Q1. Stack

#include <bits/stdc++.h>

using namespace std;

#define f 4 // number of frames

void printStack(int s[], int top)

{

    for (int i = 0; i <= top; i++)

    {

        cout << s[i] << " ";

    }

    cout << endl;

}

int main(void)

{

    int stack\_s[f];

    int top = -1;

    int cnt = 0;

    vector <int> s = {7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2};

    int n = 13;

    unordered\_map<int, int> mp;

    for (int i = 0; i < n; i++)

    {

        int c = s[i];

        //cout << c << " ";

        if (mp.find(c) == mp.end()) // nhi mila

        {

            if (top < f - 1)

            {

                top++;

                stack\_s[top] = c;

            }

            else

            {

                // remove last element

                for (int i = 0; i < top; i++)

                {

                    stack\_s[i] = stack\_s[i + 1];

                }

                stack\_s[top] = c;

                mp.erase(stack\_s[top]);

            }

            cnt++;

            mp[c]=1;

        }

        else

        { // bring it to the top

            //find

            int i = top;

            while (stack\_s[i] != c){

                i--;

            }

            // now swap locations

            for (int j=i; j<top; j++){

                stack\_s[j]=stack\_s[j+1];

            }

            stack\_s[top]=c;

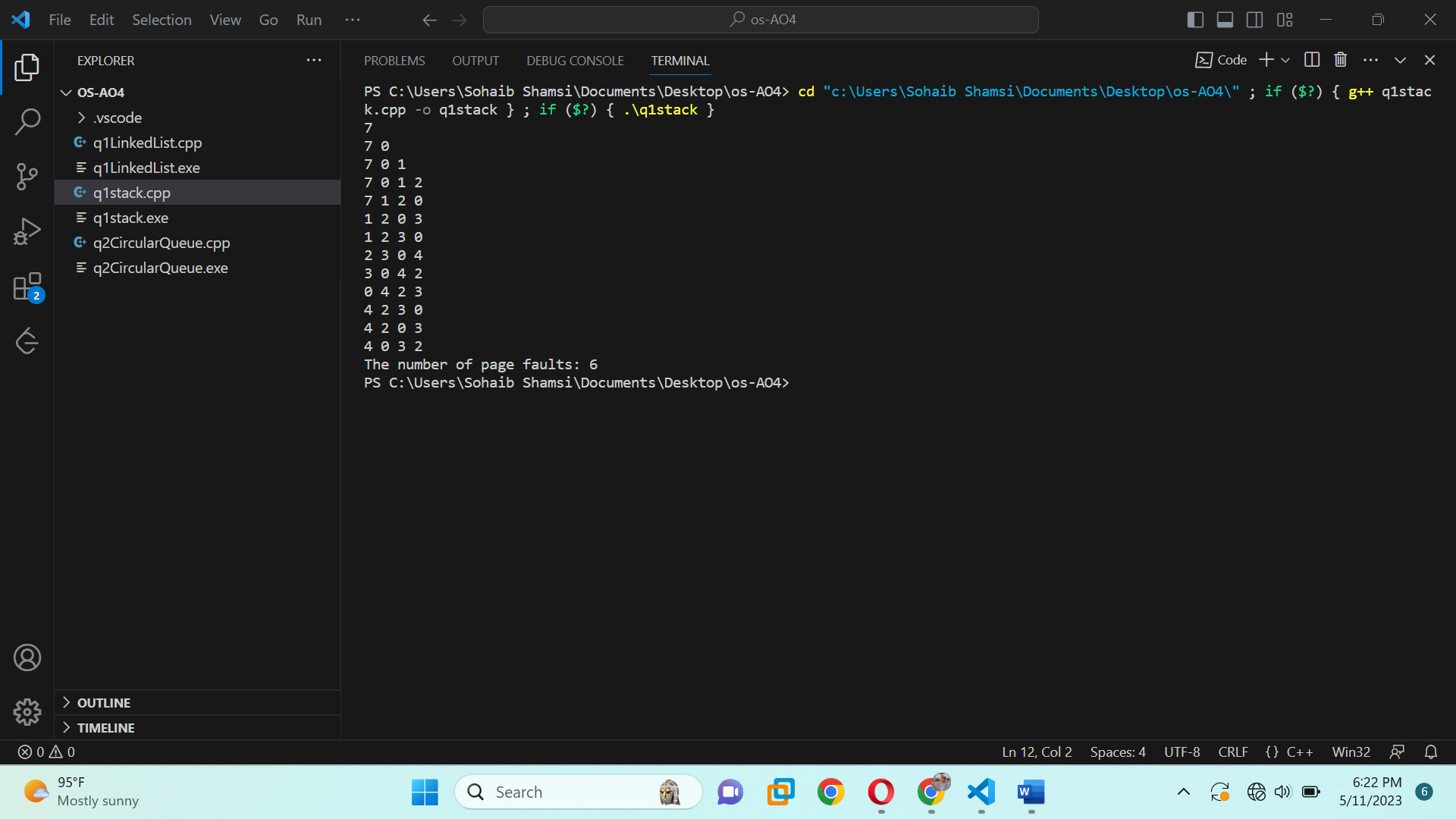
        }

        printStack(stack\_s, top);

    }

    cout << "The number of page faults: " << cnt;

}



Q1 Linked List

#include<bits/stdc++.h>

using namespace std;

class Node {

public:

    int val;

    Node\* next;

    Node(int val) {

        this->val = val;

        next = NULL;

    }

};

class LRU {

private:

    Node\* head;

    int count;

    int capacity;

    unordered\_map<int, Node\*> map;

public:

    int page\_faults;

    LRU(int capacity) {

        head = new Node(-1);

        count = 0;

        this->capacity = capacity;

        page\_faults=0;

    }

    void addNode(Node\* node) {

        node->next = head->next;

        head->next = node;

    }

    void deleteNode(Node\* prev) {

        Node\* temp = prev->next;

        prev->next = temp->next;

        map.erase(temp->val);

        delete temp;

    }

    void update(int val) {

        if (map.find(val) != map.end()) { // milagya

            // delete the last node

            Node\* prev = head;

            while (prev->next != NULL) {

                if (prev->next->val == val) {

                    break;

                }

                prev = prev->next;

            }

            Node\* temp = map[val];

            prev->next = temp->next;

            // add the node at the beginning

            addNode(temp);

        }

        else {

            page\_faults++;

            Node\* temp = new Node(val);

            if (count == capacity) {

                // delete the last node

                Node\* prev = head;

                while (prev->next->next != NULL) {

                    prev = prev->next;

                }

                deleteNode(prev);

                count--;

            }

            // add to the linked list

            addNode(temp);

            map[val] = temp;

            count++;

        }

    }

    void display() {

        Node\* temp = head->next;

        while (temp != NULL) {

            cout << temp->val << " ";

            temp = temp->next;

        }

        cout << endl;

    }

};

int main() {

    LRU x(4); // frame size of 4

    vector <int> arr = {7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2};

    int n = 13;

    for (int i = 0; i < n; i++) {

        x.update(arr[i]);

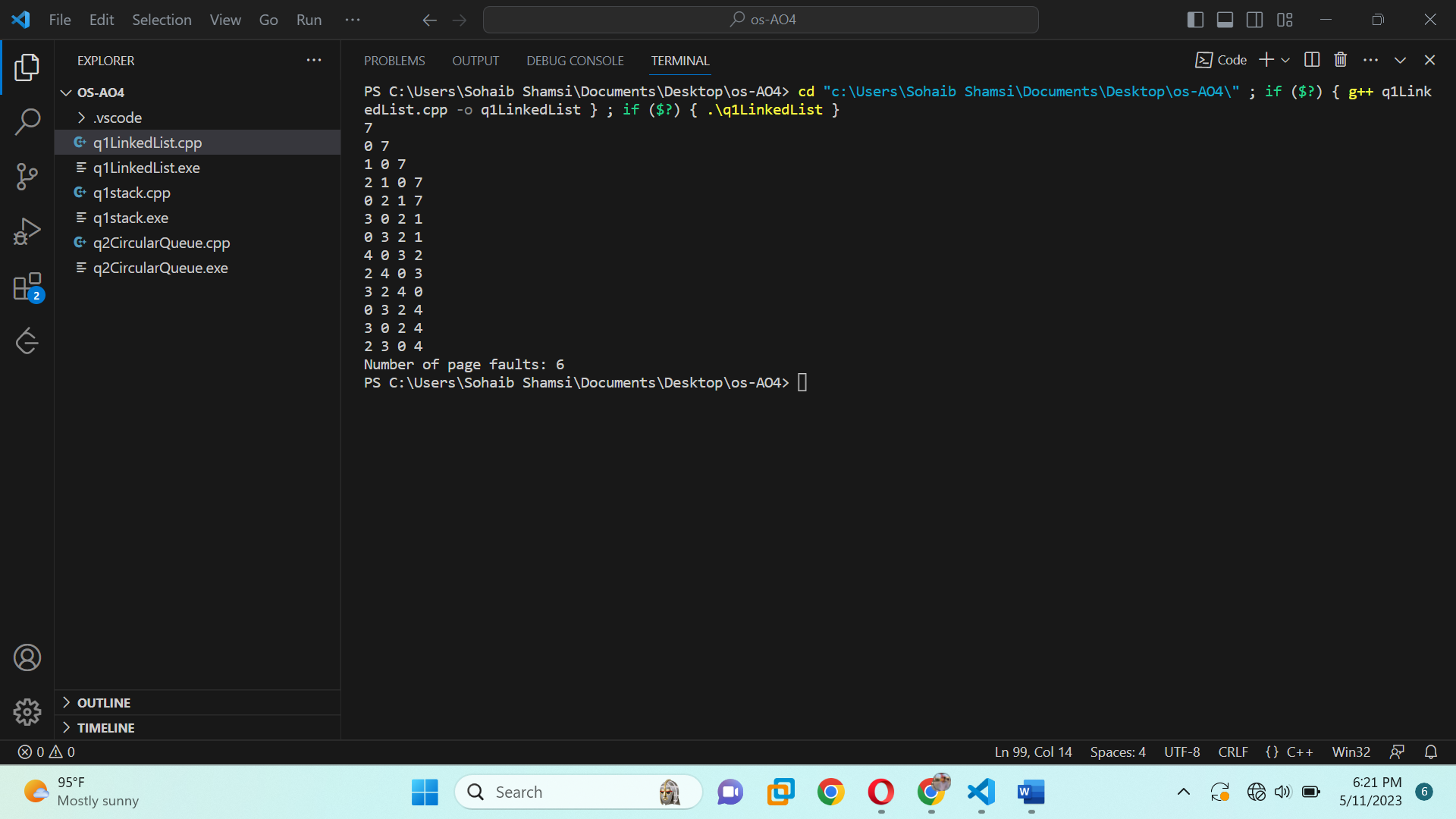
        x.display();

    }

    cout<<"Number of page faults: "<<x.page\_faults;

    return 0;

}



Q2. Circular Queue

#include <bits/stdc++.h>

using namespace std;

#define SIZE 3 // Frame Size

class Circular\_Queue

{

public:

    pair<char, pair<int, int>> arr[SIZE];

    int front, rear;

    Circular\_Queue()

    {

        front = rear = -1;

    }

    void print\_queue()

    {

        if (isEmpty())

        {

            cout << "Queue is empty" << endl;

            return;

        }

        cout << "Queue: ";

        int i;

        for (i = front; i != rear; i = (i + 1) % SIZE)

            cout << "(" << arr[i].first << "," << arr[i].second.first << "," << arr[i].second.second << "), ";

        cout << "(" << arr[i].first << "," << arr[i].second.first << "," << arr[i].second.second << ")" << endl;

    }

    int size\_queue()

    {

        if (front != -1)

        {

            return (rear - front + 1);

        }

        return 0;

    }

    bool isFull()

    {

        if (front == 0 && rear == SIZE - 1)

        {

            return true;

        }

        if (front == rear + 1)

        {

            return true;

        }

        return false;

    }

    bool isEmpty()

    {

        if (front == -1)

        {

            return true;

        }

        return false;

    }

    void Push(pair<char, pair<int, int>> element)

    {

        if (isFull())

        {

            cout << "Queue is full";

        }

        else

        {

            if (front == -1)

                front = 0;

            rear = (rear + 1) % SIZE;

            arr[rear] = element;

        }

    }

    char deQueue()

    {

        pair<char, pair<int, int>> element;

        if (isEmpty())

        {

            cout << "Queue is empty" << endl;

            return (-1);

        }

        else

        {

            element = arr[front];

            if (front == rear)

            {

                front = -1;

                rear = -1;

            }

            else

            {

                front = (front + 1) % SIZE;

            }

            return (element.first);

        }

    }

    int remove()

    {

        for (int i = front; i <= rear; i++)

        {

            if (arr[i].second.first == 0 && arr[i].second.second == 0)

            {

                return i;

            }

        }

        for (int i = front; i <= rear; i++)

        {

            if (arr[i].second.first == 0 && arr[i].second.second == 1)

            {

                return i;

            }

        }

        for (int i = front; i <= rear; i++)

        {

            if (arr[i].second.first > 0)

            {

                arr[i].second.first--;

            }

        }

        return -1;

    }

    void insert(int index, pair<char, pair<int, int>> element)

    {

        arr[index] = element;

    }

    int search(char element)

    {

        for (int i = front; i <= rear; i++)

        {

            if (arr[i].first == element)

            {

                return i;

            }

        }

        return -1;

    }

};

int main()

{

    string ref = "01362452502312524";

    string modified = "YNNNNYNNYNNNYNYNN";

    Circular\_Queue q;

    for (int i = 0; i < ref.size(); i++)

    {

        // q.print\_queue();

        int index = q.search(ref[i]);

        if (index == -1)

        {

            if (q.size\_queue() < SIZE)

            {

                if (modified[i] == 'Y')

                {

                    q.Push(make\_pair(ref[i], make\_pair(1, 1)));

                }

                else

                {

                    q.Push(make\_pair(ref[i], make\_pair(1, 0)));

                }

                cout << "Element " << ref[i] << " Pushed with thy status (" << 1 << "," << 0 << ")\n";

            }

            else

            {

                int check = q.remove();

                if (check == -1)

                {

                    check = q.remove();

                }

                cout << "Element " << q.arr[check].first << " Removed with the status (" << q.arr[check].second.first << ","

                     << q.arr[check].second.second << ")\n";

                if (modified[i] == 'Y')

                {

                    q.insert(check, make\_pair(ref[i], make\_pair(1, 1)));

                    cout << "Element " << ref[i] << " Pushed with the status (" << 1 << "," << 1 << ")\n";

                }

                else

                {

                    q.insert(check, make\_pair(ref[i], make\_pair(1, 0)));

                    cout << "Element " << ref[i] << " Pushed with the status (" << 1 << "," << 0 << ")\n";

                }

            }

        }

        else

        {

            if (modified[i] == 'Y')

            {

                q.insert(index, make\_pair(ref[i], make\_pair(1, 1)));

            }

            else

            {

                q.insert(index, make\_pair(ref[i], make\_pair(1, 0)));

            }

            cout << "Element " << q.arr[index].first << " Status updated to (" << q.arr[index].second.first << ","

                 << q.arr[index].second.second << ")\n";

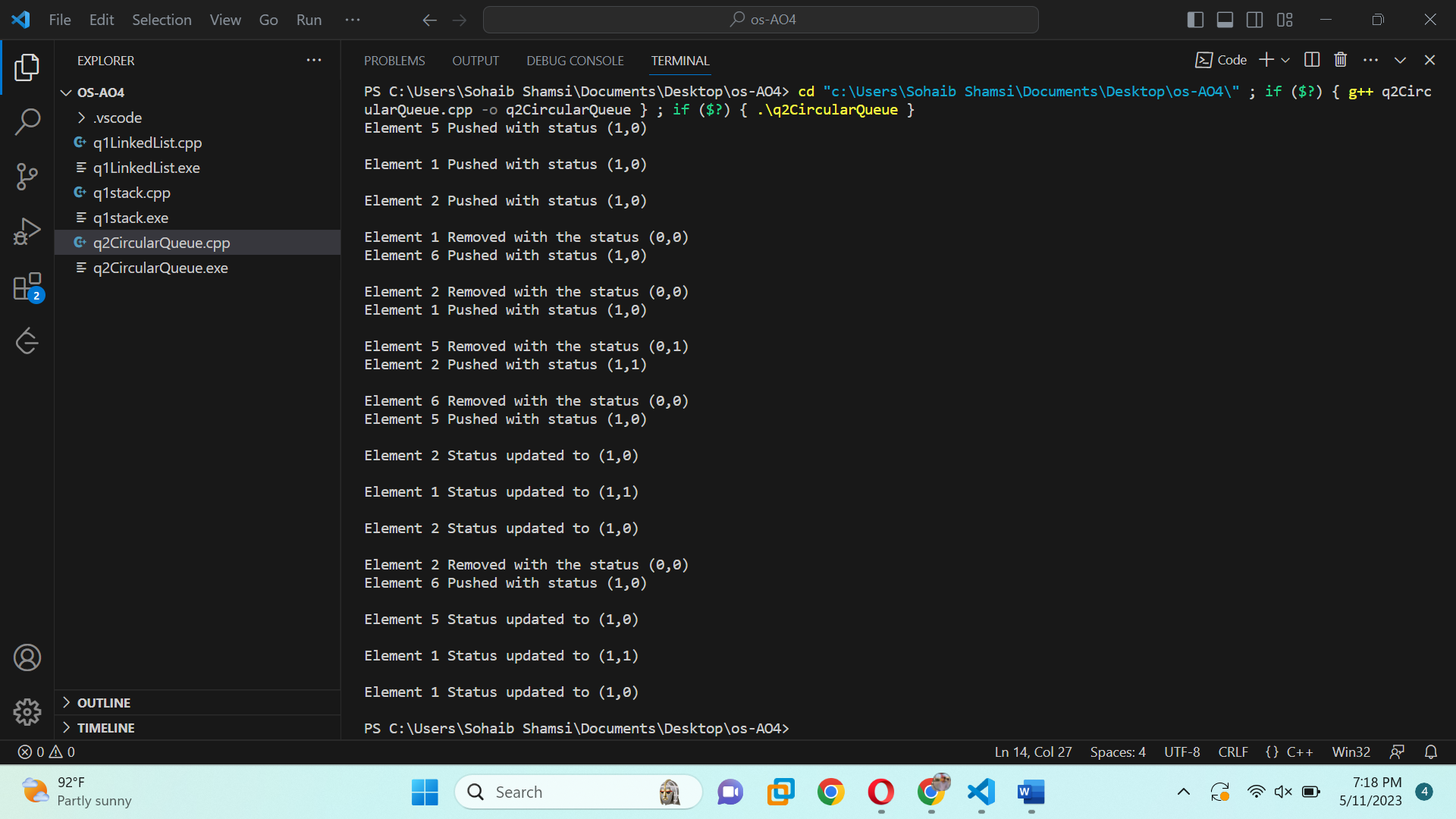
        }

        cout << endl;

    }

    return 0;

}



Q2 Circular Linked list

#include <bits/stdc++.h>

using namespace std;

#define SIZE 3

class Node

{

    public:

        pair<char, pair<int,int>> data;

        Node \*next;

        Node(pair<char, pair<int,int>> d)

        {

            data = d;

            next = NULL;

        }

};

class Circular\_Linked\_List

{

    public:

        Node \*head;

        Circular\_Linked\_List()

        {

            head = NULL;

        }

        int count\_nodes()

        {

            Node \*temp = head;

            int count=0;

            while (temp != NULL)

            {

                temp = temp->next;

                count++;

            }

            return count;

        }

        void Append\_Node(pair<char, pair<int,int>> data)

        {

            Node \*newnode = new Node(data), \*temp = head;

            if(temp == NULL)

            {

                head = newnode;

                newnode->next = head;

            }

            else

            {

                while(temp->next != head)

                {

                    temp = temp->next;

                }

                temp->next = newnode;

                newnode->next = head;

            }

        }

        bool Search(char d)

        {

            Node \*temp = head;

            while(temp != NULL)

            {

                if(temp->data.first == d)

                {

                    return true;

                }

            }

            return false;

        }

        void delete\_node(char pos)

        {

            Node \*temp = head;

            while(temp->next != head)

            {

                if(temp->next->data.first == pos)

                {

                    break;

                }

                temp = temp->next;

            }

            if(temp->next == NULL)

            {

                cout << "No node";

            }

            else

            {

                Node \*del = temp->next;

                temp->next = temp->next->next;

                delete del;

            }

        }

        char remove()

        {

            Node \*temp = head;

            while(temp != NULL)

            {

                if(temp->data.second.first == 0 && temp->data.second.second == 0){return temp->data.first;}

                temp = temp->next;

            }

            temp = head;

            while(temp != NULL)

            {

                if(temp->data.second.first == 0 && temp->data.second.second == 1){return temp->data.first;}

                temp = temp->next;

            }

            temp = head;

            while(temp != NULL)

            {

                if(temp->data.second.first > 0){temp->data.second.first--;}

                temp = temp->next;

            }

            return 'E';

        }

        void update(char s, int s1, int s2)

        {

            Node \*temp = head;

            while(temp != NULL)

            {

                if(temp->data.first == s)

                {

                    break;

                }

                temp = temp->next;

            }

            temp->data.second.first = s1;

            temp->data.second.second = s2;

        }

        // void print()

        // {

        //     Node \*temp = head;

        //     do

        //     {

        //         cout << temp->data << " -> ";

        //         temp = temp->next;

        //     }while(temp != head);

        // }

};

int main()

{

     string ref = "01362452502312524";

    string modified = "YNNNNYNNYNNNYNYNN";

    Circular\_Linked\_List ll;

    for(int i=0; i<ref.size(); i++)

    {

        bool check = ll.Search(ref[i]);

        if(check == false)

        {

            int nodes = ll.count\_nodes();

            if(nodes < SIZE)

            {

                if(modified[i] == 'Y'){ll.Append\_Node(make\_pair(ref[i], make\_pair(1,1)));}

                else{ll.Append\_Node(make\_pair(ref[i], make\_pair(1,0)));}

            }

            else

            {

                char remove = ll.remove();

                if(remove == 'E')

                {

                    remove = ll.remove();

                }

                ll.delete\_node(remove);

                if(modified[i] == 'Y'){ll.Append\_Node(make\_pair(ref[i], make\_pair(1,1)));}

                else{ll.Append\_Node(make\_pair(ref[i], make\_pair(1,0)));}

            }

        }

        else

        {

            if(modified[i] == 'Y'){ll.update(ref[i], 1, 1);}

            else{ll.update(ref[i], 1, 0);}

        }

    }

    return 0;

}