"                                 Reprobados (01-08)      "<<repro<<endl;
    archivo1<<endl;
    archivo1<<endl;
    archivo1<<"                                 _____________________"<<endl;
    archivo1<<"                                   "<<profesor<<endl;
    archivo1<<"                                   "<<cedula;

    archivo1.close();
}
```

Fig. 14 últimas líneas de la función para mostrar calificaciones.

Ahora, luego de haber descrito las principales librerías usadas, es posible volver a la función main. Lo primero que se hacer es declarar las variables que se usaran, así como construir el menú principal que se usará para acceder a todas las funciones implementadas del programa. Tanto el menú, como la selección de opciones fueron implementadas con un do while y un switch, estas dos instrucciones siempre suelen ir juntas en este tipo de código.

```
do
    {
        cout << endl << "Men�:" << endl;
        cout << "\n1.- Ingresar nombre del docente: " << endl;
        cout << "\n2.- Ingresar la materia: " << endl;
        cout << "\n3.- Ingresar el n�mero de estudiantes a registrar: " << endl;
        cout << "\n4.- Ingresar las notas de cada estudiante: " << endl;
        cout << "\n5.- Almacenar los datos en un archivo: " << endl;
        cout << "\n6.- Ordenar las calificaciones obtenidas: " << endl;
        cout << "\n7.- Almacenar las calificaciones ordenadas en un archivo: " << endl;
        cout << "\n8.- Buscar una calificaci�n" << endl;
        cout << "\n9.- Guardar la calificaci�n buscada en un archivo " << endl;
        cout << "\n10.- SALIR" << endl;
        cout << endl;
        cout << "\nIngrese la opci�n que desee: ";
        cin >> n;
        //system ("cls");
        switch (n)
        {
        case 1:
        {
            cin.ignore();
            cout << "\nIngrese el nombre del docente: ";
            getline(cin, nombre_docente);
            cout << "\nIngrese el n�mero de c�dula del docente: ";
            cin >> cedula;


            break;
        }
        case 2:
        {

            cin.ignore();
            cout << "\nIngrese el nombre de la materia: ";
            getline(cin, materia);

        }
        break;
```

Fig. 15 Implementación del menú principal y opciones

Posteriormente se vuelve a usar menus y switches para acceder a las diferentes opciones de ordenamiento/búsqueda que fueron implementadas; el programa, a través de la función main irá llamando a la función que corresponda para ejecutar el ordenamiento/búsqueda y al final de la ejecución del programa se podrá revisar su carpeta de archivos para verificar que los resultados hayan sido guardados con éxito.

4 CONCLUSIONES

- Fue posible observar la diferencia entre un programa con muchísimas líneas de código si se usan o no librerías, quedó claro que, efectivamente, las librerías son la mejor alternativa.
- En la realización de este proyecto se lograron vincular todos los conocimientos adquiridos y usarlos para obtener un buen resultado.

5 BIBLIOGRAFÍA

- Carpeta compartida AED (2019 – B)