Control de Calificaciones

2-12-2014

Objetivo

Generales

La aplicación consiste en un sistema que administra las calificaciones de un grupo.

Específicos

- Desarrollar un lenguaje regular, generando una gramática regular y un autómata finito que lo valide.
- Implementar una de las estructuras de datos vistas en clases: las listas.
- Programación de funciones que realiza el sistema: añadir, modificar, eliminar y buscar.
- Crear el entorno gráfico de la aplicación en C, haciendo uso de la librería <graphics.h>.
- Validación del ingreso y manejo de datos.

Introducción

El universo de los sistemas automatizados, digital o virtual, cada día absorbe más terreno y forma parte ya de nosotros. La tecnología cada día busca satisfacer las necesidades del ser humano, mejorar y optimizar la forma de hacer las cosas, más rápido y mejor.

Todos o la mayoría de estos desarrollos tecnológicos emplean en el diseño y la construcción de software algunos de los conceptos como el de autómata finito y determinados, tipos de gramáticas formales y otros.

La lógica computacional o lógica formal siempre ha sido una herramienta fundamental para el progreso de las ciencias computacionales, y sus desarrollos son la base para elaborar soluciones informáticas, tanto en software como en hardware, incluyendo así también el lenguaje que comunica a ambos.

Un lenguaje de programación es un lenguaje que puede ser utilizado para controlar el comportamiento de una máquina, haciendo referencia particularmente a una computadora. Este consiste en un conjunto de símbolos y reglas sintácticas y semánticas que definen su estructura y el significado de sus elementos y expresiones.

El lenguaje, la forma de decir y comprender el significado y la semántica de las palabras generando lenguajes de comunicación. Existen innumerables expresiones y formas de decir las cosas por lo que se hace necesaria la estandarización o formalización de un lenguaje, de manera que pueda adquirir un significado real para nosotros.

Por medio de un autómata (construcción lógica) que recibe como entrada una cadena de símbolos y produce una salida indicando si dicha cadena pertenece o no a un determinado lenguaje, ayudando a determinar lo eficiente del lenguaje para el programa.

En ciencia de la computación, la asignación dinámica de la memoria es muy importante en la creación de aplicaciones, debido a que la asignación de almacenamiento de memoria por parte de un programa de computador durante el tiempo de ejecución del programa distribuye la propiedad de recursos de memoria entre muchas piezas de código y datos.

Una aplicación de esta técnica para ahorrar memoria es necesario reservarla dinámicamente (sobre la marcha). En una lista enlazada la memoria se va tomando según se necesita. Cuando queremos añadir un nuevo elemento reservamos memoria para él y lo añadimos a la lista. Cuando queremos eliminar el elemento simplemente lo sacamos de la lista y liberamos la memoria usada.

Para ofrecer un mejor entendimiento del proyecto, este documento se ha estructurado en 3 partes.

De inicio se presenta el desarrollo del proyecto. En esta sección se abordará el procedimiento utilizado para la elaboración del proyecto, las herramientas utilizadas y la definición de los procesos utilizados.

Posteriormente se presentan el resultado obtenido de la aplicación obtenida, la integración de la programación de la aplicación.

Finalmente, se presenta las conclusiones de la realización de este proyecto, las mejoras a futuro y observaciones que se detectaron.

Desarrollo de proyecto

Para el desarrollo del proyecto se dividió en 3 fases:

Primera fase (Lenguaje Formal).

Crear un lenguaje formal donde se puedan reconocer las palabras dentro de la cadena, para después determinar el significado de cada una de ellas, identificar el nivel de significado o significados, y así formular una respuesta.

Para ello, la metodología a utilizar consiste en definir los símbolos a utilizar para, posteriormente, definir el alfabeto, y así definir el lenguaje con su gramática, acto seguido se elaborará un autómata que valide las cadenas del lenguaje en cuestión.

Definición de símbolos.

En esta parte se determinó cuáles serían los símbolos a utilizar, así como el significado de cada uno de ellos, se optó por utilizar la simbología de calificaciones empleado en Estados Unidos, ya que es una forma de combinar símbolos y letras del alfabeto.

El sistema utiliza las letras del alfabeto de la A a la F (sin pasar por la E). Para aprobar, hay que obtener como mínimo D- . El máximo puntaje es A+, en tanto que la letra F denotan los niveles deficiente (el alumno resulta reprobado).

Más alta	Más baja	Letra
100.00 %	97.00 %	A+
96.99 %	93.00 %	A
92.99 %	90.00 %	A-
89.99 %	87.00 %	B+
86.99 %	83.00 %	B
82.99 %	80.00 %	B-
79.99 %	77.00 %	C+
76.99 %	73.00 %	C
72.99 %	70.00 %	C-
69.99 %	67.00 %	D+
66.99 %	63.00 %	D
62.99 %	60.00 %	D-
59.99 %	0.00 %	F

Alfabeto.

El alfabeto está determinado por caracteres alfabéticos de A hasta la F, sin contar la E y la unión de símbolos + y – y λ (lambda).

$$\sum = \{A, B, C, D, F, +, -, \lambda\}$$

Gramática.

Gramática regular (tipo 3)

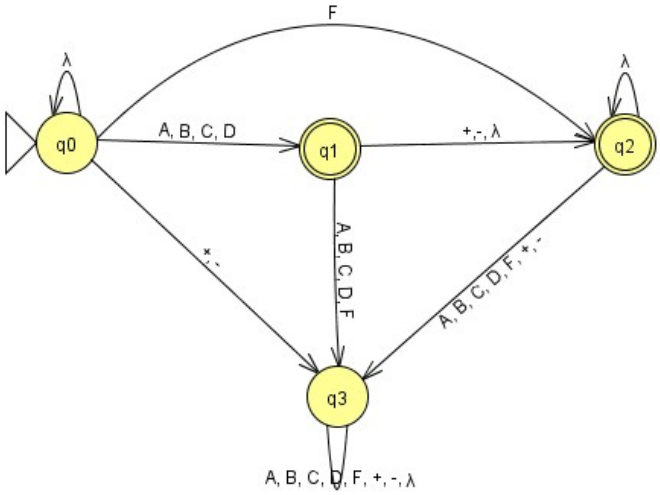
Lineal por la derecha
G = (V, T, P, S)
V = Elementos no terminales = {S, M}
T = Elementos terminales = {A, B, C, D, F, +, -, λ}
P = Reglas de producción =
P:
S - > (A|B|C|D) M | F
M -> +| - |λ
S = Símbolo inicial

Lenguaje.

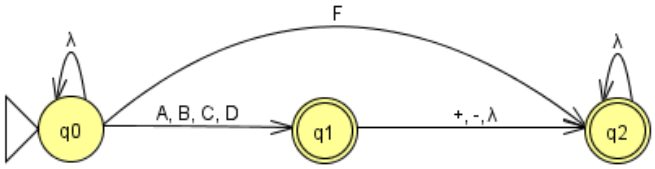
El lenguaje creado es:
L (R) = {A+, A, A-, B+, B, B-, C+, C, C-, D+, D, D-, F}

Autómata.

A partir del alfabeto, la gramática y el lenguaje, se construyó el autómata finito determinista (AFD) que reconociera las palabras sobre el alfabeto, así como también el autómata finito no determinista (AFND) correspondiente.



Autómata finito determinista



Autómata finito no determinista

Después de determinar el lenguaje regular, se puede proseguir a la siguiente fase de programación.

Segunda fase.

Programación del autómata

Una vez planteado el autómata de manera teórica, se llevó a cabo la programación. Para validar todas las posibles entradas, se hizo la programación tomando como base el autómata finito determinista.

El autómata consta de 6 funciones, explicadas a continuación.

Función autómata

En esta función se obtiene como parámetro la longitud de la cadena y se manda llamar la función *ini(),* una vez analizada la cadena se regresara el valor de verdadero o falso si la cadena es válida o invalida, respectivamente.

```
int automata (int cal)
{
    t=cal;
    printf("CADENA: %s",cadena);
    ini();
    if (aceptado)
       return(true);
    else return(false);
    getche();
}
```

Función ini

En esta función se inicializa la variable "cont" en 0, esta variable es la encargada de ir recorriendo cada carácter de la cadena, se establece la variable "aceptado" como falsa y se manda llamar a *q0()*.

```
void ini()
{
      cont =0;
      aceptado = false;
      q0();
}
```

Función q0

Estado inicial del autómata, si el carácter es un espacio se manda llamar al mismo estado y aumenta el contador, si es algún carácter entre "A" y "D" se va al segundo estado, *q1(),* el cual es válido, y se aumenta el contador, si es el carácter "F" se va al tercer estado *q2()* , también valido y si es cualquier otro carácter se va al estado *qError(),* el cual no es válido, por lo tanto la cadena no es válida.

```
void q0()
{
                if (cadena[cont]==32)
                {
                    cont++;
                    q0();
                }
                else if (cadena[cont]=='A' || cadena[cont]=='B' || cadena[cont]=='C' || cadena[cont]=='D')
                {
                     cont++;
                     q1();
                }
                else if (cadena[cont]=='F')
                {
                     cont++;
                     q2();
                }
                else qError();
}
```

Función q1

Segundo estado del autómata, es un estado valido, siempre y cuando la longitud de la cadena sea igual al valor del contador, es decir, si solo se introdujo un símbolo entre "A" y "D". Si el valor del contador es menor al de la cadena se valida si el carácter siguiente es el símbolo "+", "-", o un espacio, en cualquiera de los casos anteriores se manda llamar al tercer estado, *q2()*, y se aumenta el contador, si no es ninguno de los símbolos anteriores se va a *qError().*

```
void q1()
{
     if (cont==t)
            aceptado = true;
     else
     {
                 if (cadena[cont]=='+' || cadena[cont]=='-' || cadena[cont]==32)
                {
                     cont++;
                     q2();
                }
                else qError();
     }
}
```

Función q2

Tercer estado, válido siempre y cuando el contador sea del mismo tamaño de la cadena. Si detecta un espacio se manda llamar a sí mismo, cuando el contador sea del mismo tamaño de la cadena, la cadena será declarada como válida, de lo contrario, si detecta algún otro símbolo se mandara llamar a *qError().*

```
void q2()
{
     if (cadena[cont]==32)
     {
          cont++;
          q2();
     }
     else if (cont==t)
          aceptado = true;
     else qError();
}
```

Función qError

Último estado, estado inválido. Cada que se mande llamar a este estado quiere decir que la cadena es inválida, por lo tanto se establece la variable "aceptado" como falso.

```
void qError()
{
     aceptado = false;
}
```

Listas simplemente enlazadas.

Para la aplicación, como se mencionó, se utilizaron listas enlazadas. Se determinó utilizar este tipo de estructura de datos debido a que en las listas pueden ser utilizadas cuando se necesitan hacer varias operaciones de inserción y eliminación de elementos.

Para definir un elemento de la lista, será utilizado el tipo *struct*. Cada elemento de la lista contendrá campos de datos (Matricula, Nombre, Calificación 1, Calificación 2, Calificación 3) y un puntero siguiente.

```
struct alumnos
{
               char nombre[30];
               char cal1[50];
               char cal2[50];
               char cal3[50];
               int matricula;
               struct alumnos *sig;
};

struct alumnos *inicio, *final, *temp, *temp2;
```

Para la programación de la aplicación se desarrollaron funciones que simplificaron la implementación. Estas funciones fueron:

Ingresar

Para esta función se pide al usuario realizar el ingreso de cuantos alumnos desea ingresar. Determinando el espacio de memoria que se dispondrá a ocupar.

Esta función cumple distintas etapas:

- asignación de memoria al nuevo elemento alumno
- rellenar el campo de datos del nuevo elemento
- el puntero siguiente del último elemento apunta hacia el nuevo elemento
- el puntero fin apunta hacia el nuevo elemento
- el puntero inicio no cambia
- el tamaño es incrementado

```
inicio=NULL; final=NULL; temp=NULL;
int n;
printf("\nALUMNOS A INGRESAR: ");
scanf("%d",&n);
for (int i=0;i<n;i++)
{
            temp=(struct alumnos *) malloc(sizeof(struct alumnos));
            if (temp==NULL)
            {
                    printf("NO HAY MEMORIA DISPONIBLE\n");
                    exit(1);
            }

            printf("\nMatricula: ");  cin>>temp->matricula;
            printf("Nombre: "); cin>>temp->nombre;
            printf("Calif. 1: "); cin>>temp->cal1;
            printf("Calif. 2: "); cin>>temp->cal2;
            printf("Calif. 3: "); cin>>temp->cal3;

            temp->sig=NULL;

            if (inicio==NULL) {
                    inicio=temp;
                    final=inicio;
            }
            else{
                    final->sig=temp;
                    final=temp;
            }
    }
```

Eliminar

Respecto a la función para eliminar un elemento alumno de la lista, primero se hace una búsqueda por matrícula para después apuntar el puntero siguiente del elemento actual de la búsqueda hacia la dirección del puntero siguiente del elemento a eliminar:

- liberar la memoria ocupada por el elemento eliminado
- actualizar el tamaño de la lista

```
temp=inicio;
temp2=inicio;
int a=0;
system("cls");
if (inicio==NULL){
                                    printf("Lista vacia\n");

                                    }
while((temp!=NULL)&&(a==0))
{
 if(aux==temp->matricula)
  {
    if(temp2==temp)
              inicio=inicio->sig;
    else
              temp2->sig=temp->sig;
    free(temp);
    printf("\nALUMNO ELIMINADO");
    a=1;
    }
    temp2=temp;
    temp=temp->sig;
}
```

Buscar

Para buscar dentro de la lista entera se tomara como elemento a buscar según la matricula ingresada por el usuario. Para esto en la lista hay que posicionarse al inicio, luego utilizando el puntero siguiente de cada elemento, la lista es recorrida desde 1ero al último elemento buscando coincidencias o mostrando alumno no existente, para ello se utilizaron banderas que señalaran cuando el elemento fuera encontrado.

```
temp=inicio;

int a=0;
while((temp!=NULL)&&(a==0))
{
 if(aux==temp->matricula)
  {
    printf("\n\t\t MATRICULA %d", temp->matricula);
    printf("\n\t\t NOMBRE: %s", temp->nombre);
    printf("\n\t\t CAL. 1: %s", temp->cal1);
    printf("\n\t\t CAL. 2: %s", temp->cal2);
    printf("\n\t\t CAL. 3: %s", temp->cal3);
    a=1;
    }
    temp=temp->sig;
}
```

Visualizar

Para mostrar la lista entera hay que posicionarse al inicio de la lista, luego utilizando el puntero siguiente de cada elemento la lista es recorrida del 1ero al último elemento. La condición para detener es dada por el puntero siguiente del último elemento que vale NULL.

```
temp=inicio;
int i=1;

printf("\n\n\t\t ALUMNOS \n");
printf("\n\t_____\n");

while (temp!=NULL)
{
          cout<<temp->matricula<<temp->nombre<<temp->cal1<<temp->cal2<<temp->cal3;
          temp=temp->sig;
          }
```

Modificar

Para esta función se necesitó hacer uso de la función buscar dentro de la lista tomando como referencia la matricula ingresada. Para esto en la lista hay que posicionarse al inicio, luego utilizando el puntero siguiente de cada elemento, la lista es recorrida desde 1ero al último elemento buscando coincidencias o mostrando alumno no existente.

En esta función se mostrará un submenú para modificar cualquiera de los datos ingresados (Matricula, Nombre, Calificación 1, Calificación 2 y/o Calificación 3). Para estas funciones es necesario ingresar un remplazo para cualquiera de los datos ingresado y utilizar un elemento temporal que localice el dato a modificar y lo guarde dentro de la lista.

```
temp=inicio;
int a=0, opcion, aux;
char aux1[30];
printf("\n\n INGRESE LA MATRICULA DEL ALUMNO A MODIFICAR DATOS: ");
scanf("%d", &aux);
while(1){
          do{

                    system("cls");
                    menu1();
                    opcion=getch();
                    }while(opcion <'1'|| opcion>'6');
          system("cls");
```

```
          switch(opcion){
                    case '1':
                         while((temp!=NULL)&&(a==0)){
                         if(aux==temp->matricula){
                              cout<<"Ingrese nueva matricula: ";
                              cin>>temp->matricula;
                              a=1;
                              }
                              temp=temp->sig;
                              }
                              getch();
                         break;
                    case '2':
                         while((temp!=NULL)&&(a==0)){
                         if(aux==temp->matricula){
                              cout<<"Ingrese nuevo nombre: ";
                              cin>>temp->nombre;
                              a=1;
                              }
                              temp=temp->sig;
                              }
                              getch();
                         break;
                    case '3':
                         while((temp!=NULL)&&(a==0)){
                         if(aux==temp->matricula){
                              cout<<"Ingrese nueva cal.1: ";
                              cin>>temp->cal1;
                              a=1;
                              }
```

```
                    break;
               case '4':
                    while((temp!=NULL)&&(a==0)){
                    if(aux==temp->matricula){
                         cout<<"Ingrese nueva cal.2: ";
                         cin>>temp->cal2;
                         a=1;
                         }
                         temp=temp->sig;
                         }
                    break;
               case '5':
                    while((temp!=NULL)&&(a==0)){
                    if(aux==temp->matricula){
                         cout<<"Ingrese nueva cal.3: ";
                         cin>>temp->cal3;
                         a=1;
                         }
                         temp=temp->sig;
                         }
                    break;
               case '6':
                    main();
                    break;
               }
}
```

Tercera fase (Gráficos).

En esta fase se detallara el proceso que se utilizó para el desarrollo gráfico de la aplicación.

En el modo gráfico existe una enorme cantidad de funciones que realizan desde la tarea más sencilla hasta la más compleja. Para trabajar el modo gráfico es necesario incluir la librería <graphics.h> como hacer uso de la BGI (Borlan Graphics Interphase).

A continuación se hace mención de las funciones utilizadas:

Función line()

Esta función se utiliza para dibujar una línea entre 2 puntos. Para ello, la función requiere 4 parámetros que representan las coordenadas (en pixeles) de los dos puntos que se desea unir mediante una línea recta.

Función initwindow()

Mediante esta función se ejecuta una ventana nueva en donde se nos permite visualizar el entorno gráfico que se está desarrollando. Recibe dos parámetros, el tamaño que se quiere en el eje 'x' (ancho) y el tamaño en el eje 'y' (alto).

Función setcurrentwindow()

La función cambia la ventana actual para todas las operaciones de gráficos a la ventana especificada. Este número de la ventana debe ser un número devuelto por la función initiwindow. La ventana actual es la ventana donde todas las demás operaciones de gráficos tendrán lugar.

Función cleardevice()

Esta función es usada para despejar una pantalla gráfica. Usa el color de fondo actual, como es establecido para rellenar la pantalla. La posición del cursor gráfico es la esquina superior izquierda de la pantalla - posición (0,0) - después de que la pantalla haya sido borrado.

Función closegraph()

La función desasigna toda la memoria asignada por el sistema de gráficos, y luego vuelve a la pantalla con el modo en que estaba antes.

Función clearviewport()

Esta función se utiliza para rellenar la pantalla actual del usuario con el color de fondo actual; es decir limpia la ventana gráfica actual.

Función outtextxy()

Esta función es usada para mostrar una cadena de caracteres. El argumento define la cadena de texto a ser mostrado. La cadena es mostrada en la posición descrita por los argumentos usando el color actual y fuente, dirección, valores, y justificaciones de texto.

Función getmaxy()

Esta función es usada para obtener la coordenada máxima de la pantalla en la dirección vertical.

Función getmaxx()

Esta función es usada para obtener la coordenada máxima de la pantalla en la dirección horizontal.

Función setfillstyle()

Esta función es usada para seleccionar una trama predefinida y un color de relleno. El argumento trama específica la trama predefinida, mientras que el argumento color especifica el color de relleno.

Función fillpoly()

Esta función es usada para crear un polígono rellenado.

Función settextstyle()

Esta función se emplea para especificar características del texto fuente que se muestra en el entorno gráfico, en la función se especifica la fuente, orientación, y el tamaño de los caracteres.

También fue preciso programar funciones que no se cuentan en la biblioteca de <graphics.h> y que son necesarios en la aplicación:

Función box()

Esta función es empleada para contener los botones de la aplicación. Esta función requiere 6 argumentos: Coordenadas en '*x*' e '*y*' iníciales, coordenada en '*x*' e '*y*' finales, relleno o estilo para el botón y finalmente el color.

Resultados

A continuación se presentan las interfaces finales del proyecto, presentando las funcionalidades y sus interfaces derivadas.

Figura 1. Visualización de la interfaz de inicio

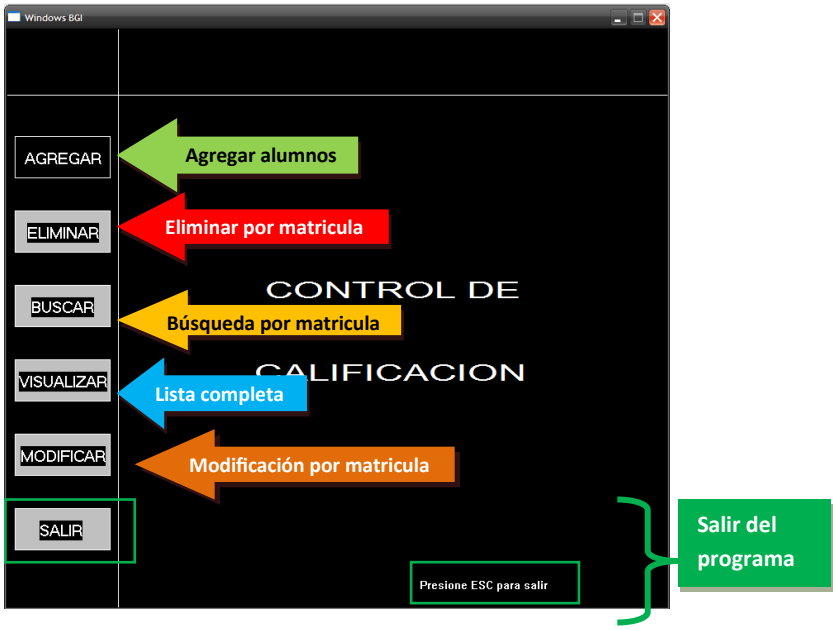


Figura 2. Visualización de la interfaz de inicio, mostrando cada una de sus partes.

Figura 3. Ingreso de alumnos a la aplicación.

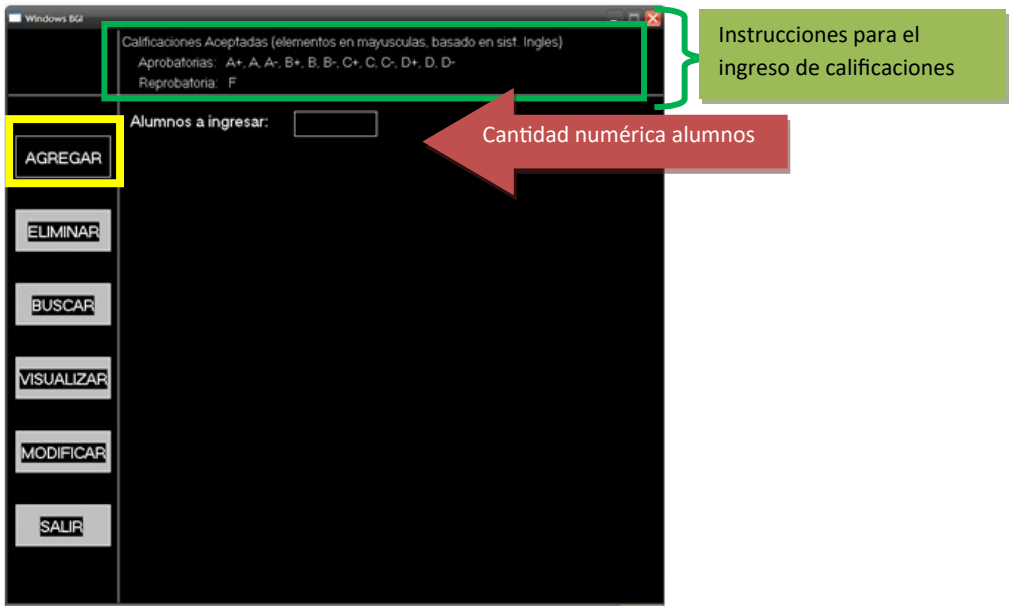


Figura 4. Ingreso de alumnos a la aplicación.

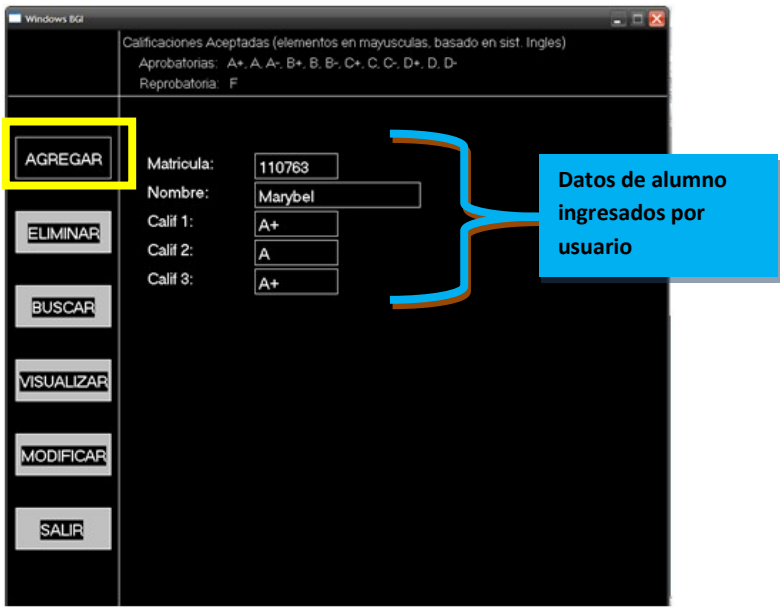


Figura 5. Ingreso de alumnos a la aplicación.

Figura 6. Ingreso de alumnos a la aplicación.

Figura 7. Eliminación de alumno de la aplicación.

Figura 8. Eliminación de alumno de la aplicación.

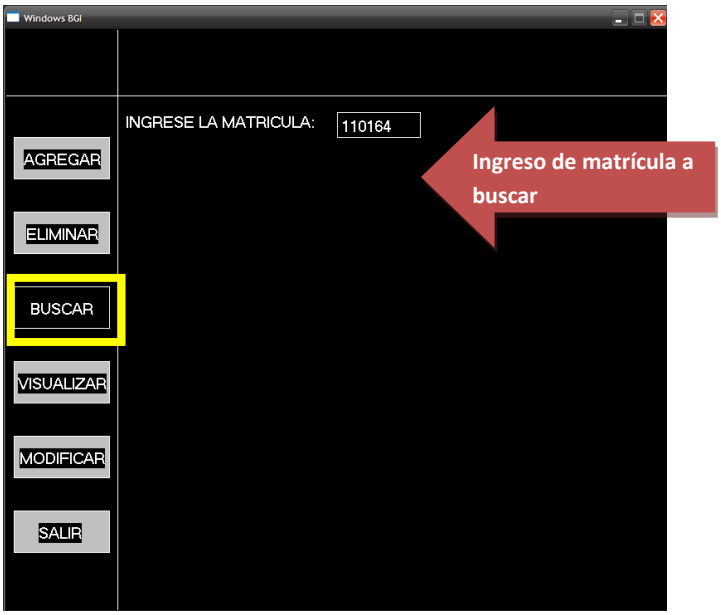


Figura 9. Búsqueda de alumno de la aplicación.

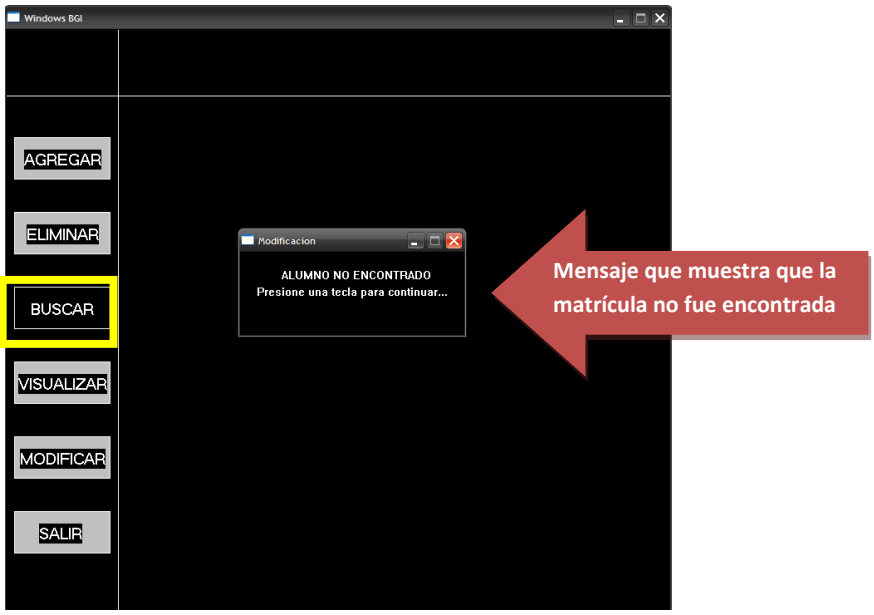


Figura 10. Búsqueda de alumno de la aplicación.

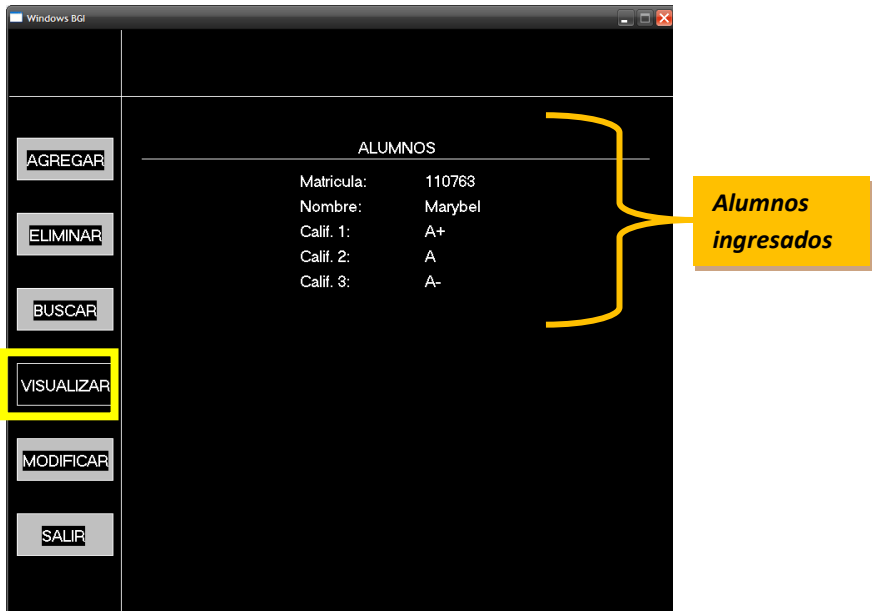


Figura 11. Visualización de todos los alumnos ingresados en la aplicación.

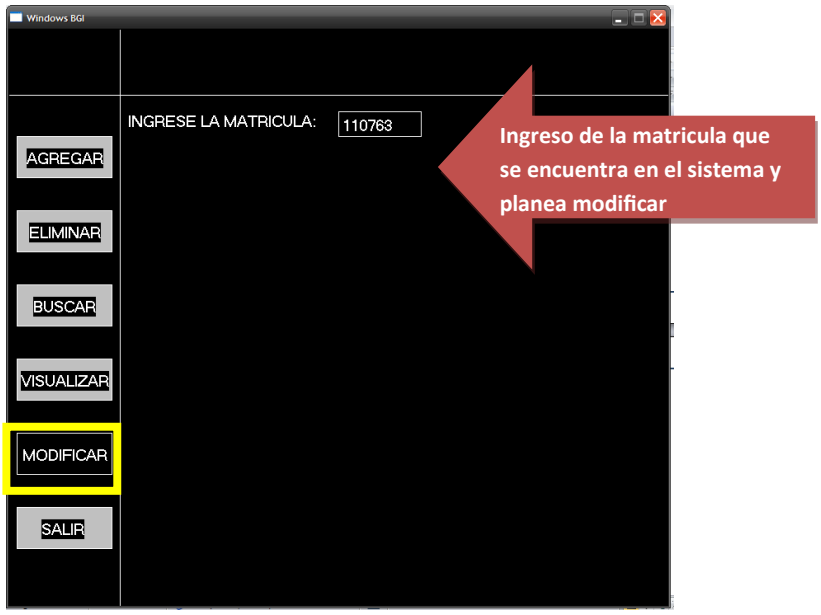


Figura 12. Visualización de todos los alumnos ingresados en la aplicación.

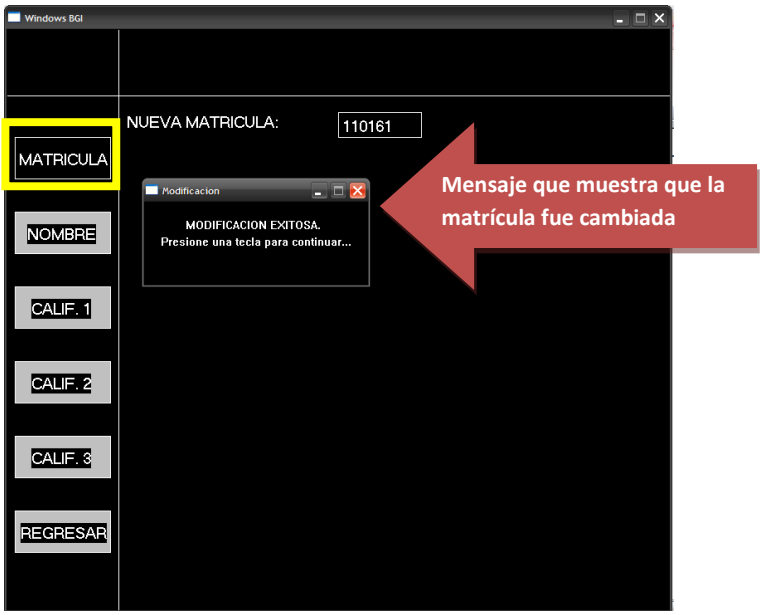


Figura 13. Visualización de todos los alumnos ingresados en la aplicación.

Figura 14. Visualización de todos los alumnos ingresados en la aplicación.

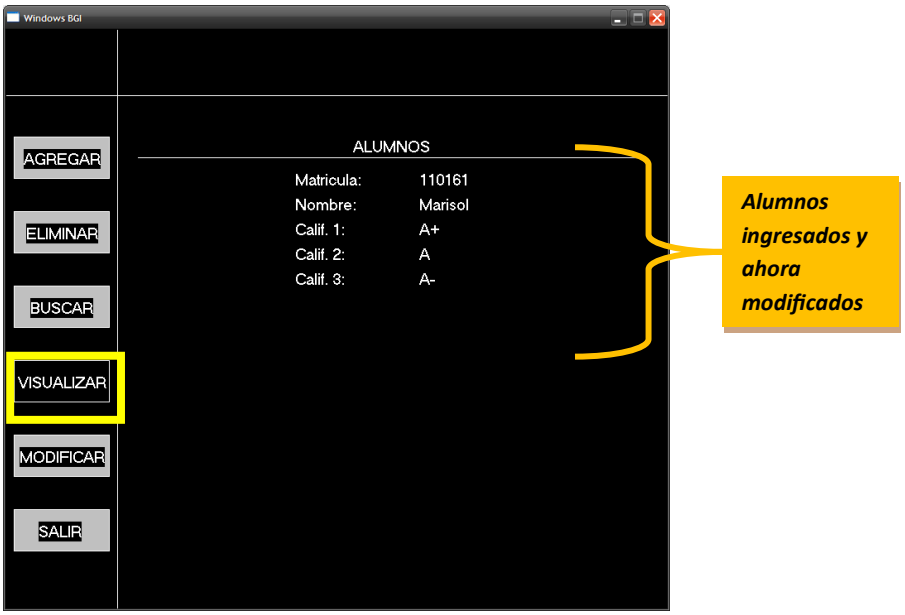


Figura 15. Visualización de todos los alumnos ingresados en la aplicación.

Conclusión

Como conclusión al unificar cada una de las fases en un solo programa, nos dimos cuenta que el proyecto que realizamos ha contribuido de manera muy importante para identificar y resaltar los puntos que se pretendieron cubrir y considerar para llevar a cabo la implementación exitosa de este proyecto. Nos deja muchas cosas importantes que reflexionar y muchas otras para llevar a cabo una buena implementación de proyectos a futuro.

Sin embargo, a pesar del éxito que se tuvo, hay cuestiones que pueden mejora a futuro para el uso pleno de esta aplicación, como lo son:

- Gráficos más amigables
- Más funcionalidades del sistema
- Portabilidad a otros sistemas

Dentro de los puntos que consideramos tienen más importancia dentro de un proyecto de esta naturaleza son el detectar cuáles son las necesidades reales de las personas que trabajan día a día con los sistemas, que los procesos operativos de una empresa se apeguen a la realidad del trabajo diario y no sean un obstáculo, que se involucre a los usuarios en el proceso de implementación de los sistemas de manera que se sepa que es lo que ellos esperan y qué es lo que no esperan de él.