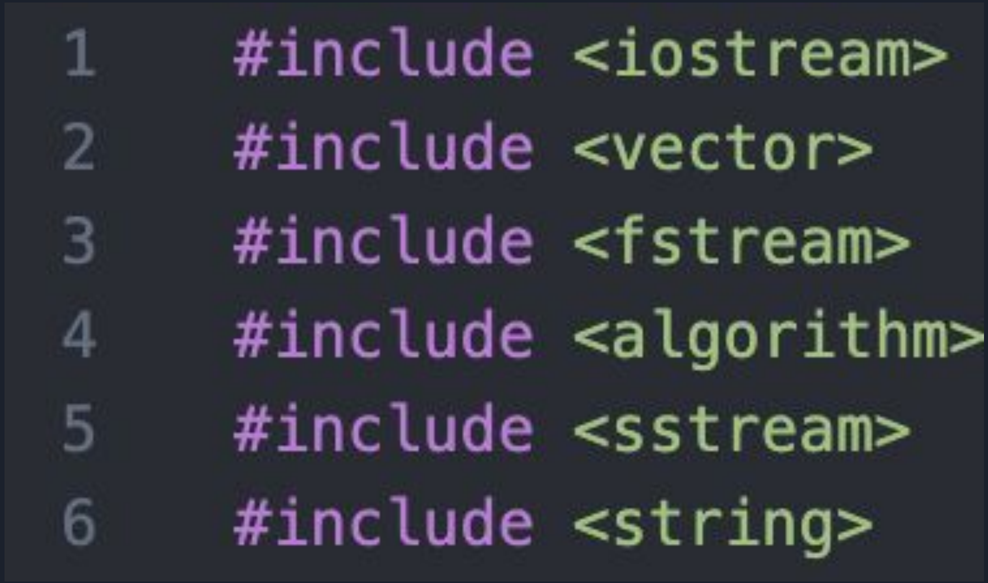
Act 2.3 - Actividad Integral estructura de datos lineales

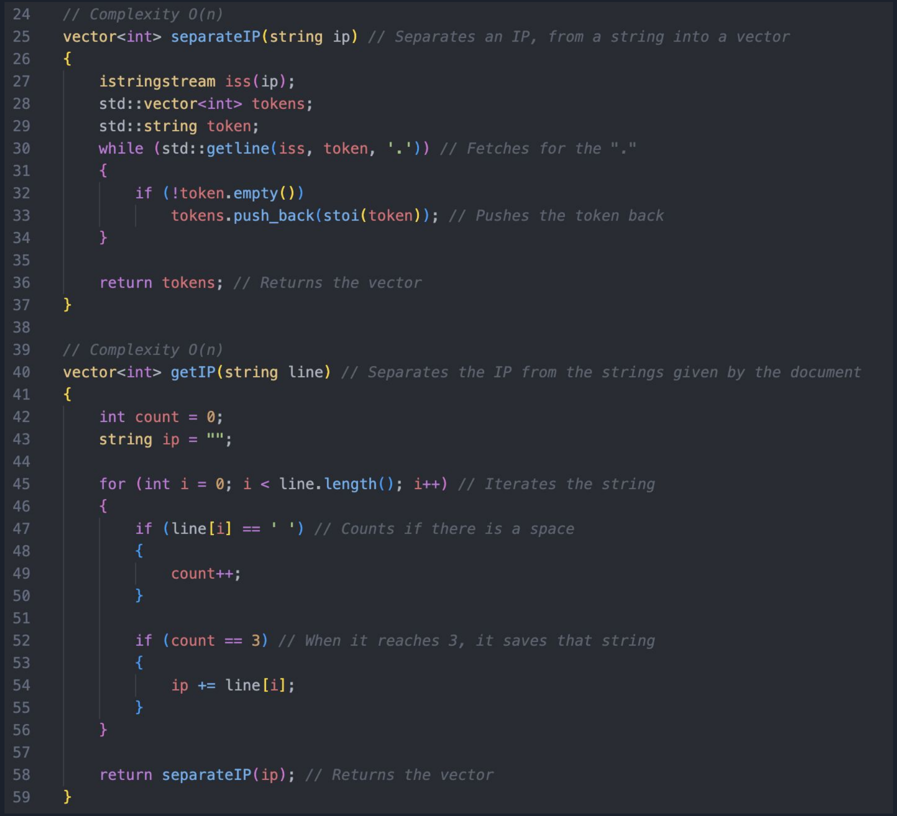
Integrantes:
Santiago Vera Espinoza - A01641585
Iker Ochoa Villaseñor - A01640984

Librerías usadas



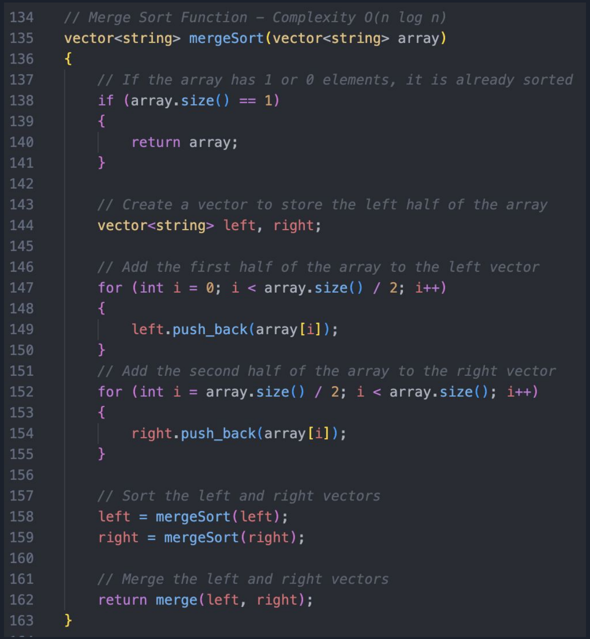
```
#include <iostream>
#include <vector>
#include <fstream>
#include <algorithm>
#include <sstream>
#include <string>
```

Funciones para la obtención de la IP

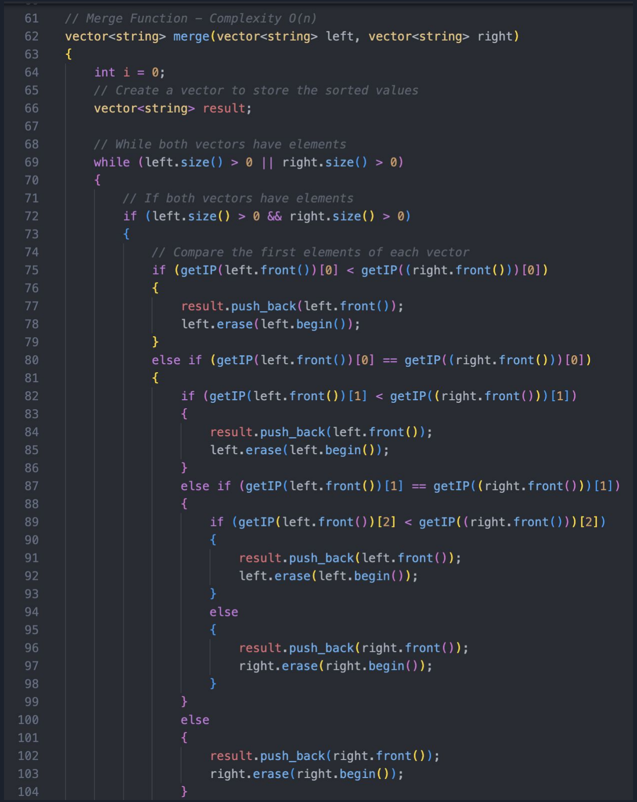


```cpp
// Complexity O(n)
vector<int> separateIP(string ip) // Separates an IP, from a string into a vector
{
    istringstream iss(ip);
    std::vector<int> tokens;
    std::string token;
    while (std::getline(iss, token, '.')) // Fetches for the "."
    {
        if (!token.empty())
            tokens.push_back(stoi(token)); // Pushes the token back
    }

    return tokens; // Returns the vector
}

// Complexity O(n)
vector<int> getIP(string line) // Separates the IP from the strings given by the document
{
    int count = 0;
    string ip = "";

    for (int i = 0; i < line.length(); i++) // Iterates the string
    {
        if (line[i] == ' ') // Counts if there is a space
        {
            count++;
        }

        if (count == 3) // When it reaches 3, it saves that string
        {
            ip += line[i];
        }
    }

    return separateIP(ip); // Returns the vector
}
```

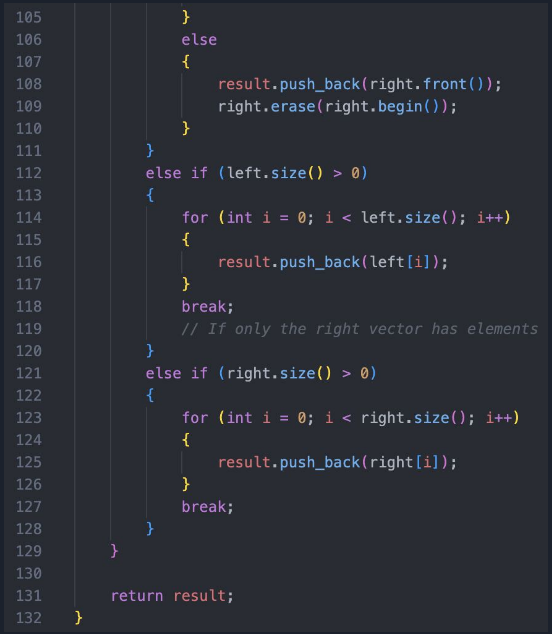
Merge Sort Function



```cpp
// Merge Sort Function - Complexity O(n log n)
vector<string> mergeSort(vector<string> array)
{
    // If the array has 1 or 0 elements, it is already sorted
    if (array.size() == 1)
    {
        return array;
    }

    // Create a vector to store the left half of the array
    vector<string> left, right;

    // Add the first half of the array to the left vector
    for (int i = 0; i < array.size() / 2; i++)
    {
        left.push_back(array[i]);
    }
    // Add the second half of the array to the right vector
    for (int i = array.size() / 2; i < array.size(); i++)
    {
        right.push_back(array[i]);
    }

    // Sort the left and right vectors
    left = mergeSort(left);
    right = mergeSort(right);

    // Merge the left and right vectors
    return merge(left, right);
}
```

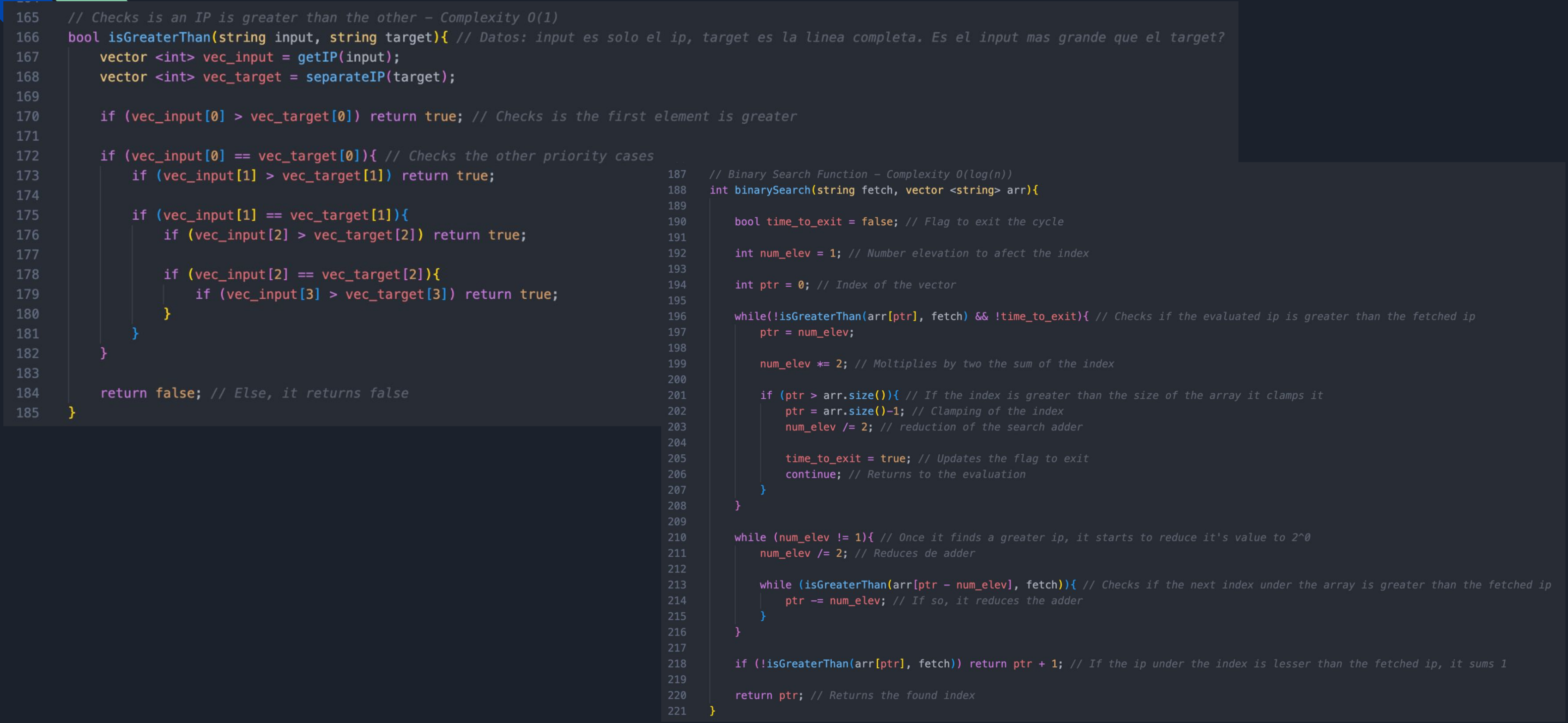


```cpp
// Merge Function - Complexity O(n)
vector<string> merge(vector<string> left, vector<string> right)
{
    int i = 0;
    // Create a vector to store the sorted values
    vector<string> result;

    // While both vectors have elements
    while (left.size() > 0 || right.size() > 0)
    {
        // If both vectors have elements
        if (left.size() > 0 && right.size() > 0)
        {
            // Compare the first elements of each vector
            if (getIP(left.front())[0] < getIP((right.front()))[0])
            {
                result.push_back(left.front());
                left.erase(left.begin());
            }
            else if (getIP(left.front())[0] == getIP((right.front()))[0])
            {
                if (getIP(left.front())[1] < getIP((right.front()))[1])
                {
                    result.push_back(left.front());
                    left.erase(left.begin());
                }
                else if (getIP(left.front())[1] == getIP((right.front()))[1])
                {
                    if (getIP(left.front())[2] < getIP((right.front()))[2])
                    {
                        result.push_back(left.front());
                        left.erase(left.begin());
                    }
                    else
                    {
                        result.push_back(right.front());
                        right.erase(right.begin());
                    }
                }
                else
                {
                    result.push_back(right.front());
                    right.erase(right.begin());
                }
```



```cpp
            }
            else
            {
                result.push_back(right.front());
                right.erase(right.begin());
            }
        }
        else if (left.size() > 0)
        {
            for (int i = 0; i < left.size(); i++)
            {
                result.push_back(left[i]);
            }
            break;
            // If only the right vector has elements
        }
        else if (right.size() > 0)
        {
            for (int i = 0; i < right.size(); i++)
            {
                result.push_back(right[i]);
            }
            break;
        }
    }

    return result;
}
```

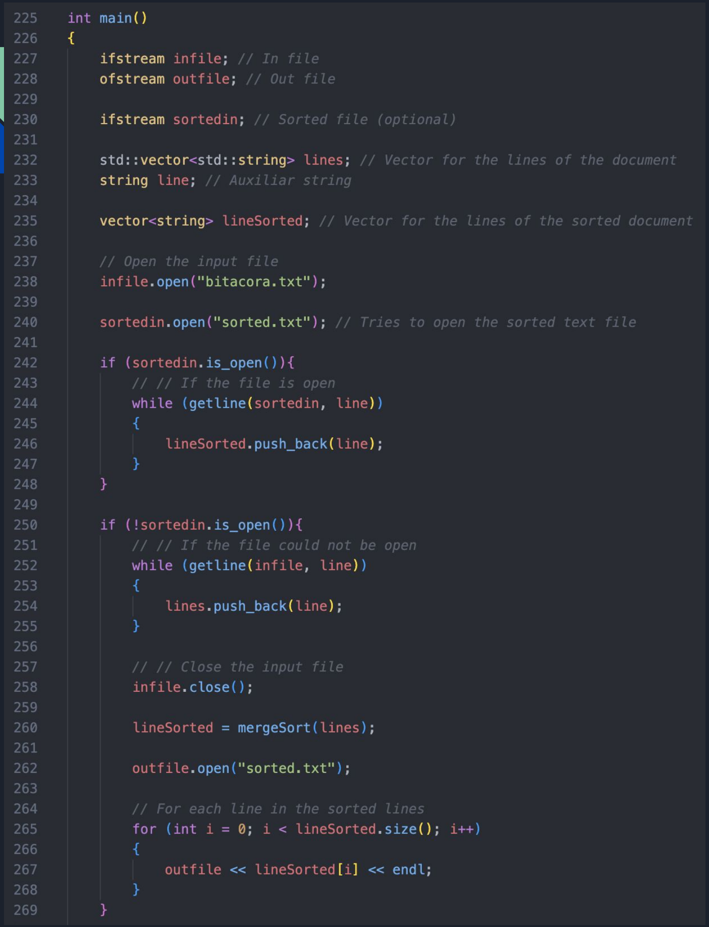
Binary Search Function



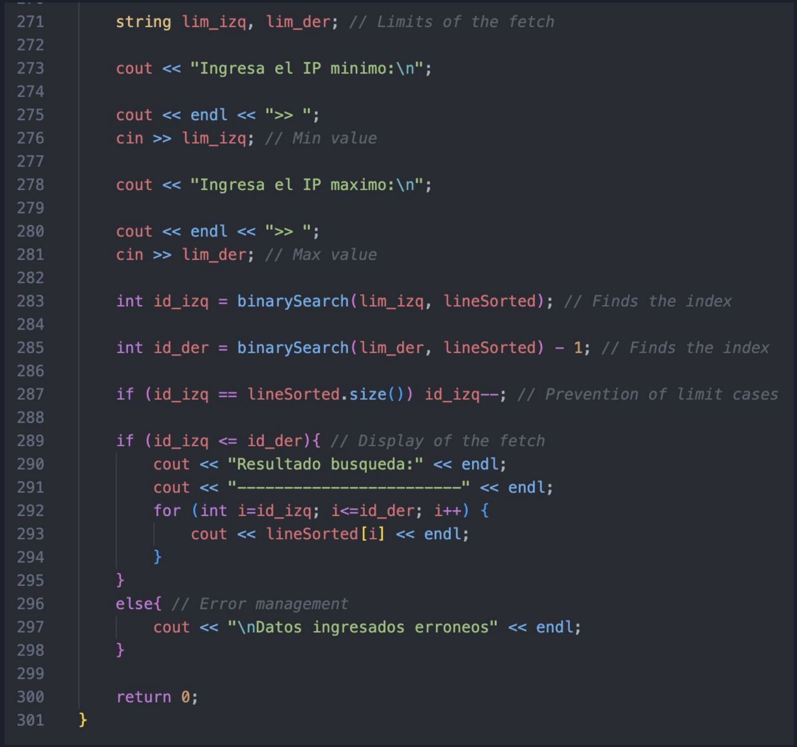
```cpp
// Checks is an IP is greater than the other - Complexity O(1)
bool isGreaterThan(string input, string target){ // Datos: input es solo el ip, target es la linea completa. Es el input mas grande que el target?
    vector <int> vec_input = getIP(input);
    vector <int> vec_target = separateIP(target);

    if (vec_input[0] > vec_target[0]) return true; // Checks is the first element is greater

    if (vec_input[0] == vec_target[0]){ // Checks the other priority cases
        if (vec_input[1] > vec_target[1]) return true;

        if (vec_input[1] == vec_target[1]){
            if (vec_input[2] > vec_target[2]) return true;

            if (vec_input[2] == vec_target[2]){
                if (vec_input[3] > vec_target[3]) return true;
            }
        }
    }

    return false; // Else, it returns false
}
```

```cpp
// Binary Search Function - Complexity O(log(n))
int binarySearch(string fetch, vector <string> arr){

    bool time_to_exit = false; // Flag to exit the cycle

    int num_elev = 1; // Number elevation to afect the index

    int ptr = 0; // Index of the vector

    while(!isGreaterThan(arr[ptr], fetch) && !time_to_exit){ // Checks if the evaluated ip is greater than the fetched ip
        ptr = num_elev;

        num_elev *= 2; // Moltiplies by two the sum of the index

        if (ptr > arr.size()){ // If the index is greater than the size of the array it clamps it
            ptr = arr.size()-1; // Clamping of the index
            num_elev /= 2; // reduction of the search adder

            time_to_exit = true; // Updates the flag to exit
            continue; // Returns to the evaluation
        }
    }

    while (num_elev != 1){ // Once it finds a greater ip, it starts to reduce it's value to 2^0
        num_elev /= 2; // Reduces de adder

        while (isGreaterThan(arr[ptr - num_elev], fetch)){ // Checks if the next index under the array is greater than the fetched ip
            ptr -= num_elev; // If so, it reduces the adder
        }
    }

    if (!isGreaterThan(arr[ptr], fetch)) return ptr + 1; // If the ip under the index is lesser than the fetched ip, it sums 1

    return ptr; // Returns the found index
}
```

Main Function



```cpp
int main()
{
    ifstream infile; // In file
    ofstream outfile; // Out file

    ifstream sortedin; // Sorted file (optional)

    std::vector<std::string> lines; // Vector for the lines of the document
    string line; // Auxiliar string

    vector<string> lineSorted; // Vector for the lines of the sorted document

    // Open the input file
    infile.open("bitacora.txt");

    sortedin.open("sorted.txt"); // Tries to open the sorted text file

    if (sortedin.is_open()){
        // // If the file is open
        while (getline(sortedin, line))
        {
            lineSorted.push_back(line);
        }
    }

    if (!sortedin.is_open()){
        // // If the file could not be open
        while (getline(infile, line))
        {
            lines.push_back(line);
        }

        // // Close the input file
        infile.close();

        lineSorted = mergeSort(lines);

        outfile.open("sorted.txt");

        // For each line in the sorted lines
        for (int i = 0; i < lineSorted.size(); i++)
        {
            outfile << lineSorted[i] << endl;
        }
    }
```



```cpp
    string lim_izq, lim_der; // Limits of the fetch

    cout << "Ingresa el IP minimo:\n";

    cout << endl << ">> ";
    cin >> lim_izq; // Min value

    cout << "Ingresa el IP maximo:\n";

    cout << endl << ">> ";
    cin >> lim_der; // Max value

    int id_izq = binarySearch(lim_izq, lineSorted); // Finds the index

    int id_der = binarySearch(lim_der, lineSorted) - 1; // Finds the index

    if (id_izq == lineSorted.size()) id_izq--; // Prevention of limit cases

    if (id_izq <= id_der){ // Display of the fetch
        cout << "Resultado busqueda:" << endl;
        cout << "------------------------" << endl;
        for (int i=id_izq; i<=id_der; i++) {
            cout << lineSorted[i] << endl;
        }
    }
    else{ // Error management
        cout << "\nDatos ingresados erroneos" << endl;
    }

    return 0;
}
```