



Universidad Europea

UNIVERSIDAD EUROPEA DE MADRID

ESCUELA DE ARQUITECTURA, INGENIERÍA Y DISEÑO

GRADO EN INGENIERÍA INFORMÁTICA

Gestión de la memoria



*Santiago Moreno
Guillermo Aos
Ismael Bouatman
Javier Franco*

ÍNDICE

1.Introducción	3.
1.1 Pila de usuario.....	3.
1.2 Mochila.....	3.
1.3 Calculadora Matrices.....	3.
1.4 Sistema alta/baja usuario.....	3.
1.5 Juego adivinar números	4.
2.Requisitos.....	4.
2.1 Requisitos Pila de usuario.....	4.
2.2 Requisitos Mochila.....	5.
2.3 Requisitos Calculadora Matrices.....	5.
2.4 Requisitos Sistema alta/baja usuario.....	5.
2.5 Requisitos Juego adivinar números.....	5.
3.Explicación del código.....	6.
3.1 Explicación del código Pila de usuario.....	6.
3.2 Explicación del código Mochila	8.
3.3 Explicación del código Calculadora Matrices.....	9.
3.4 Explicación del código Sistema alta/baja usuario	12.
3.5 Explicación del código Juego adivinar números	16.

4.Manual de usuario.....	17.
4.1 Manual de usuario Pila de usuario.....	17.
4.2 Manual de usuario Mochila.....	18.
4.3 Manual de usuario Calculadora Matrices.....	19.
4.4 Manual de usuario Sistema alta/baja usuario.....	21.
4.5 Manual de usuario Juego adivinar números.....	23.
5.Conclusiones.....	24.

1. Introducción

Se ha planteado la resolución de 5 ejercicios, donde cada uno tiene como finalidad aplicar todos los conocimientos adquiridos en clase durante la unidad formativa 2.

1.1 Pila de usuario

En este ejercicio se necesita programar un pila de usuario donde esta instancia sus valores en la misma pidiéndole al usuario esos valores por teclado, además de poder borrar cualquier posición de la pila.

1.2 Mochila

Se tendrá que programar una mochila que pueda almacenar cualquier tipo de archivo en c++, además de un menú donde se puedan gestionar esos archivos de la mochila cómo es añadir o borrar un elemento de la mochila

1.3 Calculadora Matrices

Crearemos una calculadora de matrices en c + +, donde se podrán realizar todas las operaciones básicas como son suma, resta y multiplicación, donde tanto las matrices de la operación como el resultado se tendrán que almacenar en el free store.

1.4 Sistema alta/baja usuarios

Se creará un sistema de alta y baja de usuarios donde por pantalla se le pedirá al usuario añadir tanto su nombre y apellido y con la ayuda de un menú se podrá dar tanto de baja como de alta a ese usuario creado por pantalla.

1.5 Juego adivinar números

Se programara un juego sencillo donde el usuario tendrá que adivinar un número aleatoria del 1 al 100 y el código le indicará si el número introducido es menor o mayor al que hay que intentar adivinar, donde finalmente cuando el usuario adivine el número a adivinar se le enseñara el número de intentos que le ha costado lograr adivinar el número

2. Requisitos

- Gestión de memoria
- Uso de excepciones
- Resolución de varios problemas
- Perfecto uso de los conocimientos adquiridos en clase

2.1 Requisitos Pila de memoria

- Solicite al usuario por consola un valor que almacenar en la pila de usuario.
- Permite ver que valores se almacenan en la pila del usuario en las diferentes
- posiciones.
- Debe poder eliminarse el valor de cualquier posición de la pila de usuario.
- El código generado debe ser a prueba de errores. Para ello implementa los métodos necesarios para poder mantener el proceso de ejecución.

2.2 Requisitos Mochila

- Almacenar objetos
- Menú de gestión de objetos
- Eliminar objeto deseado
- Gestión de objetos eficiente

2.3 Requisitos Calculadora matrices

- Realizar las operaciones: suma, resta y multiplicación
- Almacenar las matrices en la freestore
- Almacenar la matriz resultante
- Extra

2.4 Requisitos Sistema alta/baja usuarios

- Crear un usuario con los campos: Nombre, Apellidos y número de usuario
- ID de los usuarios no se sustituyen
- Para dar de baja, debe eliminarse usando el destructor del objeto usuario
- Mostrar por pantalla el usuario destruido

2.5 Requisitos Juego adivinar números

- El ordenador debe generar un número aleatorio entre 1 y 100
- Determinar si el número introducido por el usuario es mayor o menor que el número correcto
- Controlar excepciones ocasionadas por el usuario
- Técnicas necesarias para mejorar las eficiencia

3. Explicación del código:

3.1 Explicación del código Pila de usuario

Añadimos los includes necesarios.

Creamos la clase STACK, definimos los diferentes tipos de datos dentro de la estructura node.

```
#include <iostream>
#include <stack>
#include <conio.h>
#include <stdlib.h>
using namespace std;

class STACK {
    struct node{
        string data;
        node* link;
    }*TOP;
public:
    STACK(){
        TOP=NULL;
    }

    void PUSH(){
        struct node* p= new node;
        cout<<"Inserte un elemento: ";
        cin>>p->data;
        p->link=NULL;
        if(TOP==NULL){
            TOP=p;
        }
        else{
            p->link=TOP;
            TOP=p;
        }
    };

    void POP(){
        if(TOP==NULL){
            cout<<"stack vacio"<<endl;
        }
        else{
            struct node*t=new node;
            t=TOP;
            cout<<"El elemento extraido es: "<<t->data<<endl;
            TOP=TOP->link;
            delete t;
        }
    };
};
```

Función Push, crea y hace una nueva dirección de nodo para apuntar a TOP y hacer el nuevo nodo TOP

Al principio de esta función reservamos memoria para p.

Guardamos los valores del usuario en p

Seteamos la dirección de P-link a NULL

Creamos una condición, si el stack está vacío, entonces p es TOP. Por otra parte si hay un valor en el stack hacemos que el nuevo nodo apunte arriba del nodo y que TOP apunte a p.

Función Pop, elimina un nodo y hace que el nodo anterior sea TOP.

Si no hay ningún elemento en el stack, TOP

es NULL, a continuación creamos un nodo temporal y hacemos que t y TOP apunten al mismo nodo.

Finalmente TOP apunta al próximo nodo y t se elimina.

```
void display(){
    struct node*t =new node;
    t=TOP;
    cout<<"Valores dentro de la pila: "<<endl;
    while(t!=NULL){
        cout<<t->data<<endl;
        t=t->link;
    }
}

};

int main(){
    int choice;
    STACK obj1;
    do{
        cout<<"-----"<<endl;
        cout<<"Dime una opcion"<<endl;
        cout<<"Menu"<<endl;
        cout<<"\t1.Insertar"<<endl;
        cout<<"\t2.Extraer"<<endl;
        cout<<"\t3.Imprimir"<<endl;
        cout<<"\t4.EXIT"<<endl;
        cout<<"-----"<<endl;
        cin>>choice;
        switch (choice){
            case 1: obj1.PUSH();break;
            case 2: obj1.POP();break;
            case 3: obj1.display();break;
            default: "Opcion no valida";
            return(0);
        }
    }while(choice!=4);
    exit(-1);
};
```

La función display, simplemente es una función que nos permite ver imprimir todos los valores que están dentro de la pila.

En el main creamos un menú que nos permite acceder a las diferentes funciones además de salirnos del programa.

3.2 Explicación del código Mochila

```

6   vector<string> mochila;
7   vector<string> *pmochila=&mochila;
8   string r;
9   string v;
0   string k;
1   bool c=false;

```

Declaramos el vector mochila y almacenamos en p*mochila la dirección de memoria del vector

La estructura principal del código es un do while

La primera condición que ponemos es que cuando le preguntamos al usuario ¿Quieres meter, eliminar o mostrar?, nos pone una cosa diferente le volvamos a preguntar. Esto sucede gracias a que instanciamos un booleano en false y si el usuario pone algo diferente el booleano es true por lo tanto cumple la condición en el while y se vuelve a ejecutar.

A continuación si el usuario introduce "meter" el programa mediante el

push_back introduce en el vector lo que ha puesto el usuario

Por otro lado si pone "eliminar" al principio le mostramos los objetos que hay y le pedimos al usuario que ponga el número asociado al objeto que desea eliminar. Tras esto gracias a la función erase eliminamos el objeto del vector

```

2   do {
3       cout << "Quieres meter, eliminar un objeto o mostrar o salir \n";
4       cin >> r;
5
6       if("meter" != r && "eliminar" != r && r != "mostrar" && r!="salir") {
7           c = true;
8           cout << "Solo se aceptan: meter, eliminar o mostrar\n";
9       }
0
1       if (r == "meter") {
2           cout << "Inserta el objeto que quieras meter\n";
3           cin >> k;
4           mochila.push_back(k);
5       }
6       else if (r == "eliminar") {
7           for (int i = 0; i < mochila.size(); i++) {
8               cout << i << ": " << mochila[i] << "\n";
9           }
0           cout << "Que objeto desea eliminar, introduzca un numero\n";
1           cin >> v;
2           cout << "Se va a borrar " << mochila[stoi(v)] << endl;
3           mochila.erase(mochila.begin() + stoi(v));
4           cout << "El objeto ha sido eliminado" << endl;
5       }
6       else if (r == "mostrar") {
7           for (int i = 0; i < mochila.size(); i++) {
8               cout << mochila[i] << "\n";
9           }
0       }
1       else if (r == "salir") {
2           mochila.clear();
3           exit(-1);
4       }
5
6   } while (r == "meter" || r == "eliminar" || r == "mostrar" || c == true);
7   mochila.clear();
8   };

```

Si el usuario introduce “mostrar” mediante el for y el cout le imprimimos el contenido del vector donde finalmente destruimos el vector mochila.

Por último si le da a la opción de salir el programa se cierra

3.3.Explicación del código Calculadora Matrices

```

1  #include <iostream>
2  using namespace std;
3  void Rellenar( int **M, int fil, int col )
4  {
5      cout << "\nRellenar la matriz:\n";
6      for( int i = 0; i < fil; i++ ){
7          for( int j = 0; j < col; j++ ){
8              cout << "Matriz[" << i << "][" << j << "]: ";
9              cin >> M[i][j];
10         }
11     }
12 }
13 void imprimir ( int **M, int fil, int col )
14 {
15     for( int i = 0; i < fil; i++ ){
16         cout << "\n| ";
17         for( int j = 0; j < col; j++ )
18             cout << M[i][j] << " ";
19         cout << " |";
20     }
21     cout << endl;
22 }
23 }
```

Añadimos el include necesario y el using namespace std para poder usar los cout y cin para tanto imprimir por pantalla como para guardar el dato en la variable necesaria. Realizamos dos funciones que se va utilizar en todas las operaciones que son la de Rellenar(Con esta función vamos recorriendo la matriz mientras la va rellenando valor por valor para luego poder operar) e Imprimir (Esta función recorre la matriz añadiendo “|” al final y al principio de cada fila y “\n” para que a la hora de imprimirla se vea bien .

En la función suma nos piden las filas y las columnas , reserva en el free store las matrices operacionales y la matriz resultante ,posteriormente rellenamos la matriz y realizamos un for que nos permite recorrer las matrices y nos las suma ,imprimiendo el resultado .

```

void suma()
{
    int f, c;

    cout << "\nFilas: "; cin >> f;
    cout << "Columnas: "; cin >> c;

    int** A = new int*[f];
    for( int i = 0; i < f; i++ )
        A[i] = new int[c];

    int** B = new int*[f];
    for( int i = 0; i < f; i++ )
        B[i] = new int[c];

    int** C = new int*[f];
    for( int i = 0; i < f; i++ )
        C[i] = new int[c];

    cout << "\nDatos de la matriz A: ";
    Rellenar( A, f, c );

    cout << "\nDatos de la matriz B: ";
    Rellenar( B, f, c );

    for( int i = 0; i < f; i++ )
        for( int j = 0; j < c; j++ )
            C[i][j] = A[i][j] + B[i][j];
    cout << "\nSuma de las matrices (A+B):\n";
    imprimir( C, f, c );
}
```

```
void resta()
{
    int f, c;

    cout << "\nFilas: "; cin >> f;
    cout << "Columnas: "; cin >> c;

    int** A = new int*[f];
    for( int i = 0; i < f; i++ )
        A[i] = new int[c];

    int** B = new int*[f];
    for( int i = 0; i < f; i++ )
        B[i] = new int[c];

    int** C = new int*[f];
    for( int i = 0; i < f; i++ )
        C[i] = new int[c];

    cout << "\nDatos de la matriz A: ";
    Rellenar( A, f, c );

    cout << "\nDatos de la matriz B: ";
    Rellenar( B, f, c );

    for( int i = 0; i < f; i++ )
        for( int j = 0; j < c; j++ )
            C[i][j] = A[i][j] - B[i][j];

    cout << "\nResta de las matrices (A-B):\n";
    imprimir( C, f, c );
}
```

La función resta hace lo mismo que la función suma pero en lugar de recorrer las matrices para sumar, lo hace para restar, e imprime el resultado

La función multiplicación nos pide que pongamos las filas y columnas de las dos matrices para ver si se puede realizar la multiplicación o no, posteriormente nos reserva un espacio en el free store para las 2 matrices operacionales y la resultante.

Posteriormente vamos rellenando las matrices y comprobamos que se puede realizar la multiplicación si la Fila de la matriz A es igual de tamaño que la columna de la Matriz B, si es así recorremos las matrices, realizamos la correspondiente multiplicación, y la imprimimos. Si no es posible realizar la multiplicación por las dimensiones, mandamos un mensaje.

```
void multiplicacion()
{
    int filaA, colA, filaB, colB;
    cout << "\nFilas de la matriz A: "; cin >> filaA;
    cout << "Columnas de la matriz A: "; cin >> colA;

    int** A = new int*[filaA];
    for( int i = 0; i < filaA; i++ )
        A[i] = new int[colA];

    Rellenar( A, filaA, colA );

    cout << "\nFilas de la matriz B: "; cin >> filaB;
    cout << "Columnas de la matriz B: "; cin >> colB;

    int** B = new int*[filaB];
    for( int i = 0; i < filaB; i++ )
        B[i] = new int[colB];

    Rellenar( B, filaB, colB );

    int** C = new int*[filaA];
    for( int i = 0; i < filaA; i++ )
        C[i] = new int[colA];

    if( colA == filaB ){
        for( int i = 0; i < filaA; ++i ){
            for( int j = 0; j < colB; ++j ){
                C[i][j] = 0;
                for( int z = 0; z < colA; ++z )
                    C[i][j] += A[i][z] * B[z][j];
            }
        }
    }
}
```

```

}

cout << "\nMultiplicacion de las matrices (A*B):\n";
imprimir( C, filaA, colB );

}else{
    cout << "\n    EL NUMERO DE COLUMNAS DE LA MATRIZ A DEBE COINCIDIR CON EL DE FILAS DE LA MATRIZ B";
}
```

Esta es la función de la transpuesta en la cual pedimos que nos introduzca de que dimensiones es la matriz y sus valores, una vez escritos, el primer for nos dibuja la matriz original y el segundo nos dibuja como quedaría su transpuesta

```
int main()
{
    int opcion;

    do{
        cout << "\n    CALCULADORA DE MATRICES"
        << "\n*****"
        << "\n1.- Suma"
        << "\n2.- Resta"
        << "\n3.- Multiplicacion"
        << "\n4.- Transpuesta "
        << "\n*****"
        << "\nSELECCIONA UNA OPCION (1 - 4): ";

        cin >> opcion;
        switch( opcion ){
            case 1:
                suma();
                break;

            case 2:
                resta();
                break;

            case 3:
                multiplicacion();
                break;

            case 4:
                Transpuesta();
                break;

            case 5:
                cout << "\nFIN DEL PROGRAMA\n\n";
                break;

            default:
                cout << "\n\nOPCION NO VALIDA\n\n";
                break;
        }
    }while( opcion != 5 );
}
```

3.4. Explicación del código Sistema alta/bajas usuario

```
#include <iostream>
#include <stdlib.h>
#include <string.h>
#include <vector>
#include <cstdlib>
#include <conio.h>
#include <stack>
#include <stdio.h>
#include <string.h>
#include <algorithm>
#include <iterator>
using namespace std;
```

Añadimos todos los include que nos servirán tanto para crear nuestro número aleatorio como el uso de excepciones para mejorar y depurar nuestro código.

Añadimos también el using namespace std para poder usar los cout y cin para tanto imprimir por pantalla como para pedirle un dato al usuario.

Luego empezamos programando la clase usuario de nuestro programa donde primero creamos los atributos de la clase usuario como son el nombre y el apellido en el private, mientras que en la parte del público instanciamos todas las funciones que están relacionadas con nuestra clase usuario

```
class Usuario{
private:
    string nombre;

    string apellido;

    int numero_usuario;

public:
    void imprimir_usuario();
    void imprimir_altas_usuario();
    string getnombre();
    string getapellido();
    int getnumero_usuario();
    void setnombre(string nombre_new);
    void setapellido(string apellidos_new);
    void setnumero_usuario(int numero_usuario_new);
    Usuario(string nombre, string apellido, int numero_usuario);
};

Usuario::Usuario(string nombre, string apellido, int numero_usuario){
    this->nombre= nombre;
    this->apellido=apellido;
    this->numero_usuario=numero_usuario;
};
```

```
string Usuario::getnombre(){
    return nombre;
};
string Usuario::getapellido(){
    return apellido;
};
int Usuario::getnumero_usuario(){
    return numero_usuario;
};
void Usuario:: setnumero_usuario(int numero_usuario_new){
    numero_usuario=numero_usuario_new;
};
void Usuario:: setnombre(string nombre_new){
    nombre=nombre_new;
};
void Usuario:: setapellido(string apellido_new){
    apellido=apellido_new;
};
void Usuario::imprimir_usuario(){
    cout<<"Nombre del usuario: "<<nombre<<" Apellido del usuario: "<<apellido<<endl;
};
void Usuario::imprimir_altas_usuario(){
    cout<<"Nombre del usuario: "<<nombre<<" Apellido del usuario: "<<apellido<<" Identificador del usuario: "<<numero_usuario<<endl;
};
```

Luego crearemos todos los getters y setters necesarios para poder usar la información de nuestros atributos ya programados

```
class Registro{
private:
    vector<Usuario> usuarios;
    vector<Usuario> *pusuarios=&usuarios;
    vector<Usuario> altas_usuarios;
    vector<Usuario> *paltas_usuarios=&usuarios;
    int choose_baja;
public:
    int getchoose_baja();
    void setchoose_baja(int choose_baja_new);
    ~Registro(){
        delete(&altas_usuarios[choose_baja]);
    };
    void anadir_usuario();
    void dar_alta();
    void dar_baja();
    void menu();
    void reajuste_vector();
};
```

Continuamos programando la clase registro que nos servirá para almacenar toda la información en el free store además de poder almacenar toda la información como instanciar las funciones de dar de alta y de baja a un usuario.

Igual que antes en el public añadimos los atributos de la clase Registro y en el public las funciones.

Como hemos hecho antes seguimos instanciando los getters y setter pero en este caso los de la clase Registro.

Después ya empezamos programando las funciones que tendrá nuestra aplicación, en este caso decidimos empezar por la de añadir un usuario donde

pedimos al usuario por pantalla que escriba tanto un nombre como un apellido y los guarda en el vector usuarios

```
int Registro::getchoose_baja(){
    return choose_baja;
};

void Registro::setchoose_baja(int choose_baja_new){
    choose_baja=choose_baja_new;
};

void Registro::anadir_usuario(){
    string nombre_recibido;
    string apellido_recibido;
    int numero_usuario_recibido=0;
    cout<<"Dime tu nombre"<<endl;
    cin>>nombre_recibido;
    cout<<"Dime tu apellido"<<endl;
    cin>>apellido_recibido;
    Usuario voidUsuario( nombre: nombre_recibido, apellido: apellido_recibido, numero_usuario: numero_usuario_recibido);
    voidUsuario.imprimir_usuario();
    usuarios.push_back(voidUsuario);
    cout<<"-----"<<endl;
    cout<<"Usuario creado: "<<endl;
};
```

```
void Registro::dar_alta() {
    cout<<"Dime el numero de usuario quieres dar de alta"<<endl;
    int choose;
    //cout << "Size" << usuarios.size()<<endl;
    for(int i = 0; i < usuarios.size(); i++){
        cout<<"<i<<".-";
        usuarios[i].imprimir_usuario();// como pollas se pone esto bien
        cout<<"<<endl;
    };
    cin>>choose;
    altas_usuarios.push_back(usuarios[choose]);
    altas_usuarios[altas_usuarios.size()].setnumero_usuario( numero_usuario_new: altas_usuarios.size());
    cout<<"Usuario dado de alta con éxito"<<endl;
};

void Registro::dar_baja(){
    cout<<"Dime el identificador del usuario que quieres dar de baja"<<endl;

    for(int k = 0; k < usuarios.size(); k++){
        cout<<"<k<<".-";
        usuarios[k].imprimir_usuario();
        cout<<"<<endl;
    };
    cin>>choose_baja;
    usuarios.erase( position: usuarios.begin()+choose_baja);
    reajuste_vector();
    cout<<"Usuario dado de baja con éxito"<<endl;
};
```

Continuamos con las funciones de dar alta y de dar baja que consisten en recorrer el vector usuarios y ya el usuario es quien decide qué usuario se le da de alta en la aplicación como consecuencia ese usuario será dado de alta en la aplicación.

En la función dar de baja, el código hace lo mismo recorre el vector pero en este caso el usuario decide qué usuario dar de baja.

```
void Registro::menu(){

    int choose;
    do{
        cout<<"Elige una opcion"<<endl;
        cout<<"-----"<<endl;
        cout<<"Opcion 1-Anadir usuario"<<endl;
        cout<<"Opcion 2-Dar de alta un usuario"<< endl;
        cout<<"Opcion 3-Dar de baja un usuario"<< endl;
        cout<<"Opcion 4-Salir"<<endl;
        cout<<"-----"<<endl;
        cin>>choose;

        switch (choose){
            case 1:
                anadir_usuario();
                break;
            case 2:
                dar_alta();
                break;
            case 3:
                dar_baja();
                break;
            default: "Opcion no valida";
                exit( Code: -1);
        };
    }

    while(choose!=4);
    altas_usuarios.~vector<Usuario>();
    usuarios.~vector<Usuario>();
    exit( Code: -1);
}
```

En esta parte del código se encuentra un sencillo menú para gestionar la aplicación, donde hay 4 opciones en el menú donde se podrá primero de todo añadir un usuario, ya teniendo varios usuarios a estos se les podrá dar de alta y de baja y para finalizar estará la cuarta opción que es la función salir donde se saldrá del menú

Para terminar tenemos el main donde se terminará ejecutando todo nuestro código explicado anteriormente.

```
int main() {
    Registro Registrobd;
    Registrobd.menu();
    return 0;
}
```


3.5. Explicación del código Juego adivinar números

```

1  #include <iostream>
2  #include <time.h>
3  #include <stdlib.h>
4  #include <conio.h>
5
6  using namespace std;
7  void limpiar_pantalla()
8  {
9      #ifdef _WIN32
10         system( Command: "cls");
11     #else
12         ...
13     #endif
14 }
```

Aquí en el main programamos el juego en sí que no tiene ninguna dificultad a priori donde instanciamos tanto el número que pediremos por pantalla como el número random de 1 al 100 y con la ayuda de unos ifs y else tendremos hecho nuestro juego.

Para finalizar para purificar el código y evitar problemas o excepciones en nuestro código nos apoyamos en la función anteriormente programada limpiar_pantalla y de los getchar y getche. Con todo esto tendríamos finalizado el código sencillo para crear nuestro juego de adivinar un número .

Empezamos añadiendo todos los include que nos servirán tanto para crear nuestro número aleatorio como el uso de excepciones para mejorar y depurar nuestro código.

Añadimos también el using namespace std para poder usar los cout y cin para tanto imprimir por pantalla como para pedirle un dato al usuario.

Luego programamos la función limpiar pantalla que ya ha sido explicada con anterioridad en códigos anteriores que nos servirá para la corrección de excepciones.

```

int main() {
    int numero;
    int n=100;
    int a;
    int intentos = 0;
    srand( Seed: time( Time: NULL));
    numero = 1 + rand() % (100 - 1 + 1);
    cout << "Bienvenidos al juego de adivinar numero" << "Escriba un numero del 1 al 100: " << endl;
    for (int i = 1; i <= n; i++) {
        cin >> a;
        if (a>=1 && a<=100)
        {
            cout << "Escriba un numero del 1 al 100: " << endl;
            if (numero>a){
                intentos++;
                cout<<"El numero introducido es menor, intentalo de nuevo."<<endl;
            }else if(numero<a){
                intentos++;
                cout<<"El numero introducido es mayor, intentalo de nuevo."<<endl;
            }else{
                intentos++;
                cout<<"Enhorabuena has adivinado el numero en : "<<intentos<<" intentos."<<endl;
                n = -1;
            }
        }
    }
    else{
        cout<<"Has introducido un valor incorrecto, vuelve a empezar"<<endl;
        cout<<"Introduzca cualquier letra para poder finalizar el programa."<<endl;
        getche();
        limpiar_pantalla();
        exit( Code: -1);
    }
}

getchar();
return 0;
}
```

4. Manual de usuario

4.1 Manual de usuario Pila de Usuario

```
-----
Dime una opcion
Menu
    1.Insertar
    2.Extraer
    3.Imprimir
    4.EXIT
-----
```

Este es el menú principal, necesario para saber que opción elegir, siempre nos aparecerá una vez finalizado el uso de la función que hayamos elegido.

```
-----
1
Inserte un elemento: 7
-----
Dime una opcion
Menu
    1.Insertar
    2.Extraer
    3.Imprimir
    4.EXIT
-----
```

Elegimos la opción 1, e insertamos el valor que queramos, en este caso el 7.

```
-----
3
Valores dentro de la pila:
7
-----
Dime una opcion
Menu
    1.Insertar
    2.Extraer
    3.Imprimir
    4.EXIT
-----
```

A continuación elegimos la opción Imprimir y se nos imprimen todos los valores dentro de la pila.

Finalmente elegimos la opción de extraer y nos salimos del programa gracias a la opción EXIT.

```
-----
2
El elemento extraido es: 7
-----
Dime una opcion
Menu
    1.Insertar
    2.Extraer
    3.Imprimir
    4.EXIT
-----
```

4.2 Manual de Usuario Mochila

```
Quieres meter, eliminar un objeto o mostrar
roca
Solo se aceptan: meter, eliminar o mostrar
Quieres meter, eliminar un objeto o mostrar
```

Al principio el programa nos pregunta si queremos meter, eliminar o mostrar

Si ponemos algo diferente nos lo vuelve a preguntar

```
Quieres meter, eliminar un objeto o mostrar
meter
Inserta el objeto que quieras meter
tijera
Quieres meter, eliminar un objeto o mostrar
meter
Inserta el objeto que quieras meter
papel
Quieres meter, eliminar un objeto o mostrar
```

Ponemos meter y nos solicita un objeto, lo insertamos y nos vuelve a mostrar las tres opciones

```
Quieres meter, eliminar un objeto o mostrar
eliminar
0: tijera
1: papel
Que objeto desea eliminar, introduzca un numero
0
Se va a borrar tijera
El objeto ha sido eliminado
Quieres meter, eliminar un objeto o mostrar
```

Ponemos eliminar y nos muestra la lista de los objetos que tenemos, luego nos pide introducir el número asociado al objeto que deseamos eliminar Finalmente nos pone que va a eliminar el objeto y elimina

```
Quieres meter, eliminar un objeto o mostrar
mostrar
papel
Quieres meter, eliminar un objeto o mostrar
```

A continuación nos pone los objetos que hay almacenados

```
Quieres meter, eliminar un objeto o mostrar o salir
salir
PS C:\Users\zrihn\3D Objects\PROYECTO\Franco>
```

Si el usuario introduce salir el programa se cerrará

4.3 Manual de usuario Calculadora de Matrices

```

          CALCULADORA DE MATRICES
*****
1.- Suma
2.- Resta
3.- Multiplicacion
4.- Transpuesta
*****
SELECCIONA UNA OPCION (1 - 4):

```

Nos aparece un menú de 4 opciones de una calculadora de matrices, el cual tenemos que elegir qué operación queremos que haga

Si tecleamos un 1, nos pedirá primero de cuantas filas y columnas son las matrices que quieres sumar, y una vez las hayamos escrito ,tendremos que escribir los valores de nuestras matrices,al finalizar nos mostrara la suma de nuestras matrices y el menú de nuevo

```
SELECCIONA UNA OPCION (1 - 4): 1
```

```

Filas: 2
Columnas: 2

```

```

Datos de la matriz A:
Rellenar la matriz:
Matriz[0][0]: 2
Matriz[0][1]: 3
Matriz[1][0]: 4
Matriz[1][1]: 5

```

```

Datos de la matriz B:
Rellenar la matriz:
Matriz[0][0]: 1
Matriz[0][1]: 6
Matriz[1][0]: 8
Matriz[1][1]: 9

```

```
Suma de las matrices (A+B):
```

```

| 3 9 |
| 12 14 |

```

```
SELECCIONA UNA OPCION (1 - 4): 2
```

```

Filas: 2
Columnas: 2

```

```

Datos de la matriz A:
Rellenar la matriz:
Matriz[0][0]: 10
Matriz[0][1]: 12
Matriz[1][0]: 13
Matriz[1][1]: 14

```

```

Datos de la matriz B:
Rellenar la matriz:
Matriz[0][0]: 22
Matriz[0][1]: 2
Matriz[1][0]: 14
Matriz[1][1]: 87

```

```
Resta de las matrices (A-B):
```

```

| -12 10 |
| -1 -73 |

```

```

          CALCULADORA DE MATRICES
*****
1.- Suma
2.- Resta
3.- Multiplicacion
4.- Transpuesta
*****
SELECCIONA UNA OPCION (1 - 4):

```

Si tecleamos un 2, de nuevo nos pedirá las filas y columnas de las matrices que queramos restar y tendremos que rellenarla, una vez escrito todos los valores, nos saldrá la solución de la resta y de nuevo el menú.

Si tecleamos un 3, nos pedirá las dimensiones de las dos matrices que queramos multiplicar y tendremos que rellenarla, si cumple con las condiciones, nos mostrará el resultado de la multiplicación y de nuevo el menú

```
SELECCIONA UNA OPCION (1 - 4): 3
Filas de la matriz A: 2
Columnas de la matriz A: 2

Rellenar la matriz:
Matriz[0][0]: 1
Matriz[0][1]: 3
Matriz[1][0]: 3
Matriz[1][1]: 5

Filas de la matriz B: 2
Columnas de la matriz B: 3

Rellenar la matriz:
Matriz[0][0]: 1
Matriz[0][1]: 2
Matriz[0][2]: 3
Matriz[1][0]: 4
Matriz[1][1]: 1
Matriz[1][2]: 2

Multiplicacion de las matrices (A*B):

| 13 5 9 |
| 23 11 19 |
```

```
SELECCIONA UNA OPCION (1 - 4): 3
Filas de la matriz A: 2
Columnas de la matriz A: 2

Rellenar la matriz:
Matriz[0][0]: 3
Matriz[0][1]: 2
Matriz[1][0]: 1
Matriz[1][1]: 2

Filas de la matriz B: 3
Columnas de la matriz B: 2

Rellenar la matriz:
Matriz[0][0]: 2
Matriz[0][1]: 3
Matriz[1][0]: 2
Matriz[1][1]: 3
Matriz[2][0]: 4
Matriz[2][1]: 1

EL NUMERO DE COLUMNAS DE LA MATRIZ A DEBE COINCIDIR CON EL DE FILAS DE LA MATRIZ B
CALCULADORA DE MATRICES
```

Si no cumple con los requisitos de la multiplicación de matrices es decir que las columnas de A son diferentes a las Filas de B, nos mostrará este mensaje por pantalla

```

SELECCIONA UNA OPCION (1 - 4): 4
Filas: 3
columnas: 3

Rellenar la matriz:
Matriz[0][0]: 2
Matriz[0][1]: 3
Matriz[0][2]: 4
Matriz[1][0]: 5
Matriz[1][1]: 6
Matriz[1][2]: 7
Matriz[2][0]: 8
Matriz[2][1]: 9
Matriz[2][2]: 1

matriz original
| 2 3 4 |
| 5 6 7 |
| 8 9 1 |
matriz transpuesta
| 2 5 8 |
| 3 6 9 |
| 4 7 1 |

```

Si tecleamos un 4, tendremos que escribir las dimensiones de nuestra matriz y rellenar sus valores, al acabar de escribirlos nos mostrará por pantalla la matriz transpuesta y de nuevo el menú por si queremos realizar otra operación.

4.4 Manual de usuario Sistema de alta/baja Usuario

```

Elige una opcion
-----

Opcion 1-Anadir usuario
Opcion 2-Dar de alta un usuario
Opcion 3-Dar de baja un usuario
Opcion 4-Salir
-----

```

Tras iniciar nuestro código, aparecerá un menú que tendrá 4 opciones, añadir un usuario, dar de alta o de baja a un usuario y por último salir de nuestro programa.

```
1
Dime tu nombre
santiago
Dime tu apellido
Moreno
Nombre del usuario: santiago Apellido del usuario: Moreno
```

Cuando pulsamos 1 en nuestro menú accederemos a la función añadir usuario en este caso para la prueba hemos creado al usuario con nombre Santiago y de apellido Moreno.

```
2
Dime el numero de usuario quieres dar de alta
0.-Nombre del usuario: santiago Apellido del usuario: Moreno

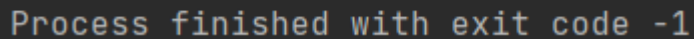
0
Usuario dado de alta con exito
```

Al pulsar 2 accederemos a la función dar de alta a un usuario donde aparecerán todos los usuarios creados hasta el momento, en este caso solo tenemos creado uno por lo que introducimos el número 0 correspondiente al usuario, dando así de alta al usuario.

```
3
Dime el identificador del usuario que quieres dar de baja
0.-Nombre del usuario: santiago Apellido del usuario: Moreno

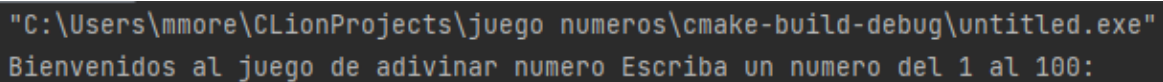
0
Usuario dado de baja con exito
```

Al pulsar la opción 3 estaremos en la opción de dar de baja a un usuario, en este caso para la prueba solo hay dado de alta un usuario, por lo que pulsamos el 0 que corresponde a ese usuario dado de alta para finalmente darlo de baja.

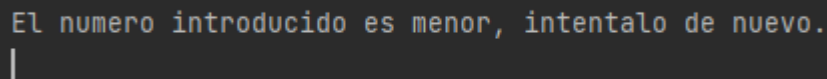


Para finalizar pulsando 4 saldremos de nuestro código.

4.5 Manual de usuario Juego adivinar número



Nada más iniciar nuestro programa nos saldrá un mensaje dándonos la bienvenida al juego, donde el usuario deberá introducir un número.



El código te va a dar indicaciones de si el número es menor o mayor al que hay que intentar adivinar, en este caso el usuario introdujo un número menor.


```
90
```

```
El numero introducido es mayor, intenalo de nuevo.
```

Si el usuario introduce un número mayor, el código le indicará que el número introducido es mayor al que buscamos

```
20
```

```
Enhorabuena has adivinado el numero en : 4 intentos.
```

En el momento que el usuario adivine el número, le mostrará un mensaje de que has conseguido adivinar el número deseado y además te dirá el número de intentos que has hecho hasta lograr adivinar el número

5. Conclusión

Hemos aprendido a aplicar todos los conocimientos adquiridos durante la unidad formativa como son almacenar la memoria en el free store el uso de excepciones, además hemos aprendido a afrontar desafíos más “complicados” en el lenguaje c++, donde hemos tenido varias dificultades y hemos sabido afrontarlas en la mayoría de lo posible y donde no, hemos aprendido a cambiar el rumbo del código y como grupo hemos mejorado las labores de comunicación y de coordinación a la hora de trabajar en grupo.