# Applied Programming



# Name: M. Hassan Sattar

# Roll No: 22F-3773

**Section: MS-1A**

# Submitted to: Sir. Muhammad Yousaf

# TASK 1

**Implement the following class.**

**class Stack**

**{**

**Node\* Top;**

**public:**

**Stack();**

**bool puch(char);**

**char pop();**

**bool isEmpty();**

**void display();**

**int calculator(string);//You are supposed to implement a basic calculator that will perform**

**addition, Subtraction, multiplication, Division on numbers from 0 to 9. It should follow the**

**BODMAS rule. Like (7+(6\*3+2)-(6/3)) = 25, Use STACK ADT to implement this.**

#include <iostream>

using namespace std;

struct Node

{

int data;

Node\* next;

};

class Stack

{

Node\* Top;

public:

Stack()

{

Top = NULL;

}

bool PUSH(char data)

{

Node\* temp = new Node;

if (temp == NULL)

return false;

temp->data = data;

temp->next = Top;

Top = temp;

return true;

}

char POP()

{

if (isEmpty())

return 0;

Node\* temp = Top;

Top = Top->next;

return temp->data;

}

bool isEmpty()

{

if (Top == NULL)

return true;

return false;

}

void display()

{

Node\* temp = Top;

// cout << "[ ";

while (temp != NULL)

{

cout << temp->data;

temp = temp->next;

}

// cout << "]";

cout << endl;

}

// Missing this function

char top()

{

return Top->data;

}

};

// INFIX\_To\_POSTFIX

int PRECDENCE\_OP(char op)

{

if (op == '+' || op == '-')

return 1;

if (op == '\*' || op == '/' || op == '%')

return 2;

return 0;

}

string INFIX\_To\_POSTFIX(string str)

{

Stack stack;

string result;

char c;

int size = str.length();

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

{

c = str[i];

if (/\*isalpha(c) || \*/ isdigit(c)) // We don't need a-z A-Z, Only digits are allowed.

result += c;

else if (c == '(')

stack.PUSH('(');

else if (c == ')')

{

while (!stack.isEmpty() && stack.top() != '(')

{

result += stack.POP();

}

if (stack.top() == '(')

{

stack.POP();

}

}

else

{

while (!stack.isEmpty() && PRECDENCE\_OP(str[i]) <= PRECDENCE\_OP(stack.top()))

{

result += stack.POP();

}

stack.PUSH(c);

}

}

while (!stack.isEmpty())

{

result += stack.POP();

}

return result;

}

int ADD\_OP(int a, int b, char op)

{

int result = 0;

switch (op)

{

case '+':

result = a + b;

break;

case '-':

result = a - b;

break;

case '\*':

result = a \* b;

break;

case '/':

result = a / b;

break;

case '%':

result = a % b;

break;

}

return result;

}

int EVALUTE(string str)

{

str = INFIX\_To\_POSTFIX(str);

cout << "Postfix: " << str << endl;

Stack values;

int value1, value2;

int size = str.length();

char c;

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

{

c = str[i];

if (/\*isalpha(c) || \*/ isdigit(c))

values.PUSH(c - '0'); // char->int

else

{

value1 = values.POP();

value2 = values.POP();

cout << "Evaluating: " << value1 << c << value2 << endl;

values.PUSH(ADD\_OP(value2, value1, c));

}

}

return values.POP();

}

int main()

{

string str = "(7+(6\*3+2)-(6/3))";

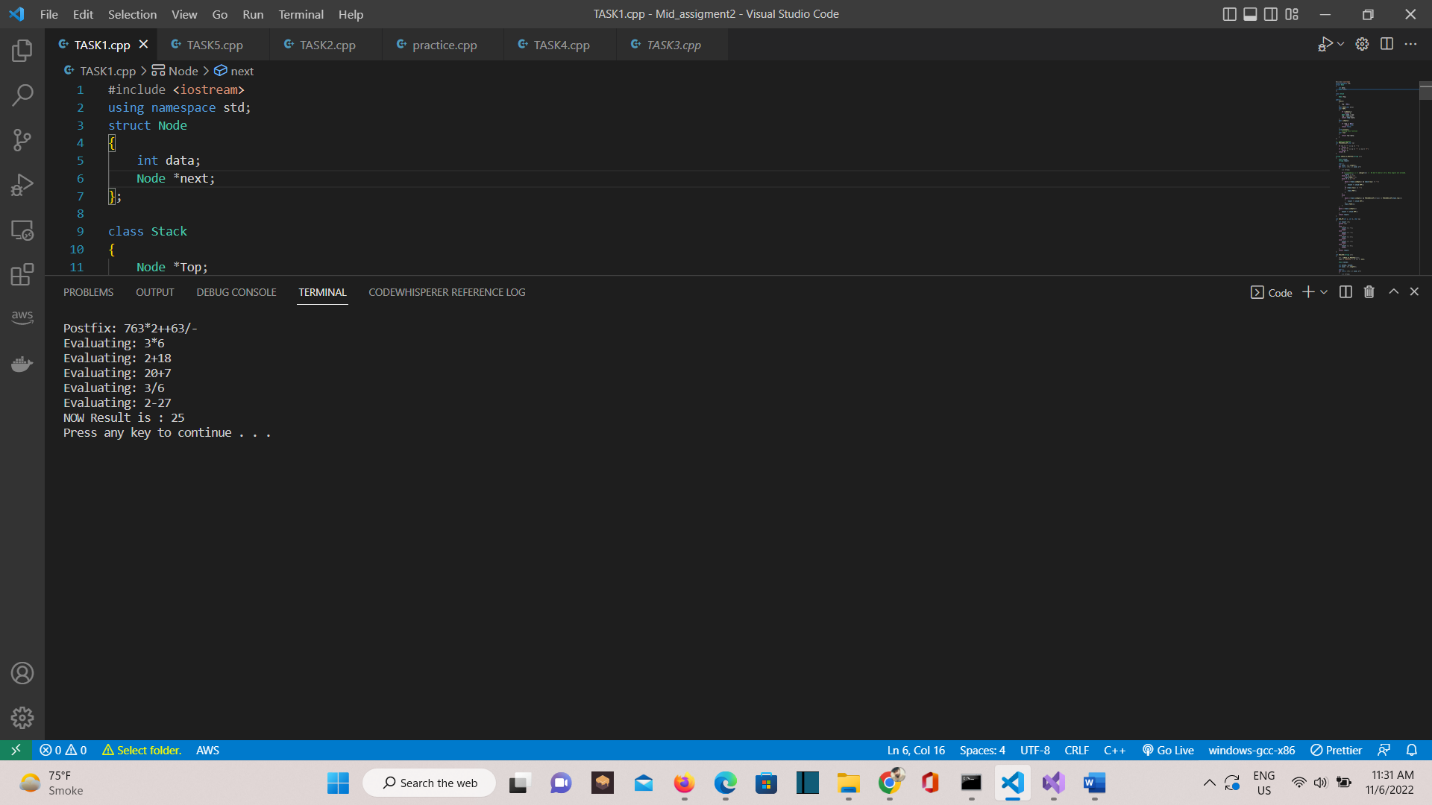
int result = EVALUTE(str);

cout << "NOW Result is : " << result << endl;

system("PAUSE");

return 0;

}

****

# TASK 2

**Suppose you are writing a messaging App that will change the sequences of words in the message, the changing mechanism is simple though, it removes the punctuation (if any) from the message for example, a message like below: “Hurray! Live for assignment.” would get changed to “assignment for live Hurray”. Use the ADT you have implemented in task 1 with implementation of following function.**

#include <iostream>

#include <string>

using namespace std;

class Node

{

private:

string data;

Node\* next;

public:

Node()

{

data = "";

next = NULL;

}

Node(string d)

{

data = d;

next = NULL;

}

void setData(string d)

{

data = d;

}

string GET\_DATA()

{

return data;

}

void SET\_NEXT(Node\* n)

{

next = n;

}

Node\* getNext()

{

return next;

}

};

class Stack

{

private:

Node\* head;

public:

Stack()

{

head = NULL;

}

bool isEmpty()

{

if (head == NULL)

{

return true;

}

return false;

}

void push(string s)

{

if (isEmpty())

{

head = new Node(s);

return;

}

Node\* n = new Node(s);

n->SET\_NEXT(head);

head = n;

}

string pop()

{

if (isEmpty())

{

return "Empty !!";

}

Node\* temp = head;

string re = temp->GET\_DATA();

head = head->getNext();

delete temp;

return re;

}

string top()

{

if (isEmpty())

{

return "Empty!!";

}

return head->GET\_DATA();

}

};

string SEQUEnCE(string msg)

{

Stack s;

for (int i = 0; i < msg.length(); i++)

{

if (msg[i] == ' ')

{

continue;

}

if (msg[i] == '.' || msg[i] == ',' || msg[i] == ':' || msg[i] == ';' || msg[i] == '?' || msg[i] == '!')

{

s.push(".");

continue;

}

string temp = "";

while (msg[i] != ' ' && i < msg.length())

{

if (msg[i] == '.' || msg[i] == ',' || msg[i] == ':' || msg[i] == ';' || msg[i] == '?' || msg[i] == '!')

{

break;

}

temp.push\_back(msg[i]);

i++;

}

s.push(temp);

}

string re = "";

while (!s.isEmpty())

{

string temp = s.pop();

if (temp == ".")

{

continue;

}

re = re + temp + " ";

}

return re;

}

int main()

{

string msg;

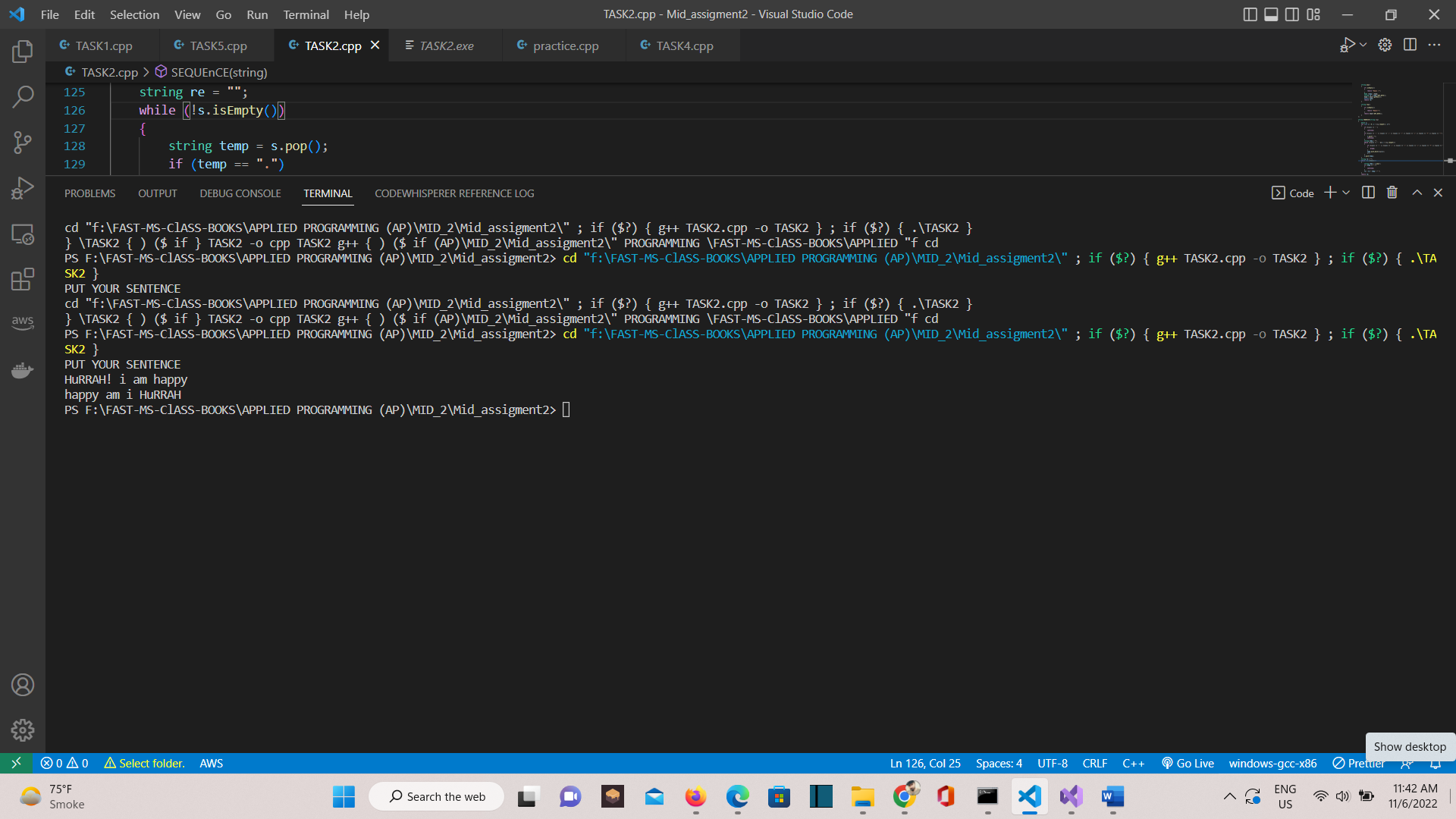
cout << "PUT YOUR SENTENCE\n";

getline(cin, msg);

cout << SEQUEnCE(msg) << endl;

return 0;

}

****

# TASK 3

**Imagine a scenario, you are on vacations and your manager asked you to help her with organizing crockery. You have three (03) empty cupboards and you placed all plates of five(05) different sizes in first cupboard with a sequence of largest set of plates at very bottom and the smallest plates at top, you asked your mother if it was fine and she said that you were supposed to place the plates in third cupboard not the first one so, now you are to move plates to third cupboard. You planned to do this by playing a game. The rules of the game are:**

** You can move only one set of plates at a time.**

** Set of larger plates cannot placed on a set of smaller plates.**

** You can move only upper most set of plates.**

**You are to implement a console-based game for above scenario. The main purpose of game is to move the plates in minimum moves by following the rules of the game. For better understanding**

#include <bits/stdc++.h>

using namespace std;

typedef struct Stack

{

int value;

int top;

int\* arr;

} Stack\_Node;

Stack\_Node\* Create\_stack(int value)

{

Stack\_Node\* stack = new Stack;

stack->value = value;

stack->top = -1;

stack->arr = new int[value];

return stack;

}

bool CHECK\_ISFull(Stack\_Node\* stack)

{

if (stack->top == stack->value - 1)

return true;

else

return false;

}

bool CHECK\_EMPTY(Stack\_Node\* stack)

{

if (stack->top == -1)

return true;

else

return false;

}

void push(Stack\_Node\* stack, int element)

{

if (CHECK\_ISFull(stack))

return;

stack->arr[++stack->top] = element;

}

int pop(Stack\_Node\* stack)

{

if (CHECK\_EMPTY(stack))

return -1;

return (stack->arr[stack->top--]);

}

void MOVE\_DISC(int disc, char from\_Rod, char to\_Rod)

{

cout << "Move the disc " << disc << " "

<< "from Rod '" << from\_Rod << "' to Rod '" << to\_Rod << "'" << endl;

}

void MOVE\_DISC\_Helper(struct Stack\* source, struct Stack\* dest, char s, char d)

{

int top1 = pop(source);

int top2 = pop(dest);

if (top1 == -1)

{

push(source, top2);

MOVE\_DISC(top2, d, s);

}

else if (top2 == -1)

{

push(dest, top1);

MOVE\_DISC(top1, s, d);

}

else if (top1 > top2)

{

push(source, top1);

push(source, top2);

MOVE\_DISC(top2, d, s);

}

else

{

push(dest, top2);

push(dest, top1);

MOVE\_DISC(top1, s, d);

}

}

void TOWER\_Of\_HANOI(int number\_of\_discs, struct Stack\* source, struct Stack\* aux, struct Stack\* dest)

{

char s = 'S', d = 'D', a = 'A';

// if n is even swap aux and dest

if (number\_of\_discs % 2 == 0)

{

char var = d;

d = a;

a = var;

}

int number\_of\_moves = pow(2, number\_of\_discs) - 1;

for (int i = number\_of\_discs; i >= 1; i--)

{

push(source, i);

}

// iteration of each i upto number of moves

for (int i = 1; i <= number\_of\_moves; i++)

{

if (i % 3 == 0)

MOVE\_DISC\_Helper(aux, dest, a, d);

else if (i % 3 == 2)

MOVE\_DISC\_Helper(source, aux, s, a);

else if (i % 3 == 1)

MOVE\_DISC\_Helper(source, dest, s, d);

}

}

int main()

{

int number\_of\_discs;

cin >> number\_of\_discs;

Stack\_Node\* source;

Stack\_Node\* dest;

Stack\_Node\* aux;

source = Create\_stack(number\_of\_discs);

aux = Create\_stack(number\_of\_discs);

dest = Create\_stack(number\_of\_discs);

TOWER\_Of\_HANOI(number\_of\_discs, source, aux, dest);

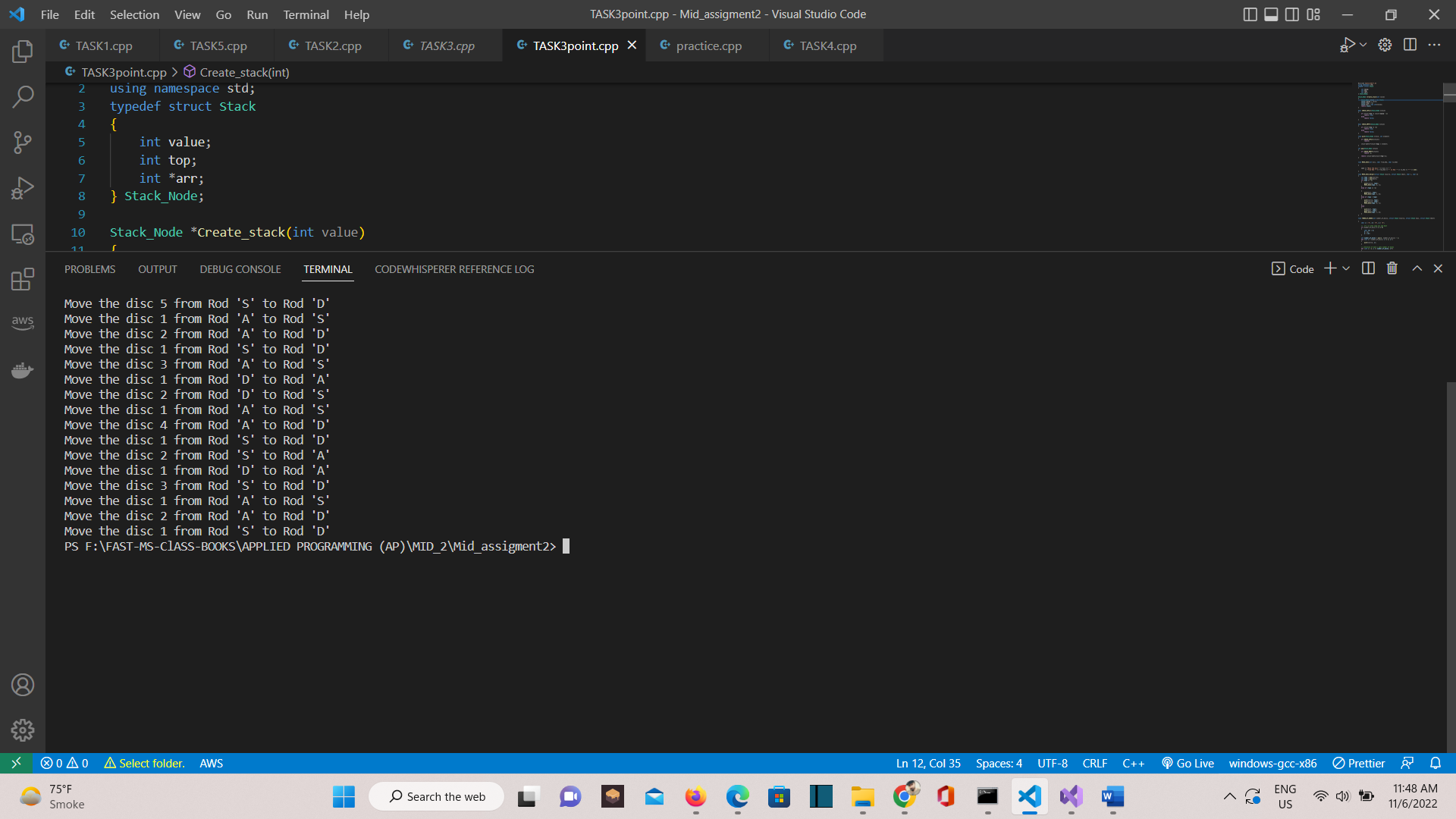
delete source;

delete aux;

delete dest;

return 0;

}

****

# TASK 4a

**Suppose you have joined the city traffic police (CTF) as a software engineer, a few days after your hiring a sudden traffic jam happened on Sargodha Road. The situation of the traffic jam is bad the vehicles are all stuck and more vehicles keep on coming that is causing the situation to get even worse. Some cars are leaving and joining this traffic jam from somewhere middle. Traffic police have arrived at the spot, and they have started to move traffic from both ends to get rid of it ASAP. You will be designing a program to deal with this situation while keeping a record of incoming and outgoing cars from traffic jams. You will be using an array-based double-ended queue (DEQUE) to implement this program that will show an appropriate message when all the cars are gone, and the situation is clear.**

#include <iostream>

using namespace std;

class Deque

{

private:

int\* Array;

int Size;

int front, rear;

public:

Deque(int s)

{

Size = s;

Array = new int[Size];

front = rear = -1;

}

bool isEmpty()

{

if (front == -1)

{

return true;

}

else

{

return false;

}

}

bool isFull()

{

if (((rear + 1) % Size) == front)

{

return true;

}

else

{

return false;

}

}

bool insertion\_at\_front(int data)

{

if (isFull())

{

cout << "Deque OverFlow." << endl;

return false;

}

else if (isEmpty())

{

front = rear = 0;

}

else if (front == 0)

{

front = Size - 1;

}

else

{

front = (front - 1) % Size;

}

}

bool insertion\_at\_rear(int data)

{

if (isFull())

{

cout << " Deque OverFlow." << endl;

return false;

}

else if (isEmpty())

{

front = rear = 0;

}

else

{

rear = (rear + 1) % Size;

}

Array[rear] = data;

return true;

}

bool deletion\_at\_front()

{

if (isEmpty())

{

cout << "All the cars are removed ." << endl;

return false;

}

else if (front == rear)

{

front = rear = -1;

}

else

{

Array[rear] = 0;

front = (front + 1) % Size;

}

}

bool deletion\_at\_rear()

{

if (isEmpty())

{

cout << "All the cars are removed ." << endl;

return false;

}

else if (front == rear)

{

front = rear = -1;

}

else

{

Array[rear] = 0;

rear = (rear + 1) % Size;

}

}

void Display()

{

cout << " Insertion";

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

{

cout << Array[i] << " ";

}

}

};

int main()

{

int size;

cout << " Enter the total number of cars : ";

cin >> size;

Deque o1(size);

int data, index;

while (true)

{

cout << " Enter 1 : to insert at front" << endl;

cout << " Enter 2 : to insert at rear" << endl;

cout << " Enter 3 : to delete at front" << endl;

cout << " Enter 4 : to delete at rear" << endl;

cout << " Enter 5 : to display deque" << endl;

cout << " Enter 6 : to exit the program" << endl;

cout << " Enter your choice : ";

cin >> index;

if (index == 1)

{

cout << "Enter the car " << endl;

cin >> data;

o1.insertion\_at\_front(data);

}

else if (index == 2)

{

cout << "Enter the car " << endl;

cin >> data;

o1.insertion\_at\_rear(data);

}

else if (index == 3)

{

o1.deletion\_at\_front();

}

else if (index == 4)

{

o1.deletion\_at\_rear();

}

else if (index == 5)

{

o1.Display();

}

else

{

exit;

}

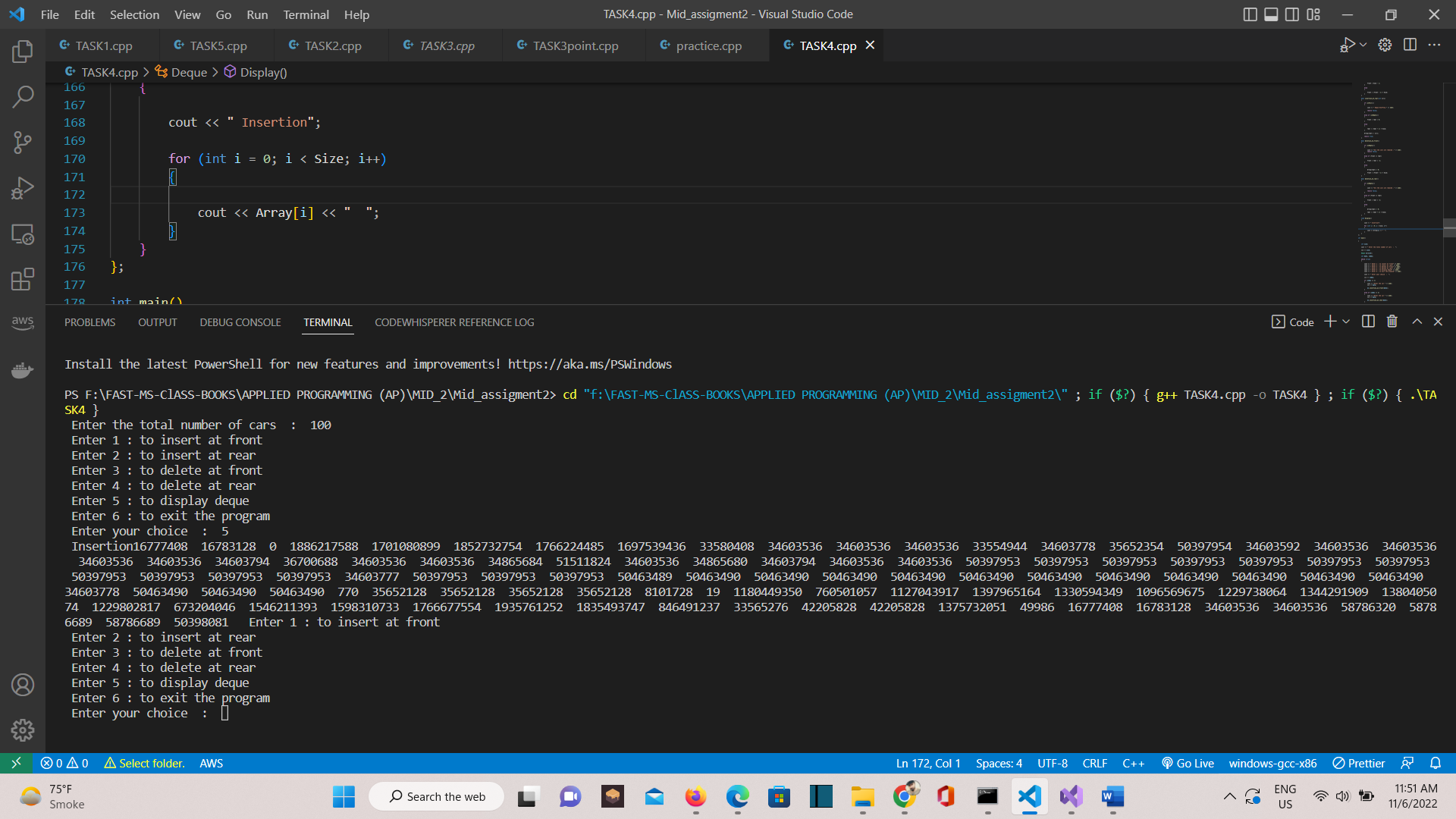
}

cout << endl;

system("pause");

return 0;

}

****

# TASK 5

**Suppose we have two parties: The Government and the Opposition. The senate consists of senators coming from these two parties. Now the senate wants to decide about an amendment in the law. The voting for this change is a round-based procedure. In each round, each senator can exercise one of the two rights:**

**1. Ban one senator's right:**

**A senator can make another senator lose all his rights in this and all the following**

**rounds.**

**2. Announce the victory:**

**If this senator found the senators who still have rights to vote are all from the same**

**party, he can announce the victory and make the decision about amendment in the law.**

**You have n senators, and you must deal with them using Queue ADT. The character ‘G’ and ‘O’ will represent Government, and the Opposition respectively the belonging party of senator.**

**The round-based procedure starts from the first senator to the last senator in the given order. This procedure will last until the end of voting. All the senators who have lost their rights will be skipped during the procedure. Suppose every senator is smart enough and will play the best strategy for his own party, you need to predict which party will finally announce the victory and make the amendment. The output should be Government or Opposition.**

#include<iostream>

#include<string>

using namespace std;

class node

{

public:

char data;

node\* next;

};

class que

{

node\* front, \* rear;

int num;

public:

que()

{

front = NULL; rear = NULL; num = 0;

}

bool isEmpty()

{

return (num == 0);

}

void enque(char x)

{

node\* nNode;

nNode = new node;

nNode->data = x;

nNode->next = NULL;

if (isEmpty())

{

front = nNode; rear = nNode;

}

else

{

rear->next = nNode; nNode->next = NULL; rear = nNode;

}

num++;

}

string deque()

{

string ans;

node\* temp;

if (isEmpty()) { cout << "\nUnderflow\n"; return 0; }

ans = front->data;

temp = front->next;

delete front;

front = temp;

num--;

return ans;

}

void checkWinner(que q)

{

bool check = 0, flag = 1;

int i = 0;

int oCount = 0, gCount = 0;

node\* current = front;

while (current != NULL)

{

if (current->data == 'o' || current->data == 'O') {

oCount++;

i++;

check = true;

}

else {

gCount++;

i++;

flag = false;

}

if (i == 1)

{

if (check)

oCount++;

else

gCount++;

}

deque();

current = current->next;

}

if (oCount > gCount)

cout << "\n\n\t -> Winner of eLECTION IS OPPOSITION <-\n\n";

else if (gCount > oCount)

cout << "\n\n\t-> Winner of eLECTION IS GOVERNMENT <-\n\n";

}

void makeNULL()

{

int x;

while (!isEmpty())

deque();

}

~que()

{

makeNULL();

}

};

int main()

{

int x;

char val;

// string s="ogo";

que obj;

cout << "\nHow many pEople are in Senate : "; cin >> x;

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

{

cout << "\tEnter Belongings For Person " << i + 1 << " : "; cin >> val;

obj.enque(val);

}

cout << "\nTotal NUmber of Persons who Takes part in Voting Are : " << x << endl << endl;

cout << endl << "\nThe Votes are in Order :\n\n\t\t";

while (!obj.isEmpty())

{

cout << obj.deque() << " ";

}

cout << endl;

obj.checkWinner(obj);

while (!obj.isEmpty())

{

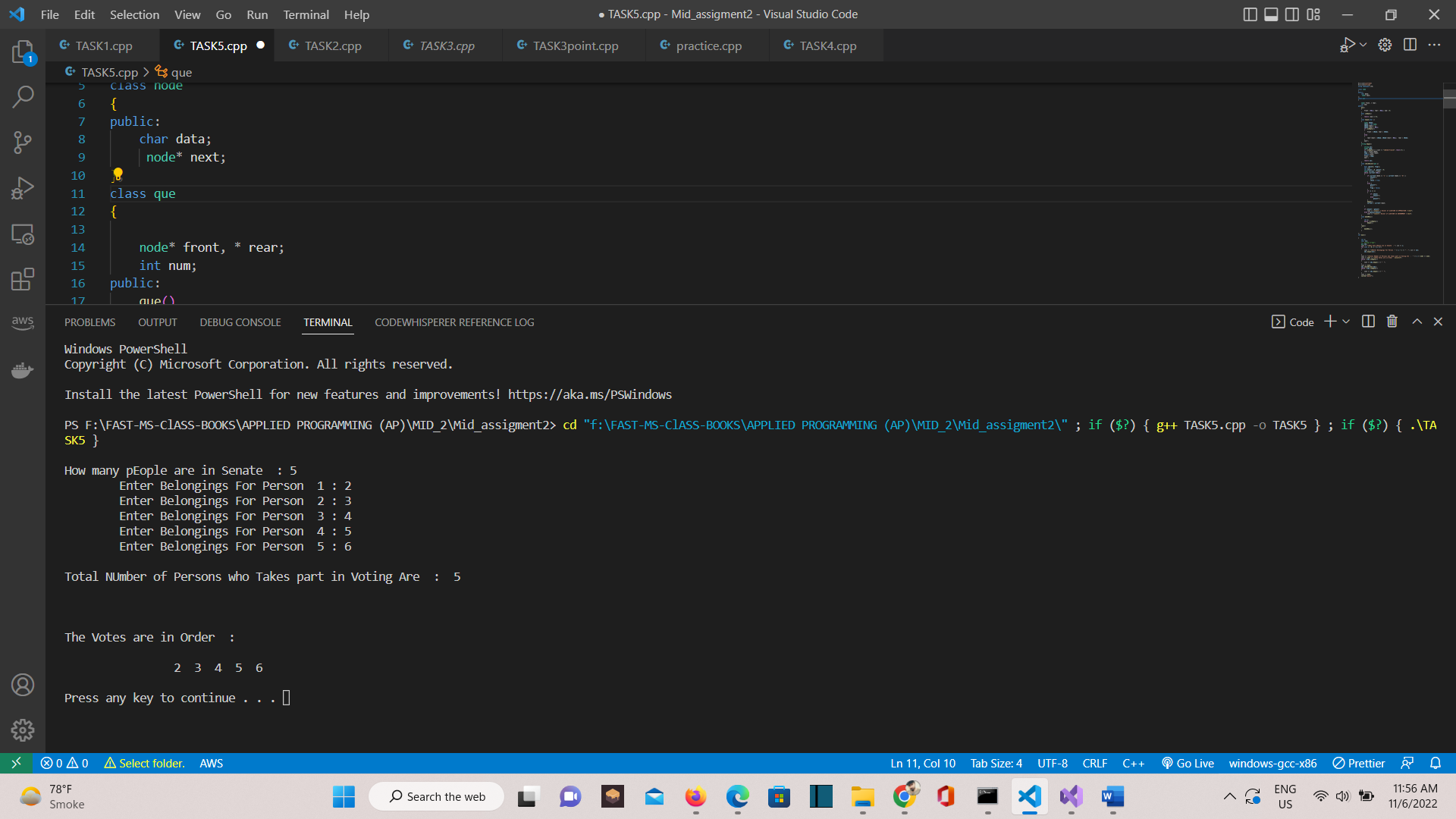
cout << obj.deque() << " ";

}

cout << endl;

system("pause");

}

****

# TASK 6

**Assume that each person is staying for 2 seconds on the counter and update the queues. You**

**should also keep the record of all the visitors and save it into file. When a person is processed its record should be added into file can be 1 or multiple as per your choice. Note: There is a limit of at most 15 persons in each queue, if the limit reaches and more visitors are coming you should display an appropriate message to ask them to wait. You will be using Queue ADT to implement above functionalities. If you think that some other Data structure in addition to the queue can be fruitful you can use that too but use of queue is compulsory.**

**i.e: After processing “Person 0” of “Queue n”, the queue moves forward like this:**

**1. “Person 0” of “Queue n-1” leaves his/her queue and enters in “Queue n”.**

**2. “Person 0” of “Queue n-2” leaves his/her queue and enters in “Queue n-1”.**

**3. “Person 0” of “Queue n-3” leaves his/her queue and enters in “Queue n-2”.**

**4. This keeps happening all the way to queue 0 and at the end, “Person 0” of “Queue 0”**

**leaves his/her queue and enters in “Queue 1”. After the queue is moved one-step forward, here’s how it looks:**

**By the end, the cashier processes all the people in the queues so that everyone can get their**

**money and manage their expenses. You are going to simulate the above explained process using Queues ADT in C++. To implement this program, you will also be creating a Template Queue Class all by yourself to keep it generic and to create queues of any-types. The flow of the program will be like this: User inputs an integer and N queues (<int>) are created. After creating N queues, you will enqueue some number of persons. For your ease, you can use same number of persons in every queue (minimum number of people in a queue must be 10). The ticket collector starts to process persons in “Queue n” one by one until all the persons are processed OR all the queues are empty.**

#include <iostream>

using namespace std;

struct Person

{

int ID;

string name;

Person\* next;

};

class BankSystem

{

private:

Person\* person;

int limit, current\_size;

bool CanAddMore()

{

if (limit == -1 || limit == 0)

return true;

return (current\_size != limit);

}

bool IsEmpty()

{

return current\_size == 0;

}

public:

BankSystem()

{

person = NULL;

limit = -1;

current\_size = 0;

}

void Insert(int ID, string name)

{

if (!CanAddMore())

{

cout << "queue is full..." << endl;

return;

}

Person\* p = new Person();

p->ID = ID;

p->name = name;

p->next = NULL;

if (person == NULL)

{

person = p;

}

else

{

Person\* temp = person;

// p = 80->NULL

// temp = 10->20->30->40->50->60->70->NULL

while (temp->next != NULL)

temp = temp->next;

// temp = 70->NULL

temp->next = p;

// temp = 70->80->NULL

}

if (limit != -1)

{

current\_size++;

}

}

Person\* GetFirstPersonFromQueue()

{

return person;

}

void Serve()

{

// 0th-case: Do not do anything, if queue is empty

if (IsEmpty())

{

cout << "queue is empty..." << endl;

return;

}

// 1st-case: Get first person from queue

Person\* p = GetFirstPersonFromQueue();

// 2nd-case: Perform some action

cout << "Say; Cashier is serving person. " << p->ID << ", " << p->name << endl;

// 3rd-case: Remove from queue

// person = 10->20->30->40->50->60->70->NULL, 20

// p = 10->20->30->40->50->60->70->NULL, 10

person = person->next;

p->next = NULL;

delete p;

if (limit != -1)

{

current\_size--;

}

}

void Display()

{

Person\* p = person;

while (p != NULL)

{

cout << "ID: " << p->ID << ", Name: " << p->name << endl;

p = p->next;

}

}

void SetLimit(int limit)

{

this->limit = limit;

}

};

int main()

{

BankSystem cashier;

cashier.SetLimit(5);

cashier.Serve();

cashier.Insert(1, "Person-1");

cashier.Insert(2, "Person-2");

cashier.Insert(3, "Person-3");

cashier.Insert(4, "Person-4");

cashier.Insert(5, "Person-5");

cashier.Insert(6, "Person-6");

cashier.Serve();

cashier.Insert(6, "Person-6");

cashier.Serve();

cashier.Serve();

cashier.Serve();

cashier.Serve();

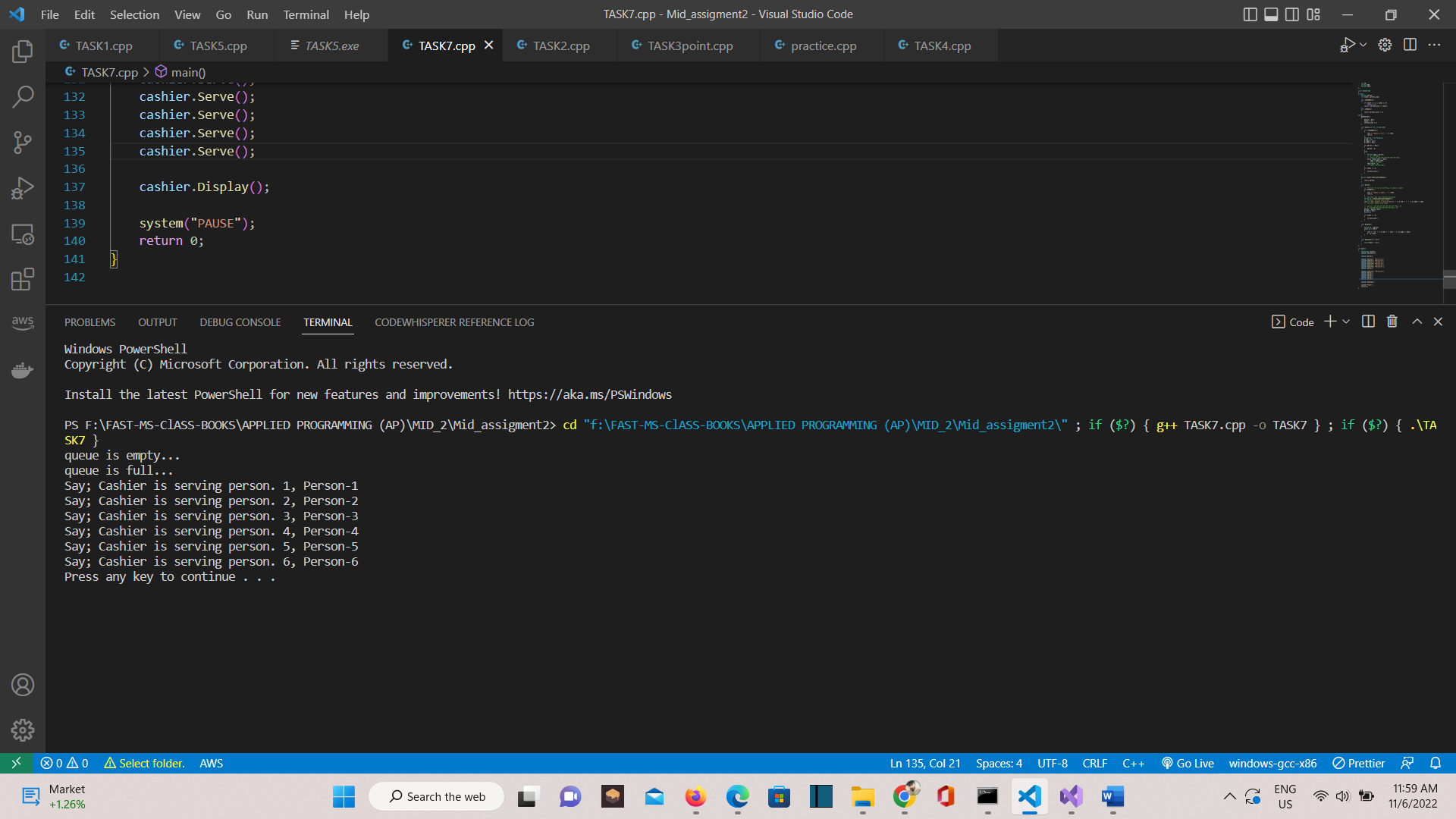
cashier.Serve();

cashier.Display();

system("PAUSE");

return 0;

}

****