

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA.
CENTRO UNIVERSITARIO DE OCCIDENTE.
DIVISIÓN DE CIENCIAS DE LA INGENIERÍA.
INGENIERÍA EN CIENCIAS Y SISTEMAS.
LABORATORIO DE ESTRUCTURA DE DATOS.

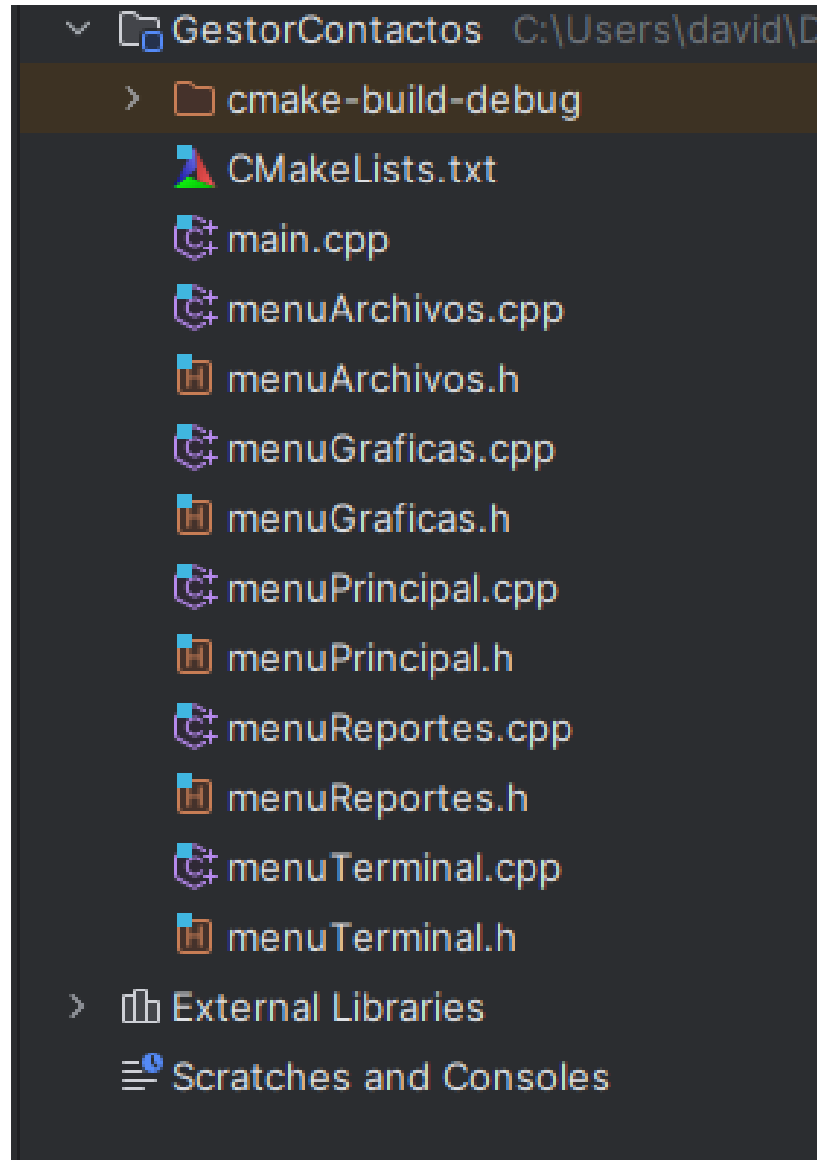


PROYECTO 02.
GESTOR DE CONTACTOS.
MANUAL TECNICO.

ESTUARDO DAVID BARRENO NIMATUJ.
CARNÉ: 201830233.

METODOS UTILIZADOS.

CREACION DE LAS CLASES.



Para la creación del proyecto se utilizaron las siguientes clases.

La clase main: es la encargada de invocar el método principal del programa.



```

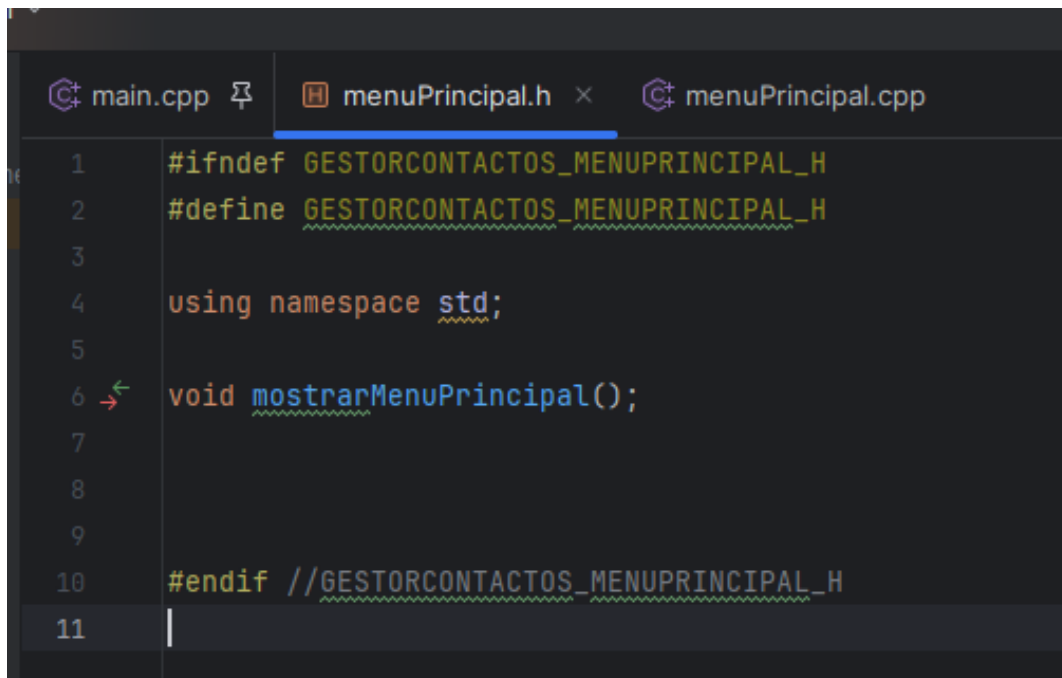
main.cpp
1  #include <iostream>
2  #include "menuPrincipal.h"
3
4  int main() {
5
6      mostrarMenuPrincipal();
7
8      return 0;
9  }

```

Cada vez que se crean más clases, se generan 2 archivos “menuPrincipal.h” y “menuPrincipal.cpp”.

En las clases.h usualmente van los atributos privados y las clases tipo públicas de los métodos de su clase cpp, que se usaran para pasar atributos o llamar funciones entre clases.

En mi caso solo esto llamando a la función “mostrarMenuPrincipal” que está ubicado en mi clase cpp.



```

main.cpp  menuPrincipal.h  menuPrincipal.cpp
1  #ifndef GESTORCONTACTOS_MENUPRINCIPAL_H
2  #define GESTORCONTACTOS_MENUPRINCIPAL_H
3
4  using namespace std;
5
6  void mostrarMenuPrincipal();
7
8
9
10 #endif //GESTORCONTACTOS_MENUPRINCIPAL_H
11

```

El método que llamo es directamente un menú, al no pasar parámetros o valores, simplemente se puede llamar a la función.

```
main.cpp  menuPrincipal.h  menuPrincipal.cpp x
1  #include "menuPrincipal.h"
2  #include "menuTerminal.h"
3  #include "menuReportes.h"
4  #include "menuGraficas.h"
5  #include "menuArchivos.h"
6  #include <iostream>
7  #include <limits> // Para limpiar el buffer de entrada
8  using namespace std;
9
10 void mostrarMenuPrincipal() {
11     int opcion;
12
13     do {
14         cout << "\n\n  ----- \n";
15         cout << "    ---  BIENBENIDO A SU    --- \n";
16         cout << "    ---  GESTOR DE CONTACTOS  --- \n";
17         cout << "    ----- \n";
18         cout << "\n\nA continuacion seleccione una opcion: \n\n";
19         cout << "1--- Acceder a la Terminal.\n";
20         cout << "2--- Menu Reportes.\n";
21         cout << "3--- Graficas de las Estructuras.\n";
22         cout << "4--- Exportacion de Contactos.\n";
23         cout << "5--- Salir del sistema.\n\n";
24         cout << "Su respuesta es: --->";
25     }
```

CREACION DE LOS MENUS.

Los menús utilizan una estructura Try - Catch, en donde se le define las opciones necesarias y el manejo de errores que se puedan presentar, cada menú se adaptó según las necesidades del programa.

```
void mostrarMenuPrincipal() {
    int opcion;

    do {
        cout << "\n\n ----- \n";
        cout << "    ---  BIENBENIDO A SU    --- \n";
        cout << "    ---  GESTOR DE CONTACTOS  --- \n";
        cout << "    ----- \n";
        cout << "\n\nA continuacion seleccione una opcion: \n\n";
        cout << "1--- Acceder a la Terminal. \n";
        cout << "2--- Menu Reportes. \n";
        cout << "3--- Graficas de las Estructuras. \n";
        cout << "4--- Exportacion de Contactos. \n";
        cout << "5--- Salir del sistema. \n\n";
        cout << "Su respuesta es: --->";

        try {
            cin >> opcion;

            if (cin.fail()) {
                throw invalid_argument("ERROR --- Ingresar solo numeros del 1 al 5.");
            }

            switch (opcion) {
                case 1:
                    mostrarMenuTerminal();
                    break;
                case 2:
                    mostrarMenuReportes();
                    break;
                case 3:
                    mostrarMenuGraficas();
                    break;
                case 4:
                    mostrarMenuArchivos();
                    break;
            }
        }
    } while (opcion < 1 || opcion > 5);
}
```

```

        case 5:
            cout << "\n\n ----- \n";
            cout << "    ---      SALIENDO DEL SISTEMA      --- \n";
            cout << "    ---   De parte de nuestros Administradores   --- \n";
            cout << "    ---      le deseamos un lindo día!!      --- \n";
            cout << "    ----- \n";
            break;
        default:
            cout << "ERROR --- Ingresa solo numeros del 1 al 5. \n";
    }
} catch (const invalid_argument& e) {
    cout << e.what() << "\n";
    cin.clear();
    cin.ignore(n: numeric_limits<streamsize>::max(), delim: '\n');
}

} while (opcion != 5);
}

```

MENU TERMINAL.

mostrarMenuTerminal();

DEFINICION EN CLASE menuTerminal.h

1. Definición de Clases y Funciones:

- Se define una clase HashTable que implementa una tabla hash.
- Se definen también las clases HashTable2 y HashTable3, que son variantes de la HashTable.
- Cada clase tiene métodos para insertar elementos, buscar elementos y mostrar la tabla.

```
main.cpp menuTerminal.cpp menuTerminal.h x
1  #ifndef GESTORCONTACTOS_MENUTERMINAL_H
2  #define GESTORCONTACTOS_MENUTERMINAL_H
3  #include <iostream>
4  #include <limits>
5  #include <vector>
6  #include <string>
7  #include <sstream>
8
9  using namespace std;
10
11  void mostrarMenuTerminal();
12
13
14  > class HashTable {...};
93
172 > class HashTable2 {...};
249 > class HashTable3 {...};
250 #endif //GESTORCONTACTOS_MENUTERMINAL_H
251 |
```

-Atributos privados en la tabla hash.

```
private:

    int size;
    int elementsCount;
    vector<pair<string, int>> table;

    int hashFunction(const string& key) {
        // Una función de hash simple para este ejemplo
        return hash<string>{}(key) % size;
    }

    int probeFunction(int index, int attempt) {
        // Función de sondeo lineal para este ejemplo
        return (index + attempt) % size;
    }

    void rehash() {
        int newSize = size * 2; // Duplicar el tamaño de la tabla
        vector<pair<string, int>> newTable(newSize);
        for (const auto& pair : table) {
            int index = hashFunction(pair.first) % newSize;
            int attempt = 0;
            while (newTable[index].first != "") {
                attempt++;
                index = probeFunction(hashFunction(pair.first), attempt) % newSize;
            }
            newTable[index] = pair;
        }
        size = newSize;
        table = std::move(newTable);
    }
}
```

2. Implementación de Tabla Hash:

- Cada clase tiene un tamaño fijo inicial de 5 para la tabla hash y un conteo de elementos.


```
using namespace std;

HashTable hashTable( tableSize: 5); // Tamaño de la tabla hash
vector<string> strings = {};
HashTable2 hashTable2( tableSize: 5);
vector<string> strings2 = {};
HashTable3 hashTable3( tableSize: 5);
vector<string> strings3 = {};
```

- La tabla hash se implementa utilizando un vector de pares (clave, valor), la clave es el parámetro que mando y el valor es la longitud de este parámetro.

```
hashTable.insert( key: nombreGrupo, value: nombreGrupo.length());
strings.push_back(nombreGrupo);
```

- Se utiliza una función de hash simple para calcular el índice de inserción de cada elemento.

```
HashTable(int tableSize) : size(tableSize), elementsCount(0) {
    table.resize(size, make_pair("", -1));
}

void insert(const string& key, int value) {
    if ((elementsCount * 100 / size) >= 60) { // Verificar el factor de carga
        rehash();
    }
    int index = hashFunction(key);
    int attempt = 0;
    while (table[index].first != "") {
        attempt++;
        index = probeFunction(hashFunction(key), attempt);
    }
    table[index] = make_pair(key, value);
    elementsCount++;
}
```

- Si se produce una colisión al insertar un elemento, se utiliza el sondeo lineal para encontrar la siguiente posición disponible en la tabla.

```

void rehash() {
    int newSize = size * 2; // Duplicar el tamaño de la tabla
    vector<pair<string, int>> newTable(newSize);
    for (const auto& pair : table) {
        int index = hashFunction(pair.first) % newSize;
        int attempt = 0;
        while (newTable[index].first != "") {
            attempt++;
            index = probeFunction(hashFunction(pair.first), attempt) % newSize;
        }
        newTable[index] = pair;
    }
    size = newSize;
    table = std::move(newTable);
}

```

- Cuando el factor de carga (número de elementos / tamaño de la tabla) supera el 60%, se duplica el tamaño de la tabla y se reorganizan los elementos.

```

void insert(const string& key, int value) {
    if ((elementsCount * 100 / size) >= 60) { // Verificar el factor de carga
        rehash();
    }
    int index = hashFunction(key);
    int attempt = 0;
    while (table[index].first != "") {
        attempt++;
        index = probeFunction(hashFunction(key), attempt);
    }
    table[index] = make_pair(key, value);
    elementsCount++;
}

```

3. Operaciones Básicas:

- Se pueden insertar elementos en la tabla proporcionando una clave y un valor.
- Se puede buscar un elemento en la tabla proporcionando la clave y se devuelve su valor asociado.
- Se puede mostrar la tabla hash con sus elementos y sus índices.

```

int search(const string& key) {
    int index = hashFunction(key);
    int attempt = 0;
    while (table[index].first != key && table[index].first != "") {
        attempt++;
        index = probeFunction(hashFunction(key), attempt);
    }
    if (table[index].first == key) {
        return table[index].second;
    } else {
        return -1; // Retorna -1 si la clave no se encuentra
    }
}

void display() {
    for (int i = 0; i < size; ++i) {
        cout << "Indice " << i << ": ";
        if (table[i].first != "") {
            cout << "( " << table[i].first << ", " << i << " ) ";

            //cout << "( " << table[i].first<< " )" ;
        }
        cout << endl;
    }
}

};

```

DEFINICION EN CLASE menuTerminal.cpp