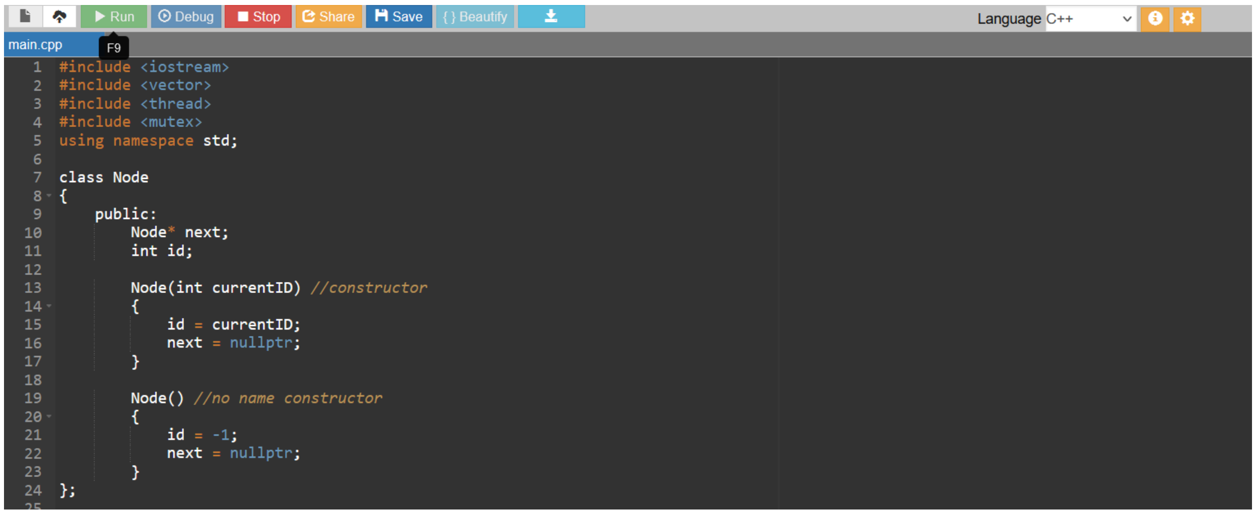
Neil Patel

Election algorithms are used to label a “leader node” among a list of nodes. This leader node is a functioning node with the highest ID value. In addition, the goal of these algorithms is to elect only one leader node based on the aforementioned criteria and to make sure the leader node is known to be the leader for all the other nodes.

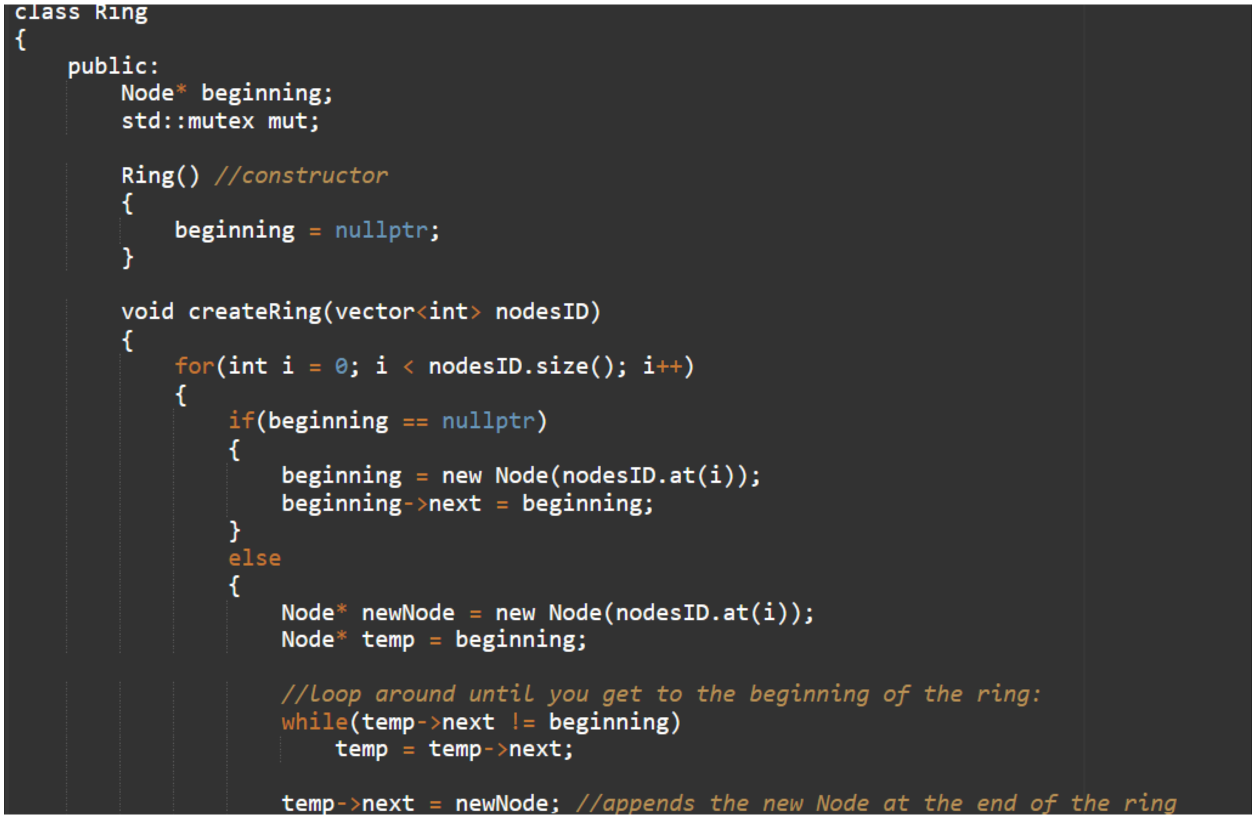
One algorithm designed to accomplish this task is the Ring Election algorithm where the election message is propagated all around a unidirectional ring of the nodes until the message reaches the node that started the election. Then, the node with the highest ID value in the list of nodes that were propagated the message successfully, becomes the new leader node.

I implemented this algorithm using C++ and multithreading. I created a class Node, consisting of the next field that points to the next Node in the ring as well as the integer ID field. A constructor was also created that creates a node with specified ID value.



```cpp
#include <iostream>
#include <vector>
#include <thread>
#include <mutex>
using namespace std;

class Node
{
    public:
        Node* next;
        int id;

        Node(int currentID) //constructor
        {
            id = currentID;
            next = nullptr;
        }

        Node() //no name constructor
        {
            id = -1;
            next = nullptr;
        }
};
```

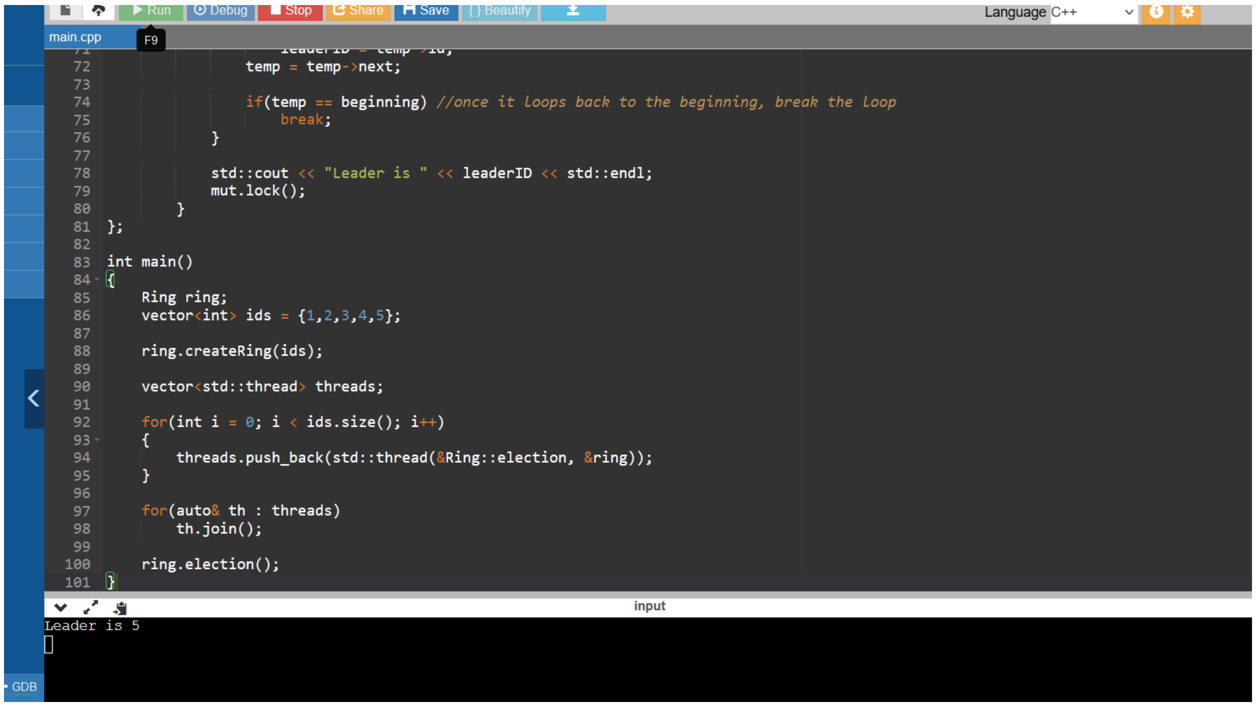
In addition to this, a Ring class was made consisting of mutex mut and Node * beginning that points to the beginning of the ring.



```
class Ring
{
    public:
        Node* beginning;
        std::mutex mut;

        Ring() //constructor
        {
            beginning = nullptr;
        }

        void createRing(vector<int> nodesID)
        {
            for(int i = 0; i < nodesID.size(); i++)
            {
                if(beginning == nullptr)
                {
                    beginning = new Node(nodesID.at(i));
                    beginning->next = beginning;
                }
                else
                {
                    Node* newNode = new Node(nodesID.at(i));
                    Node* temp = beginning;

                    //loop around until you get to the beginning of the ring:
                    while(temp->next != beginning)
                        temp = temp->next;

                    temp->next = newNode; //appends the new Node at the end of the ring
```

In this class, a createRing() function is written, where all IDs given are turned into a sequence of nodes in an unidirectional ring. An election() algorithm is also implemented where the functioning node with the largest ID value is made the leader. The threads are locked here to make sure that only one leader is selected at a time. Given the IDs: {1,2,3,4,5}, the algorithm (using threads) successfully elects the 5th node as the leader as it is the node with the largest

ID value.



```
                leaderID = temp->id;
                temp = temp->next;

                if(temp == beginning) //once it loops back to the beginning, break the loop
                    break;
            }

            std::cout << "Leader is " << leaderID << std::endl;
            mut.lock();
        }
};

int main()
{
    Ring ring;
    vector<int> ids = {1,2,3,4,5};

    ring.createRing(ids);

    vector<std::thread> threads;

    for(int i = 0; i < ids.size(); i++)
    {
        threads.push_back(std::thread(&Ring::election, &ring));
    }

    for(auto& th : threads)
        th.join();

    ring.election();
}
```

```
Leader is 5
```