EVIDENCIA

Actividad 2.3

Integrantes:

Andrés Emiliano de la Garza Rosales A01384096

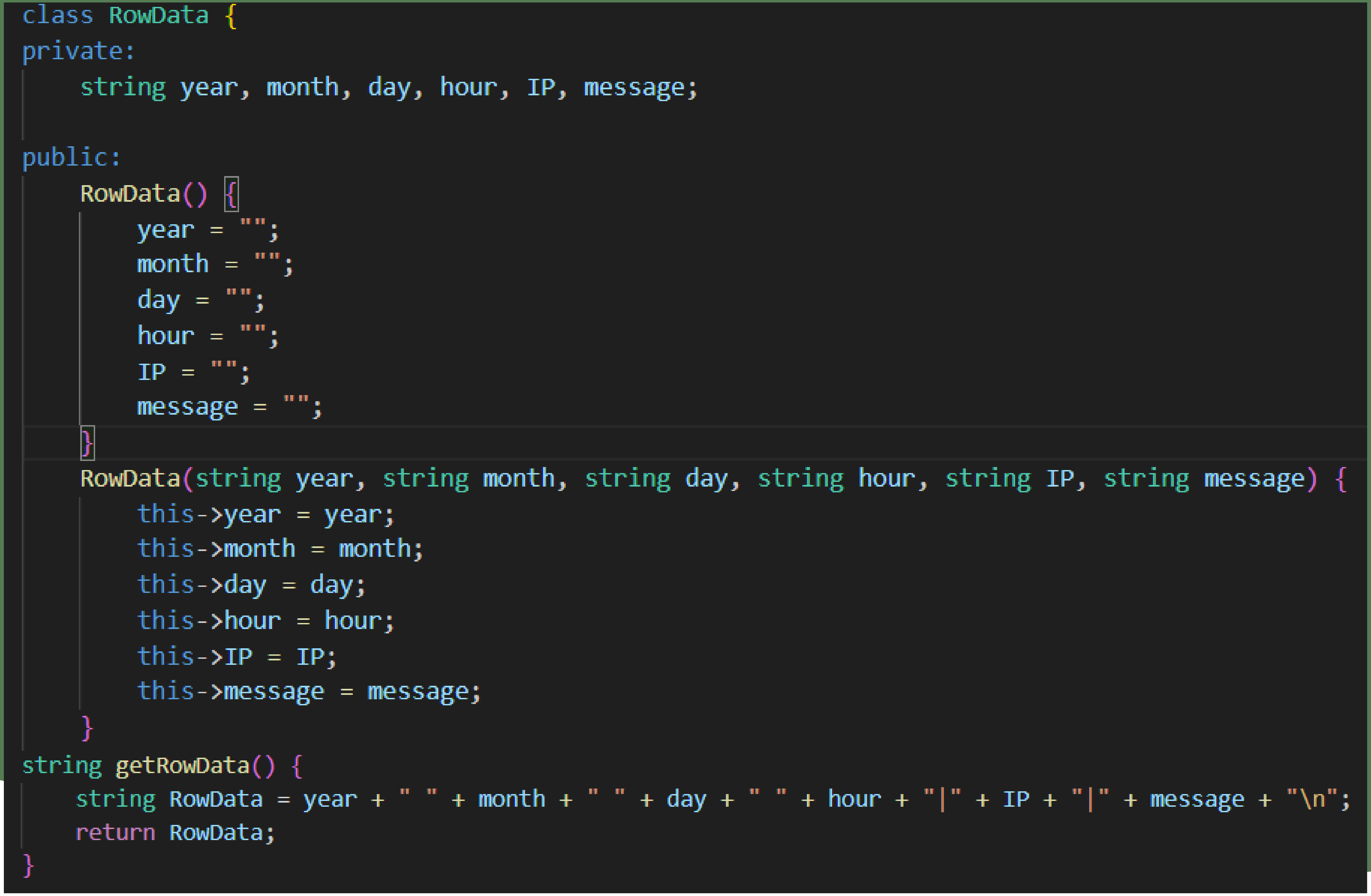
Rodrigo de Jesús Meléndez Molina A00831646

Fidel Morales briones A01198630

OBJETIVO

- Cargar el archivo bitácora en una lista ligada
- Ordenar información por fecha para realizar búsquedas y almacenar el resultado
- Mostrar un rango dependiendo de las fechas de inicio a fin
- Desplegar los registros correspondientes dentro del rango

CLASE ROW DATA

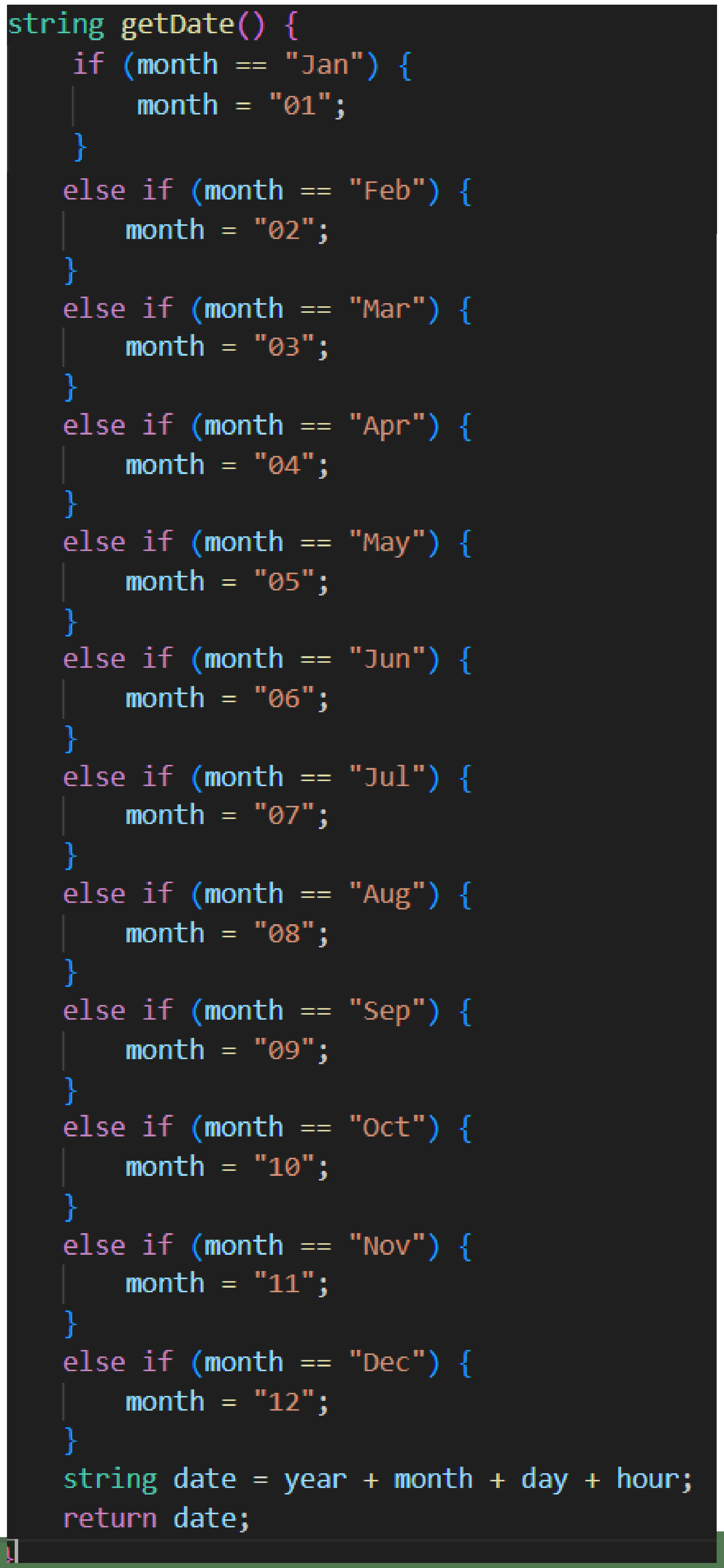


```cpp
class RowData {
private:
    string year, month, day, hour, IP, message;

public:
    RowData() {
        year = "";
        month = "";
        day = "";
        hour = "";
        IP = "";
        message = "";
    }
    RowData(string year, string month, string day, string hour, string IP, string message) {
        this->year = year;
        this->month = month;
        this->day = day;
        this->hour = hour;
        this->IP = IP;
        this->message = message;
    }
string getRowData() {
    string RowData = year + " " + month + " " + day + " " + hour + "|" + IP + "|" + message + "\n";
    return RowData;
}
```

- Clase que representa los datos de una línea de la bitácora, cuenta con un constructor sin parámetros y uno con parámetros para mayor flexibilidad al introducir los datos.
- getRowData() para formatear los datos de una línea.
- Complejidad en tiempo y espacio O(1)

FORMATO FECHA

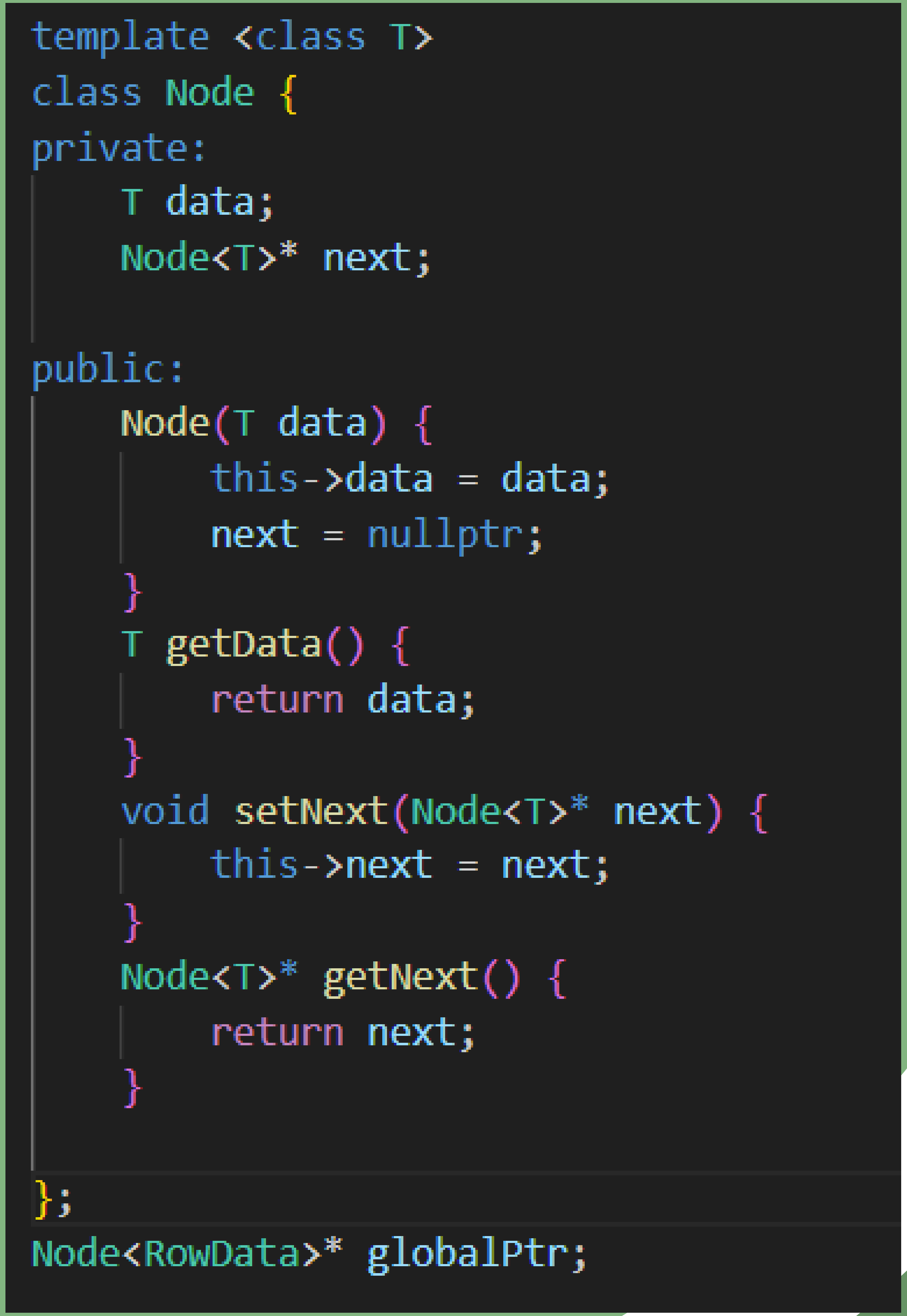
- Método tipo string getDate() para darle formato al mes y lo combina con las otras partes que conforman la fecha.
- Complejidad en tiempo y espacio O(1).



```
string getDate() {
     if (month == "Jan") {
          month = "01";
     }
    else if (month == "Feb") {
         month = "02";
    }
    else if (month == "Mar") {
         month = "03";
    }
    else if (month == "Apr") {
         month = "04";
    }
    else if (month == "May") {
         month = "05";
    }
    else if (month == "Jun") {
         month = "06";
    }
    else if (month == "Jul") {
         month = "07";
    }
    else if (month == "Aug") {
         month = "08";
    }
    else if (month == "Sep") {
         month = "09";
    }
    else if (month == "Oct") {
         month = "10";
    }
    else if (month == "Nov") {
         month = "11";
    }
    else if (month == "Dec") {
         month = "12";
    }
    string date = year + month + day + hour;
    return date;
}
```

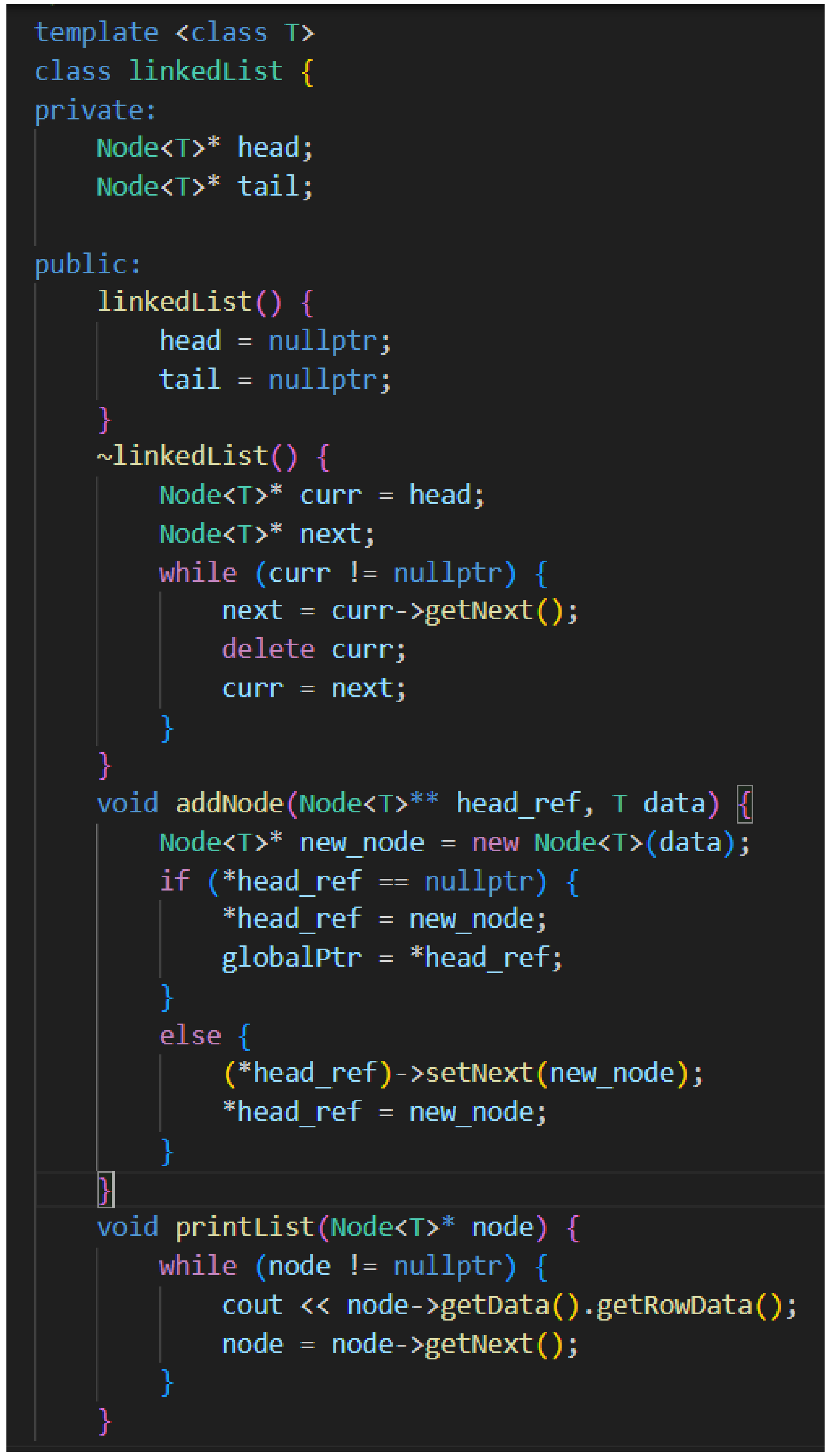
CLASE NODE

- Clase que representa un nodo de la lista ligada, se utiliza para almacenar datos de la clase RowData.
- Complejidad en espacio y tiempo O(1)
- Fuera de la función también se cuenta con un puntero global que apunta a la cabeza de la lista ligada.



```cpp
template <class T>
class Node {
private:
    T data;
    Node<T>* next;

public:
    Node(T data) {
        this->data = data;
        next = nullptr;
    }
    T getData() {
        return data;
    }
    void setNext(Node<T>* next) {
        this->next = next;
    }
    Node<T>* getNext() {
        return next;
    }

};
Node<RowData>* globalPtr;
```

CLASE LINKED LIST



```
template <class T>
class linkedList {
private:
    Node<T>* head;
    Node<T>* tail;

public:
    linkedList() {
        head = nullptr;
        tail = nullptr;
    }
    ~linkedList() {
        Node<T>* curr = head;
        Node<T>* next;
        while (curr != nullptr) {
            next = curr->getNext();
            delete curr;
            curr = next;
        }
    }
    void addNode(Node<T>** head_ref, T data) {
        Node<T>* new_node = new Node<T>(data);
        if (*head_ref == nullptr) {
            *head_ref = new_node;
            globalPtr = *head_ref;
        }
        else {
            (*head_ref)->setNext(new_node);
            *head_ref = new_node;
        }
    }
    void printList(Node<T>* node) {
        while (node != nullptr) {
            cout << node->getData().getRowData();
            node = node->getNext();
        }
    }
```

- Clase que representa la lista ligada con un constructor y destructor para liberar los espacios de memoria
  Complejidad constructor tiempo y espacio O(1)
  Complejidad destructor tiempo O(n), espacio O(1)
- Cuenta con un método para crear nodos, si la lista esta vacía entonces head es el nuevo nodo, si no establece el nodo actual como el siguiente y el nuevo nodo es la cabeza.
  Complejidad tiempo y espacio O(1)
- Método printList() que permite imprimir todos los datos de los nodos en la lista ligada
  Complejidad tiempo O(n), espacio O(1)

LENGTH

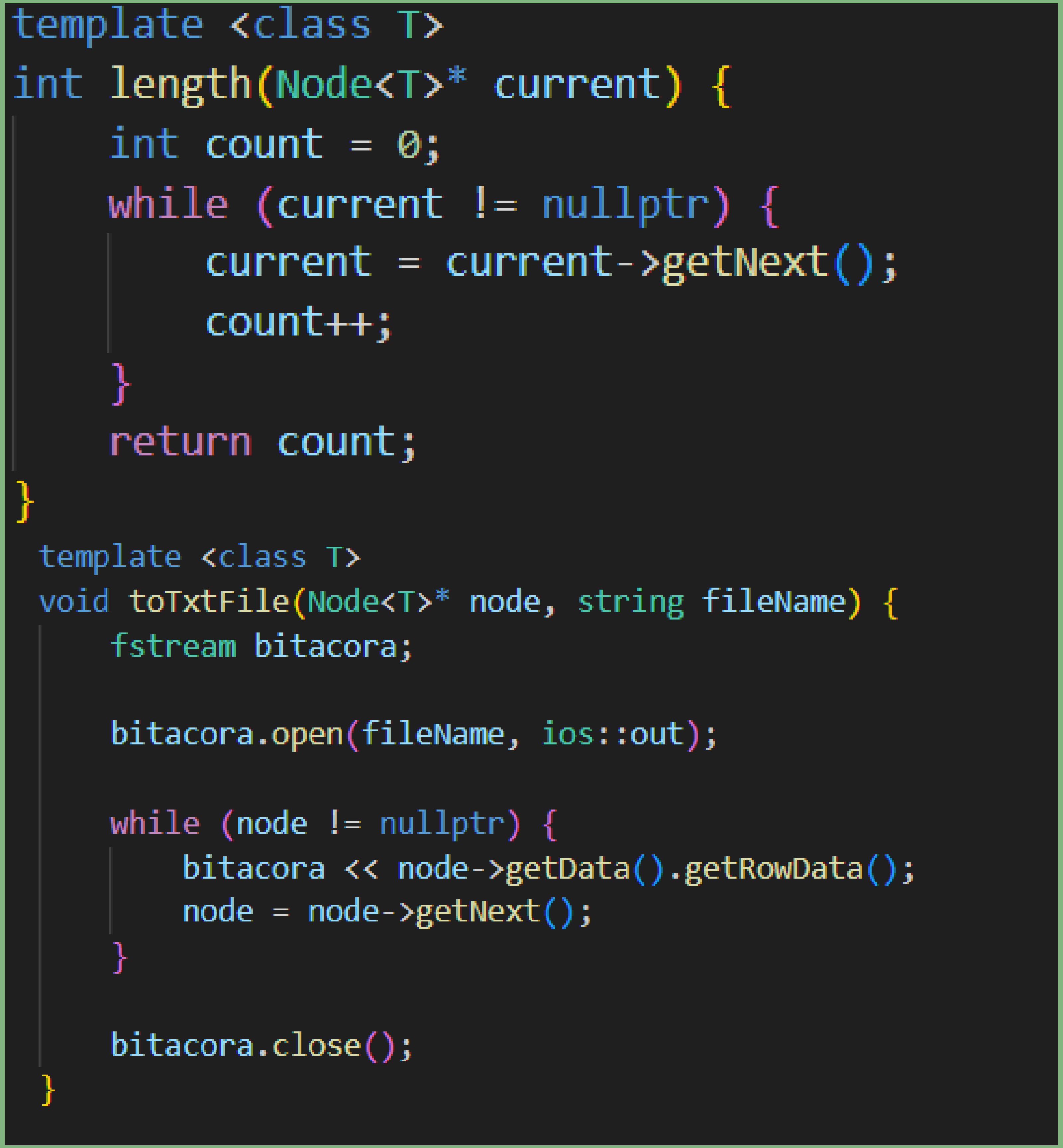
Función que recibe un puntero al nodo de inicio de una lista ligada y devuelve la longitud de la lista.

Complejidad tiempo: O(n)
Complejidad espacio: O(1)

TO TEXT FILE

Función que toma una lista ligada y un archivo .txt para escribir todos los datos de la lista, cada nodo se vuelve una línea en el archivo .txt.

Complejidad tiempo: O(n)
Complejidad espacio: O(1)



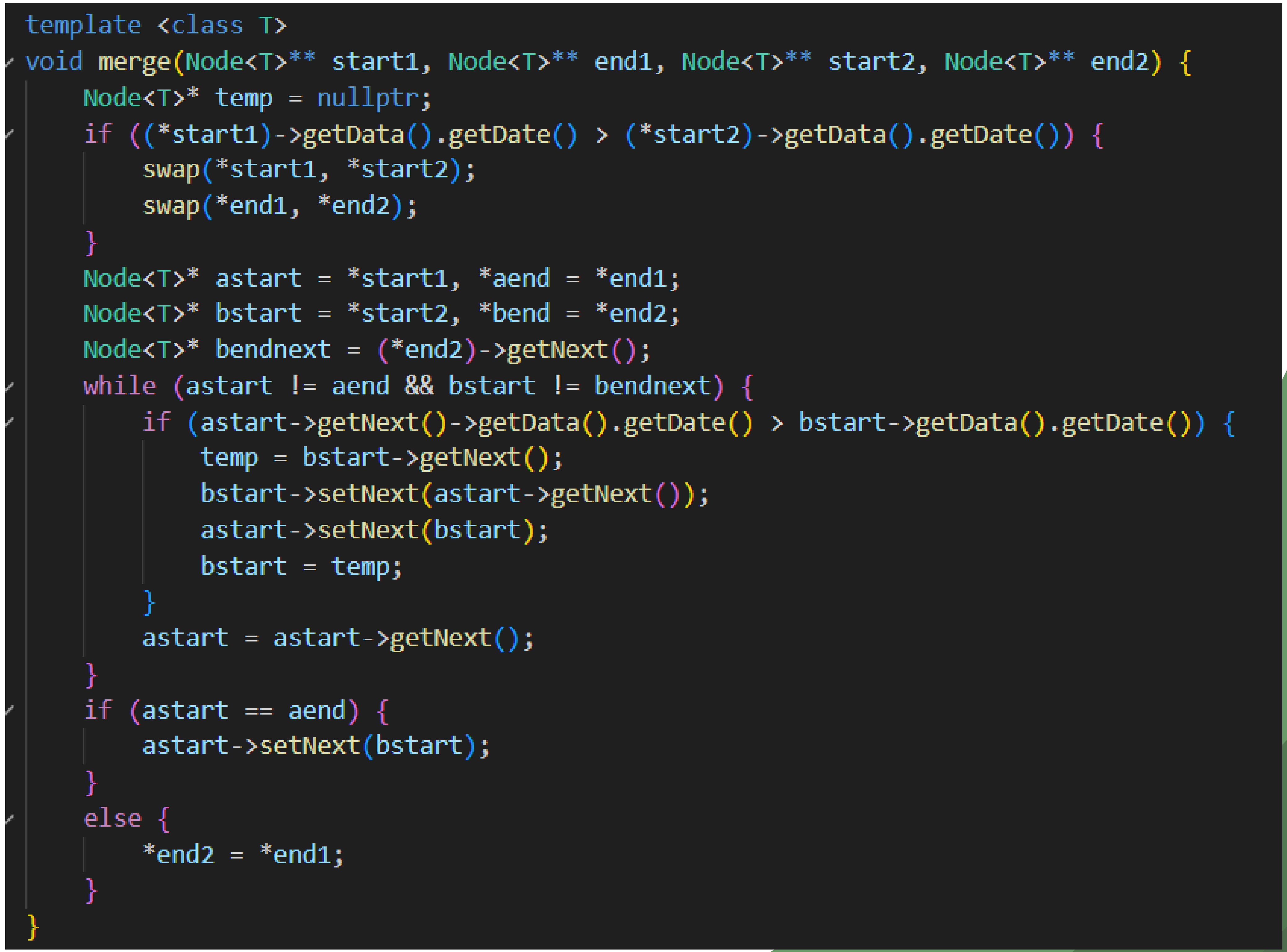
```
template <class T>
int length(Node<T>* current) {
    int count = 0;
    while (current != nullptr) {
        current = current->getNext();
        count++;
    }
    return count;
}
```

```
template <class T>
void toTxtFile(Node<T>* node, string fileName) {
    fstream bitacora;

    bitacora.open(fileName, ios::out);

    while (node != nullptr) {
        bitacora << node->getData().getRowData();
        node = node->getNext();
    }

    bitacora.close();
}
```

MERGE

Se obtienen dos sublistas, se ordenan, y se unen al final. No se crea una nueva lista ligada, se modifican los punteros de los nodos de las listas ligadas que se están uniendo por lo que no se utiliza memoria extra.

Complejidad tiempo: O(n)
Complejidad espacio: O(1)

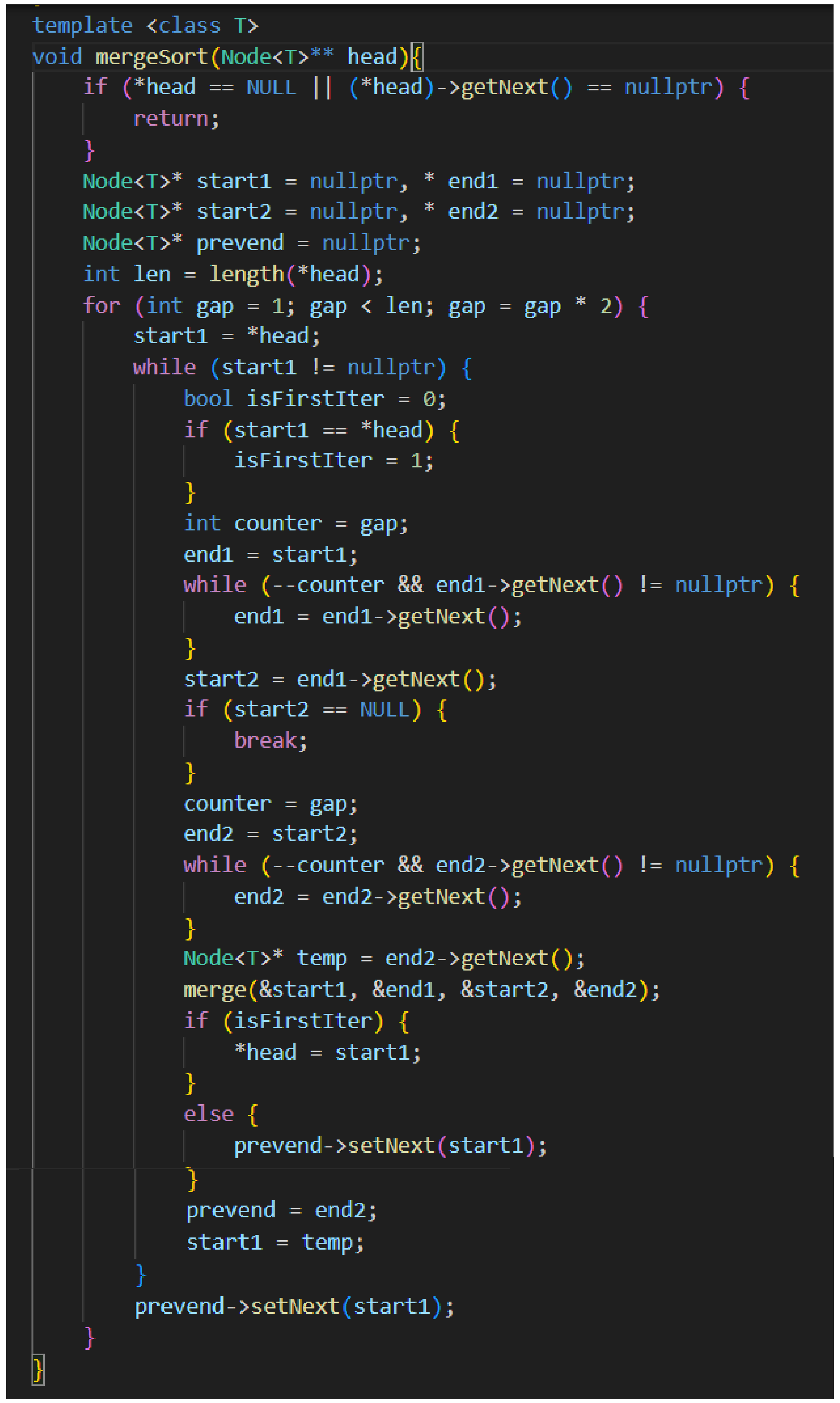


```
template <class T>
void merge(Node<T>** start1, Node<T>** end1, Node<T>** start2, Node<T>** end2) {
    Node<T>* temp = nullptr;
    if ((*start1)->getData().getDate() > (*start2)->getData().getDate()) {
        swap(*start1, *start2);
        swap(*end1, *end2);
    }
    Node<T>* astart = *start1, *aend = *end1;
    Node<T>* bstart = *start2, *bend = *end2;
    Node<T>* bendnext = (*end2)->getNext();
    while (astart != aend && bstart != bendnext) {
        if (astart->getNext()->getData().getDate() > bstart->getData().getDate()) {
            temp = bstart->getNext();
            bstart->setNext(astart->getNext());
            astart->setNext(bstart);
            bstart = temp;
        }
        astart = astart->getNext();
    }
    if (astart == aend) {
        astart->setNext(bstart);
    }
    else {
        *end2 = *end1;
    }
}
```

MERGE SORT

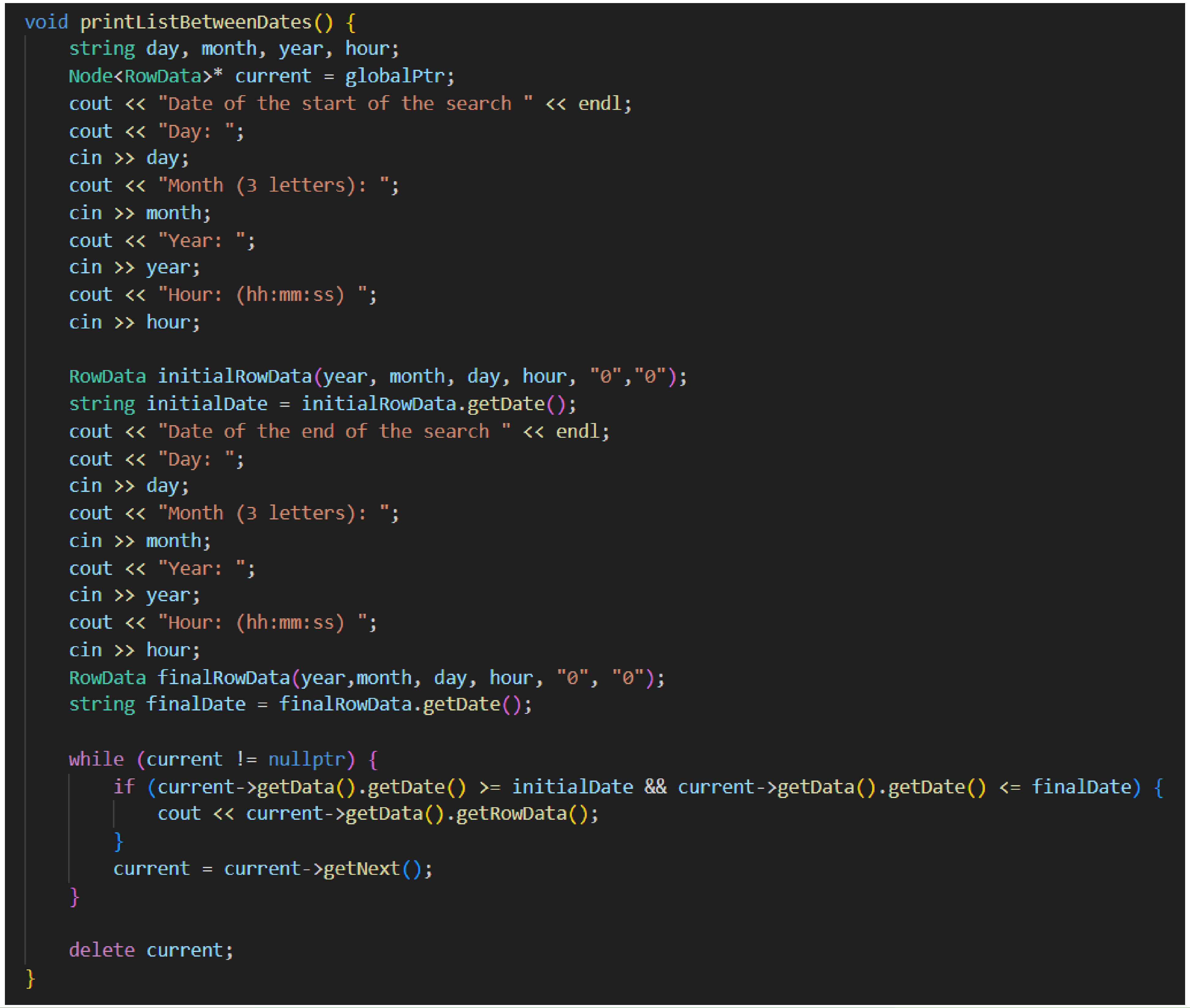
Se ingresa el puntero que apunta al puntero de la cabeza de la lista ligada, y se ordena la lista ligada con el algoritmo merge sort. En esta versión se crean sublistas y gaps, el tamaño del gap define la cantidad de nodos que va a tener cada sublista, se van a ordenar y unir las sublistas de menor a mayor con el método merge() hasta que se unan todas las sublistas y quede una sola lista ligada ordenada

Complejidad tiempo: O(n*log(n))
Complejidad espacio: O(1)



```cpp
template <class T>
void mergeSort(Node<T>** head){
    if (*head == NULL || (*head)->getNext() == nullptr) {
        return;
    }
    Node<T>* start1 = nullptr, * end1 = nullptr;
    Node<T>* start2 = nullptr, * end2 = nullptr;
    Node<T>* prevend = nullptr;
    int len = length(*head);
    for (int gap = 1; gap < len; gap = gap * 2) {
        start1 = *head;
        while (start1 != nullptr) {
            bool isFirstIter = 0;
            if (start1 == *head) {
                isFirstIter = 1;
            }
            int counter = gap;
            end1 = start1;
            while (--counter && end1->getNext() != nullptr) {
                end1 = end1->getNext();
            }
            start2 = end1->getNext();
            if (start2 == NULL) {
                break;
            }
            counter = gap;
            end2 = start2;
            while (--counter && end2->getNext() != nullptr) {
                end2 = end2->getNext();
            }
            Node<T>* temp = end2->getNext();
            merge(&start1, &end1, &start2, &end2);
            if (isFirstIter) {
                *head = start1;
            }
            else {
                prevend->setNext(start1);
            }
            prevend = end2;
            start1 = temp;
        }
        prevend->setNext(start1);
    }
}
```

PRINT LIST BETWEEN DATES



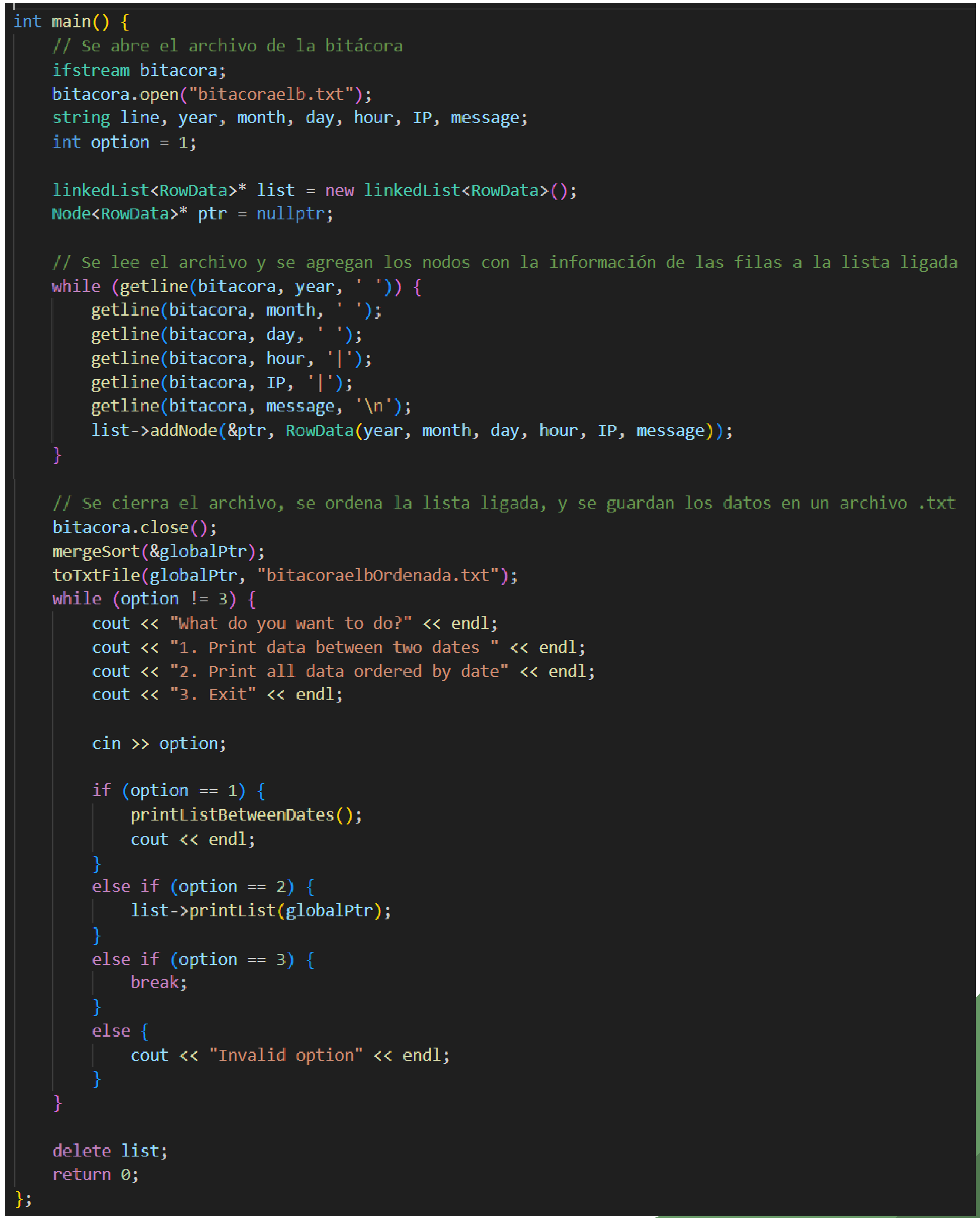
```cpp
void printListBetweenDates() {
    string day, month, year, hour;
    Node<RowData>* current = globalPtr;
    cout << "Date of the start of the search " << endl;
    cout << "Day: ";
    cin >> day;
    cout << "Month (3 letters): ";
    cin >> month;
    cout << "Year: ";
    cin >> year;
    cout << "Hour: (hh:mm:ss) ";
    cin >> hour;

    RowData initialRowData(year, month, day, hour, "0","0");
    string initialDate = initialRowData.getDate();
    cout << "Date of the end of the search " << endl;
    cout << "Day: ";
    cin >> day;
    cout << "Month (3 letters): ";
    cin >> month;
    cout << "Year: ";
    cin >> year;
    cout << "Hour: (hh:mm:ss) ";
    cin >> hour;
    RowData finalRowData(year,month, day, hour, "0", "0");
    string finalDate = finalRowData.getDate();

    while (current != nullptr) {
        if (current->getData().getDate() >= initialDate && current->getData().getDate() <= finalDate) {
            cout << current->getData().getRowData();
        }
        current = current->getNext();
    }

    delete current;
}
```

Imprime los datos de una lista ligada entre 2 fechas dadas por el usuario

Complejidad tiempo: O(n)
Complejidad espacio: O(1)

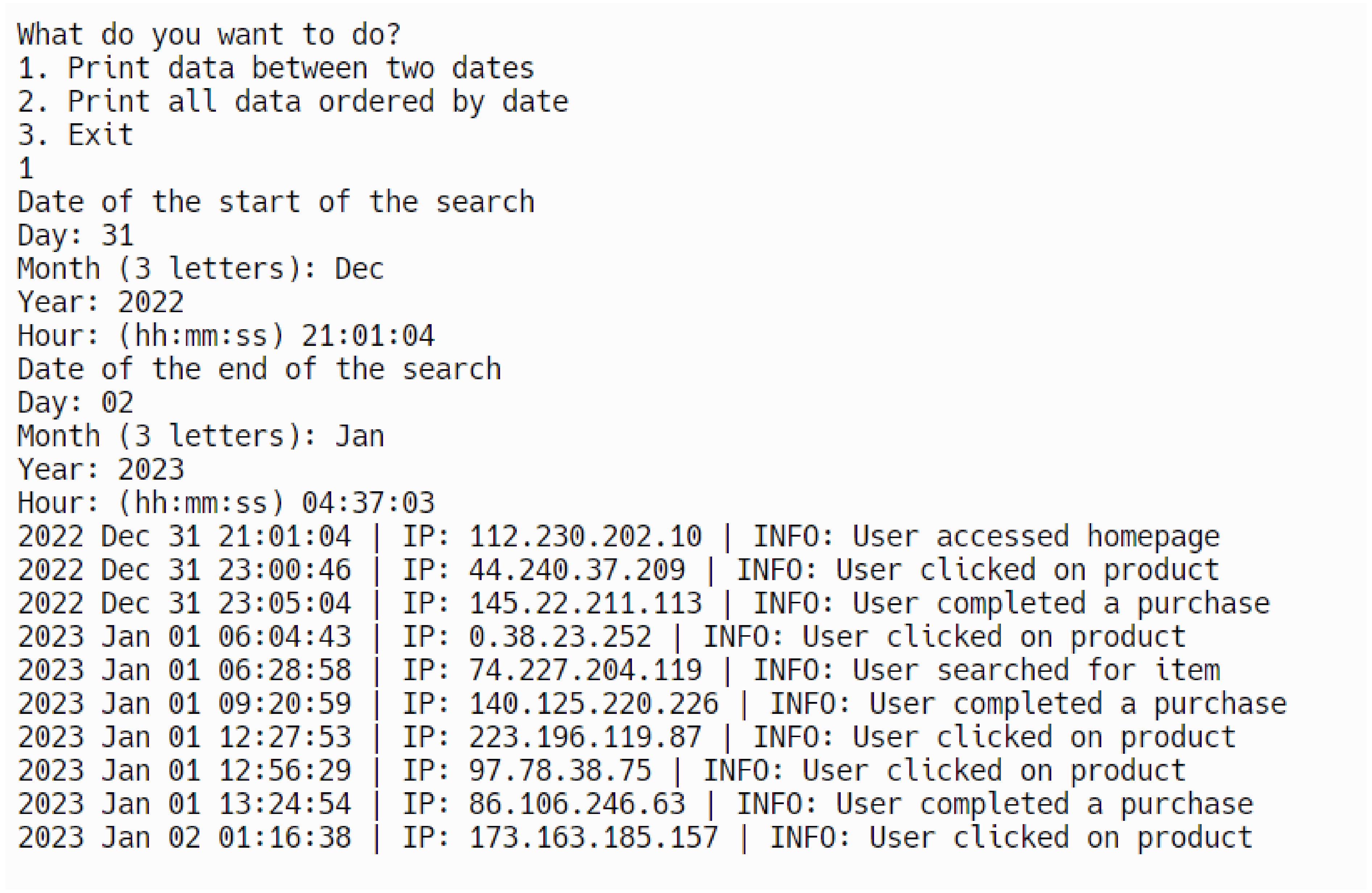
MAIN

- Abrir bitácora
- Crear una lista ligada
- Agregar datos en nodos de la lista ligada
- mergeSort()
- Importar información a un archivo .txt
- Menú de opciones
- delete list

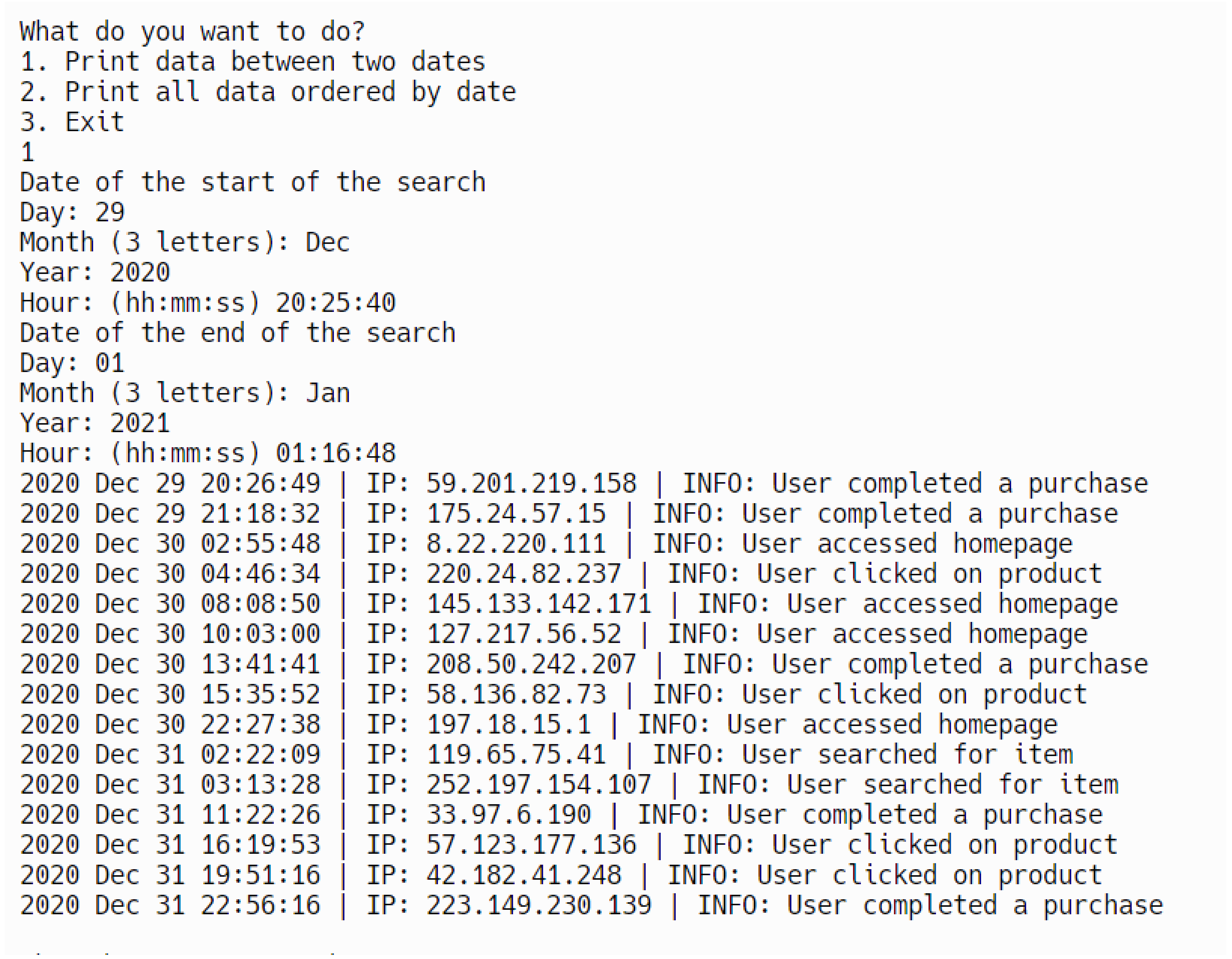


```
int main() {
    // Se abre el archivo de la bitácora
    ifstream bitacora;
    bitacora.open("bitacoraelb.txt");
    string line, year, month, day, hour, IP, message;
    int option = 1;

    linkedList<RowData>* list = new linkedList<RowData>();
    Node<RowData>* ptr = nullptr;

    // Se lee el archivo y se agregan los nodos con la información de las filas a la lista ligada
    while (getline(bitacora, year, ' ')) {
        getline(bitacora, month, ' ');
        getline(bitacora, day, ' ');
        getline(bitacora, hour, '|');
        getline(bitacora, IP, '|');
        getline(bitacora, message, '\n');
        list->addNode(&ptr, RowData(year, month, day, hour, IP, message));
    }

    // Se cierra el archivo, se ordena la lista ligada, y se guardan los datos en un archivo .txt
    bitacora.close();
    mergeSort(&globalPtr);
    toTxtFile(globalPtr, "bitacoraelbOrdenada.txt");
    while (option != 3) {
        cout << "What do you want to do?" << endl;
        cout << "1. Print data between two dates " << endl;
        cout << "2. Print all data ordered by date" << endl;
        cout << "3. Exit" << endl;

        cin >> option;

        if (option == 1) {
            printListBetweenDates();
            cout << endl;
        }
        else if (option == 2) {
            list->printList(globalPtr);
        }
        else if (option == 3) {
            break;
        }
        else {
            cout << "Invalid option" << endl;
        }
    }

    delete list;
    return 0;
};
```

CASOS DE PRUEBA

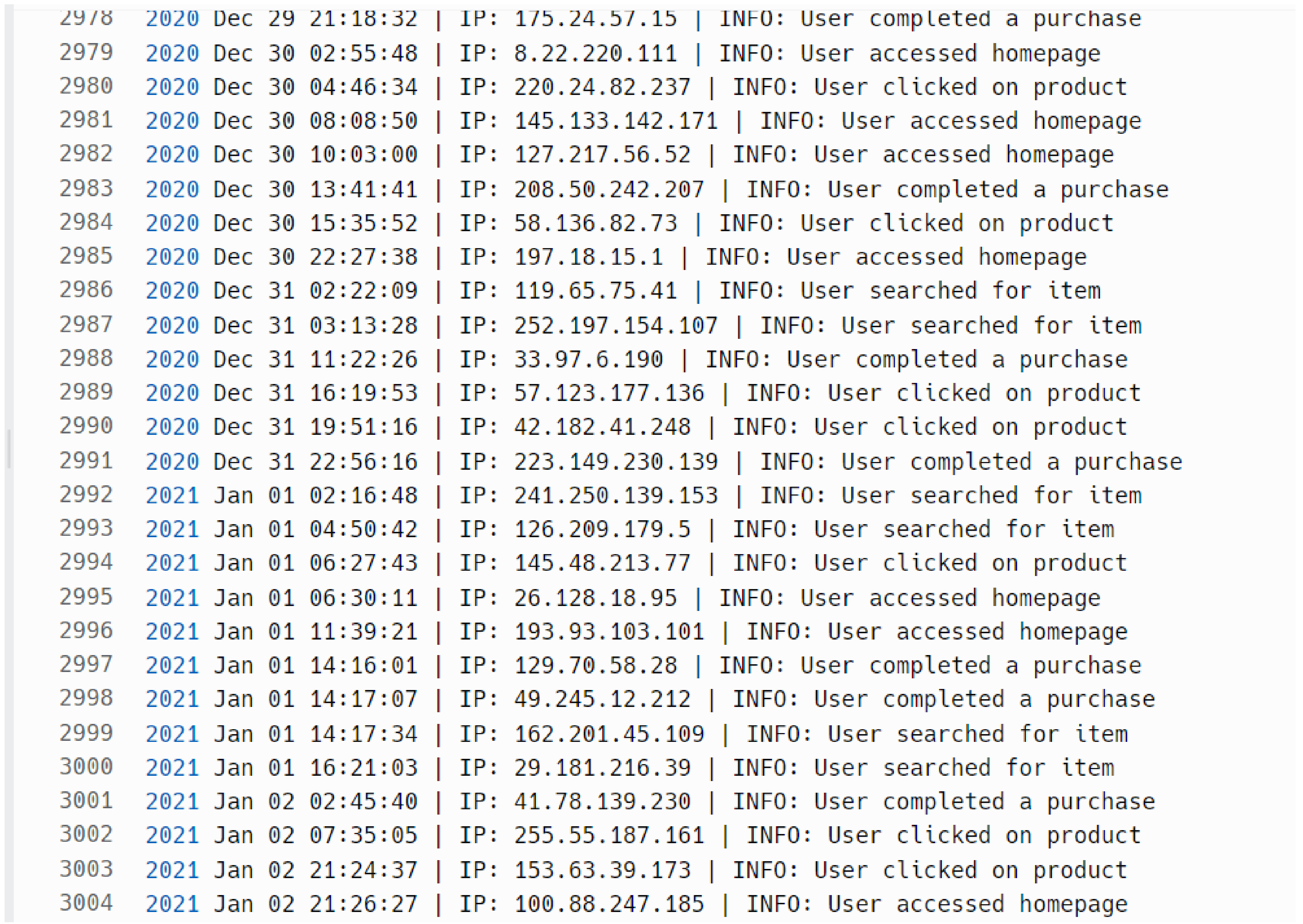


```
What do you want to do?
1. Print data between two dates
2. Print all data ordered by date
3. Exit
1
Date of the start of the search
Day: 31
Month (3 letters): Dec
Year: 2022
Hour: (hh:mm:ss) 21:01:04
Date of the end of the search
Day: 02
Month (3 letters): Jan
Year: 2023
Hour: (hh:mm:ss) 04:37:03
2022 Dec 31 21:01:04 | IP: 112.230.202.10 | INFO: User accessed homepage
2022 Dec 31 23:00:46 | IP: 44.240.37.209 | INFO: User clicked on product
2022 Dec 31 23:05:04 | IP: 145.22.211.113 | INFO: User completed a purchase
2023 Jan 01 06:04:43 | IP: 0.38.23.252 | INFO: User clicked on product
2023 Jan 01 06:28:58 | IP: 74.227.204.119 | INFO: User searched for item
2023 Jan 01 09:20:59 | IP: 140.125.220.226 | INFO: User completed a purchase
2023 Jan 01 12:27:53 | IP: 223.196.119.87 | INFO: User clicked on product
2023 Jan 01 12:56:29 | IP: 97.78.38.75 | INFO: User clicked on product
2023 Jan 01 13:24:54 | IP: 86.106.246.63 | INFO: User completed a purchase
2023 Jan 02 01:16:38 | IP: 173.163.185.157 | INFO: User clicked on product
```

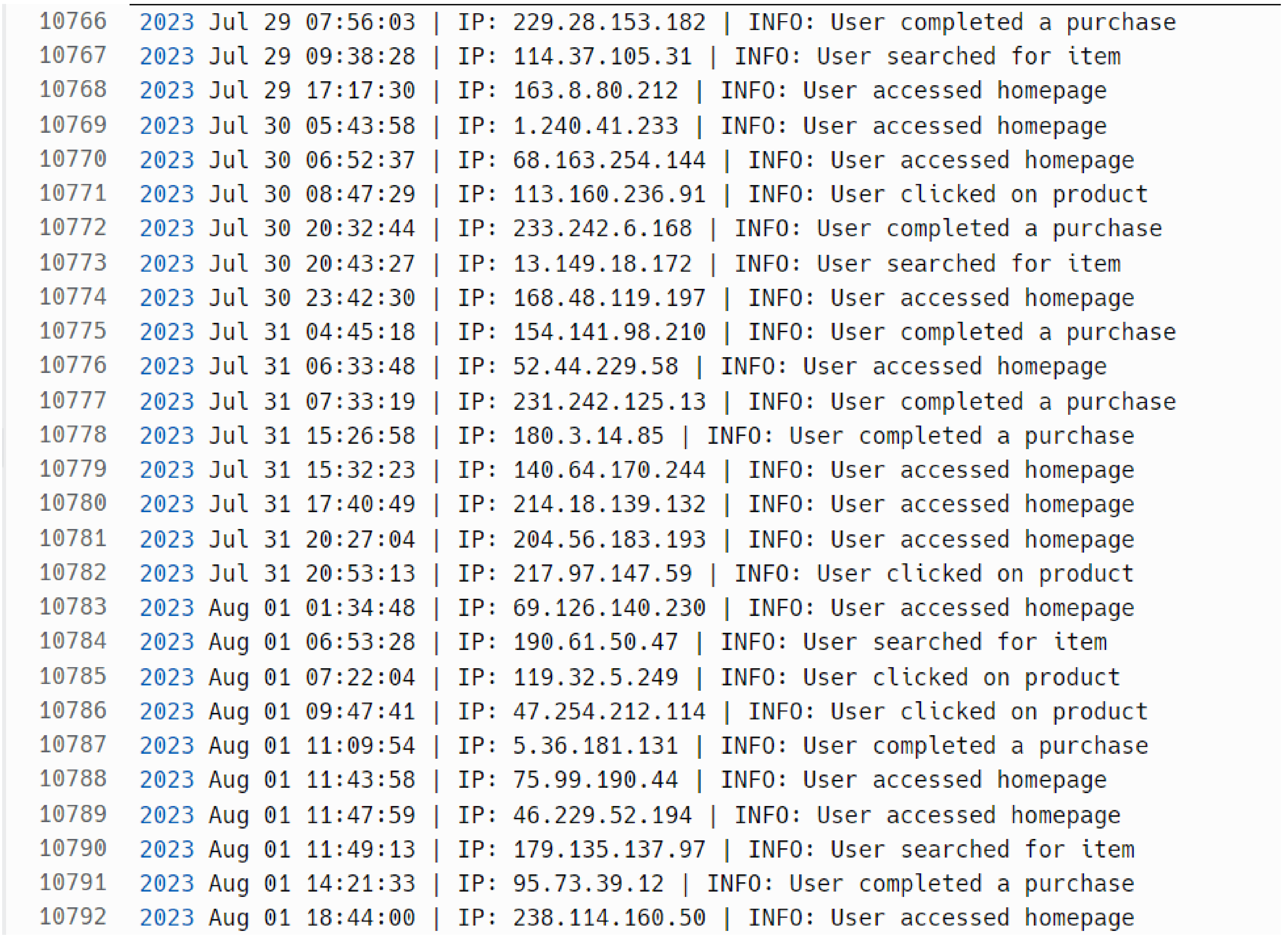


```
What do you want to do?
1. Print data between two dates
2. Print all data ordered by date
3. Exit
1
Date of the start of the search
Day: 29
Month (3 letters): Dec
Year: 2020
Hour: (hh:mm:ss) 20:25:40
Date of the end of the search
Day: 01
Month (3 letters): Jan
Year: 2021
Hour: (hh:mm:ss) 01:16:48
2020 Dec 29 20:26:49 | IP: 59.201.219.158 | INFO: User completed a purchase
2020 Dec 29 21:18:32 | IP: 175.24.57.15 | INFO: User completed a purchase
2020 Dec 30 02:55:48 | IP: 8.22.220.111 | INFO: User accessed homepage
2020 Dec 30 04:46:34 | IP: 220.24.82.237 | INFO: User clicked on product
2020 Dec 30 08:08:50 | IP: 145.133.142.171 | INFO: User accessed homepage
2020 Dec 30 10:03:00 | IP: 127.217.56.52 | INFO: User accessed homepage
2020 Dec 30 13:41:41 | IP: 208.50.242.207 | INFO: User completed a purchase
2020 Dec 30 15:35:52 | IP: 58.136.82.73 | INFO: User clicked on product
2020 Dec 30 22:27:38 | IP: 197.18.15.1 | INFO: User accessed homepage
2020 Dec 31 02:22:09 | IP: 119.65.75.41 | INFO: User searched for item
2020 Dec 31 03:13:28 | IP: 252.197.154.107 | INFO: User searched for item
2020 Dec 31 11:22:26 | IP: 33.97.6.190 | INFO: User completed a purchase
2020 Dec 31 16:19:53 | IP: 57.123.177.136 | INFO: User clicked on product
2020 Dec 31 19:51:16 | IP: 42.182.41.248 | INFO: User clicked on product
2020 Dec 31 22:56:16 | IP: 223.149.230.139 | INFO: User completed a purchase
```

BITACORA ORDENADA POR FECHA



```
2020 Dec 29 21:18:32 | IP: 175.24.57.15 | INFO: User completed a purchase
2020 Dec 30 02:55:48 | IP: 8.22.220.111 | INFO: User accessed homepage
2020 Dec 30 04:46:34 | IP: 220.24.82.237 | INFO: User clicked on product
2020 Dec 30 08:08:50 | IP: 145.133.142.171 | INFO: User accessed homepage
2020 Dec 30 10:03:00 | IP: 127.217.56.52 | INFO: User accessed homepage
2020 Dec 30 13:41:41 | IP: 208.50.242.207 | INFO: User completed a purchase
2020 Dec 30 15:35:52 | IP: 58.136.82.73 | INFO: User clicked on product
2020 Dec 30 22:27:38 | IP: 197.18.15.1 | INFO: User accessed homepage
2020 Dec 31 02:22:09 | IP: 119.65.75.41 | INFO: User searched for item
2020 Dec 31 03:13:28 | IP: 252.197.154.107 | INFO: User searched for item
2020 Dec 31 11:22:26 | IP: 33.97.6.190 | INFO: User completed a purchase
2020 Dec 31 16:19:53 | IP: 57.123.177.136 | INFO: User clicked on product
2020 Dec 31 19:51:16 | IP: 42.182.41.248 | INFO: User clicked on product
2020 Dec 31 22:56:16 | IP: 223.149.230.139 | INFO: User completed a purchase
2021 Jan 01 02:16:48 | IP: 241.250.139.153 | INFO: User searched for item
2021 Jan 01 04:50:42 | IP: 126.209.179.5 | INFO: User searched for item
2021 Jan 01 06:27:43 | IP: 145.48.213.77 | INFO: User clicked on product
2021 Jan 01 06:30:11 | IP: 26.128.18.95 | INFO: User accessed homepage
2021 Jan 01 11:39:21 | IP: 193.93.103.101 | INFO: User accessed homepage
2021 Jan 01 14:16:01 | IP: 129.70.58.28 | INFO: User completed a purchase
2021 Jan 01 14:17:07 | IP: 49.245.12.212 | INFO: User completed a purchase
2021 Jan 01 14:17:34 | IP: 162.201.45.109 | INFO: User searched for item
2021 Jan 01 16:21:03 | IP: 29.181.216.39 | INFO: User searched for item
2021 Jan 02 02:45:40 | IP: 41.78.139.230 | INFO: User completed a purchase
2021 Jan 02 07:35:05 | IP: 255.55.187.161 | INFO: User clicked on product
2021 Jan 02 21:24:37 | IP: 153.63.39.173 | INFO: User clicked on product
2021 Jan 02 21:26:27 | IP: 100.88.247.185 | INFO: User accessed homepage
```



```
2023 Jul 29 07:56:03 | IP: 229.28.153.182 | INFO: User completed a purchase
2023 Jul 29 09:38:28 | IP: 114.37.105.31 | INFO: User searched for item
2023 Jul 29 17:17:30 | IP: 163.8.80.212 | INFO: User accessed homepage
2023 Jul 30 05:43:58 | IP: 1.240.41.233 | INFO: User accessed homepage
2023 Jul 30 06:52:37 | IP: 68.163.254.144 | INFO: User accessed homepage
2023 Jul 30 08:47:29 | IP: 113.160.236.91 | INFO: User clicked on product
2023 Jul 30 20:32:44 | IP: 233.242.6.168 | INFO: User completed a purchase
2023 Jul 30 20:43:27 | IP: 13.149.18.172 | INFO: User searched for item
2023 Jul 30 23:42:30 | IP: 168.48.119.197 | INFO: User accessed homepage
2023 Jul 31 04:45:18 | IP: 154.141.98.210 | INFO: User completed a purchase
2023 Jul 31 06:33:48 | IP: 52.44.229.58 | INFO: User accessed homepage
2023 Jul 31 07:33:19 | IP: 231.242.125.13 | INFO: User completed a purchase
2023 Jul 31 15:26:58 | IP: 180.3.14.85 | INFO: User completed a purchase
2023 Jul 31 15:32:23 | IP: 140.64.170.244 | INFO: User accessed homepage
2023 Jul 31 17:40:49 | IP: 214.18.139.132 | INFO: User accessed homepage
2023 Jul 31 20:27:04 | IP: 204.56.183.193 | INFO: User accessed homepage
2023 Jul 31 20:53:13 | IP: 217.97.147.59 | INFO: User clicked on product
2023 Aug 01 01:34:48 | IP: 69.126.140.230 | INFO: User accessed homepage
2023 Aug 01 06:53:28 | IP: 190.61.50.47 | INFO: User searched for item
2023 Aug 01 07:22:04 | IP: 119.32.5.249 | INFO: User clicked on product
2023 Aug 01 09:47:41 | IP: 47.254.212.114 | INFO: User clicked on product
2023 Aug 01 11:09:54 | IP: 5.36.181.131 | INFO: User completed a purchase
2023 Aug 01 11:43:58 | IP: 75.99.190.44 | INFO: User accessed homepage
2023 Aug 01 11:47:59 | IP: 46.229.52.194 | INFO: User accessed homepage
2023 Aug 01 11:49:13 | IP: 179.135.137.97 | INFO: User searched for item
2023 Aug 01 14:21:33 | IP: 95.73.39.12 | INFO: User completed a purchase
2023 Aug 01 18:44:00 | IP: 238.114.160.50 | INFO: User accessed homepage
```

BIBLIOGRAFÍA

GeeksforGeeks. (2022). Iterative merge sort for linked list. GeeksforGeeks. https://www.geeksforgeeks.org/iterative-merge-sort-for-linked-list/

¡GRACIAS!