Lab Task

Task # 1  
Create a simple queue with five nodes containing data {3, 8, 5, 9, and 16}.  
Task # 2  
Create a function which prints the queue in reverse manner.

#include <iostream>

using namespace std;

class node {

public:

int data;

node\* next;

};

node\* front = NULL;

node\* back = NULL;

node\* top\_temp = NULL;

class list {

public:

node\* create\_node() {

node\* newnode = new node;

return newnode;

}

void read\_data() {

node\* newnode = create\_node();

cout << "ENTER THE DATA : ";

cin >> newnode->data; cout << endl;

enqueue(newnode);

system("CLS");

}

node\* find\_front() {

node\* temp = back;

if (temp == NULL) {

return NULL;

}

else {

while (temp->next != NULL) {

temp = temp->next;

}

return temp;

}

}

void enqueue(node\* newnode) {

node\* temp\_front = find\_front();

if (front == NULL) {

front = newnode;

back = front;

front->next = back;

back->next = NULL;

}

else {

back->next = newnode;

back = newnode;

back->next = NULL;

}

}

void dequeue() {

//node\* find\_sec = find\_sec\_in\_queue();

if (front == NULL) {

cout << "THE LIST IS EMPTY" << endl;

}

else if (front->next == NULL) {

delete(front);

back = NULL;

front = NULL;

}

else {

node\* temp = front;

front = front->next;

delete(temp);

}

}

node\* find\_sec\_in\_queue() {

node\* temp\_front = find\_front();

if (temp\_front == NULL) {

return NULL;

}

else if (back->next == NULL) {

return back;

}

else {

node\* temp = back;

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

temp = temp->next;

}

return temp;

}

}

void ins\_stack(int inp) {

node\* newnode = create\_node();

newnode->data = inp;

if (top\_temp == NULL) {

top\_temp = newnode;

top\_temp->next = NULL;

}

else {

newnode->next = top\_temp;

top\_temp = newnode;

}

}

void display\_original() {

node\* temp\_front = find\_front();

if (temp\_front == NULL) {

cout << "THE LIST IS EMPTY" << endl;

}

else {

node\* temp = front;

while (temp != NULL) {

cout << temp->data << endl;

temp = temp->next;

}

}

system("pause");

system("CLS");

}

void display\_reverse() {

node\* temp\_front = find\_front();

if (temp\_front == NULL) {

cout << "THE LIST IS EMPTY" << endl;

}

else {

node\* temp = front;

while (temp != NULL) {

ins\_stack(temp->data);

temp = temp->next;

}

node\* temp\_disp\_stack = top\_temp;

while (temp\_disp\_stack != NULL) {

cout << temp\_disp\_stack->data << endl;

temp\_disp\_stack = temp\_disp\_stack->next;

}

top\_temp = NULL;

}

system("pause");

system("CLS");

}

bool is\_empty() {

node\* temp\_front = find\_front();

if (temp\_front == NULL) {

return 1;

}

else {

return 0;

}

}

};

int main() {

list l;

int option;

while (1) {

cout << "ENTER 1 TO INSERT IN QUEUE" << endl;

cout << "ENTER 2 TO DISPLAY IN ORDER" << endl;

cout << "ENTER 3 TO DEQUEUE" << endl;

cout << "ENTER 4 TO CHECK IF QUEUE IS EMPTY" << endl;

cout << "ENTER 5 TO DISPLAY IN REVERSE" << endl;

cout << "ENTER 6 TO EXIT THE PROGRAM" << endl;

cin >> option; cout << endl;

switch (option) {

case 1:

{

system("CLS");

l.read\_data();

break;

}

case 2:

{

system("CLS");

l.display\_original();

break;

}

case 3:

{

system("CLS");

l.dequeue();

break;

}

case 4:

{

system("CLS");

if (l.is\_empty()) {

cout << "THE LIST IS EMPTY" << endl;

}

else {

cout << "THE LIST IS NOT EMPTY" << endl;

}

break;

}

case 5:

{

system("CLS");

l.display\_reverse();

break;

}

case 6:

{

exit(0);

break;

}

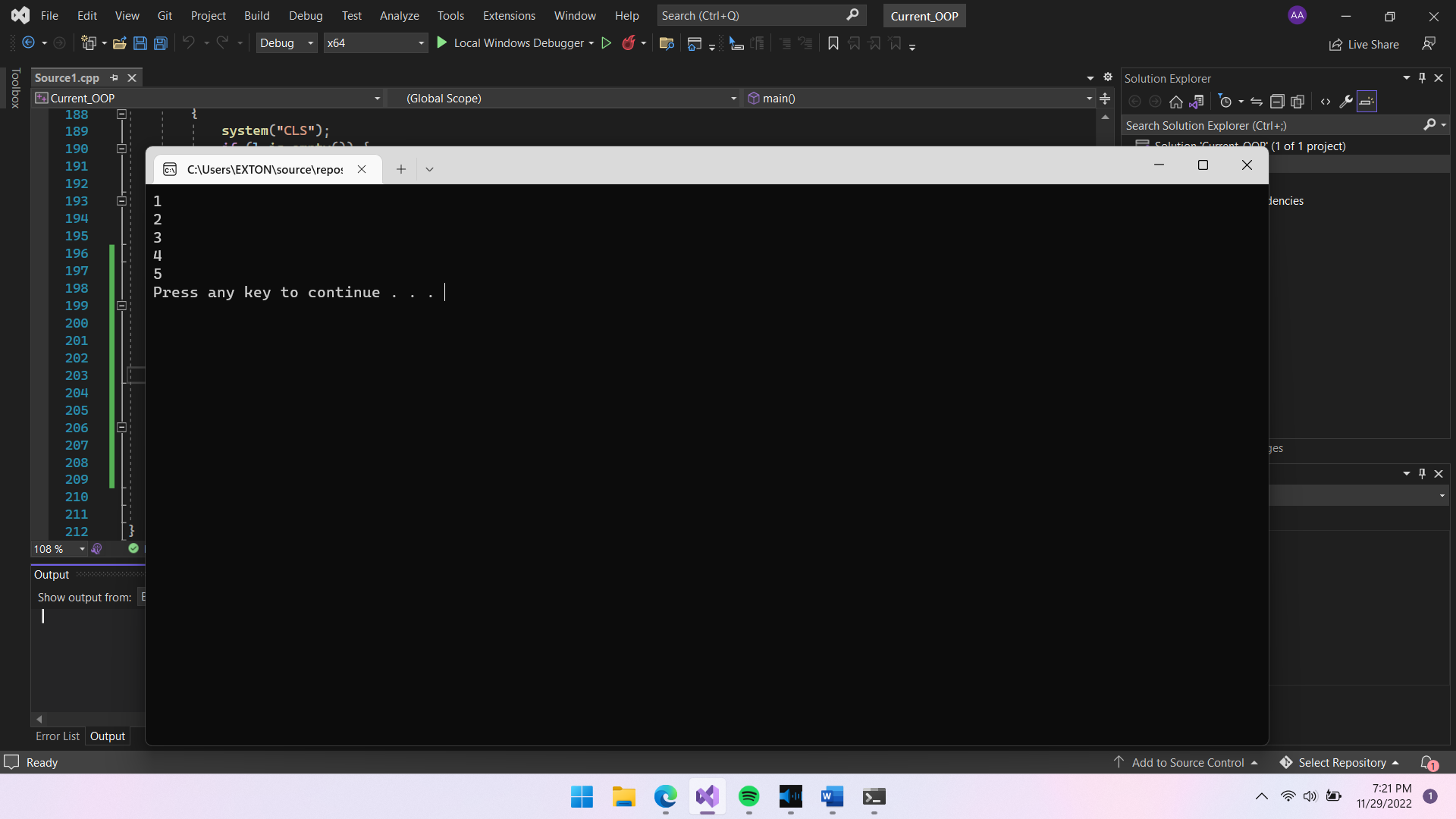
}

}

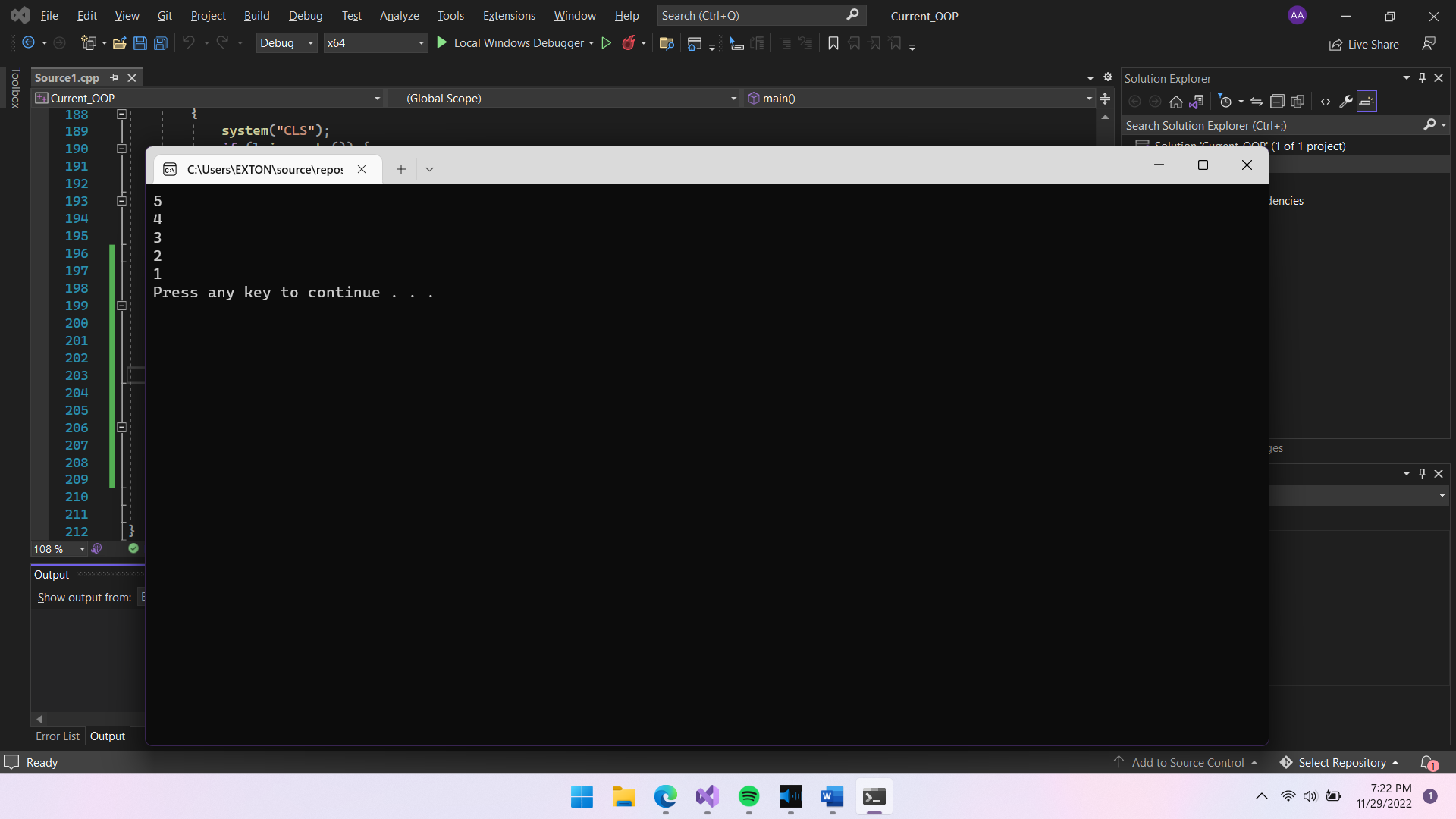
}

OUTPUT:

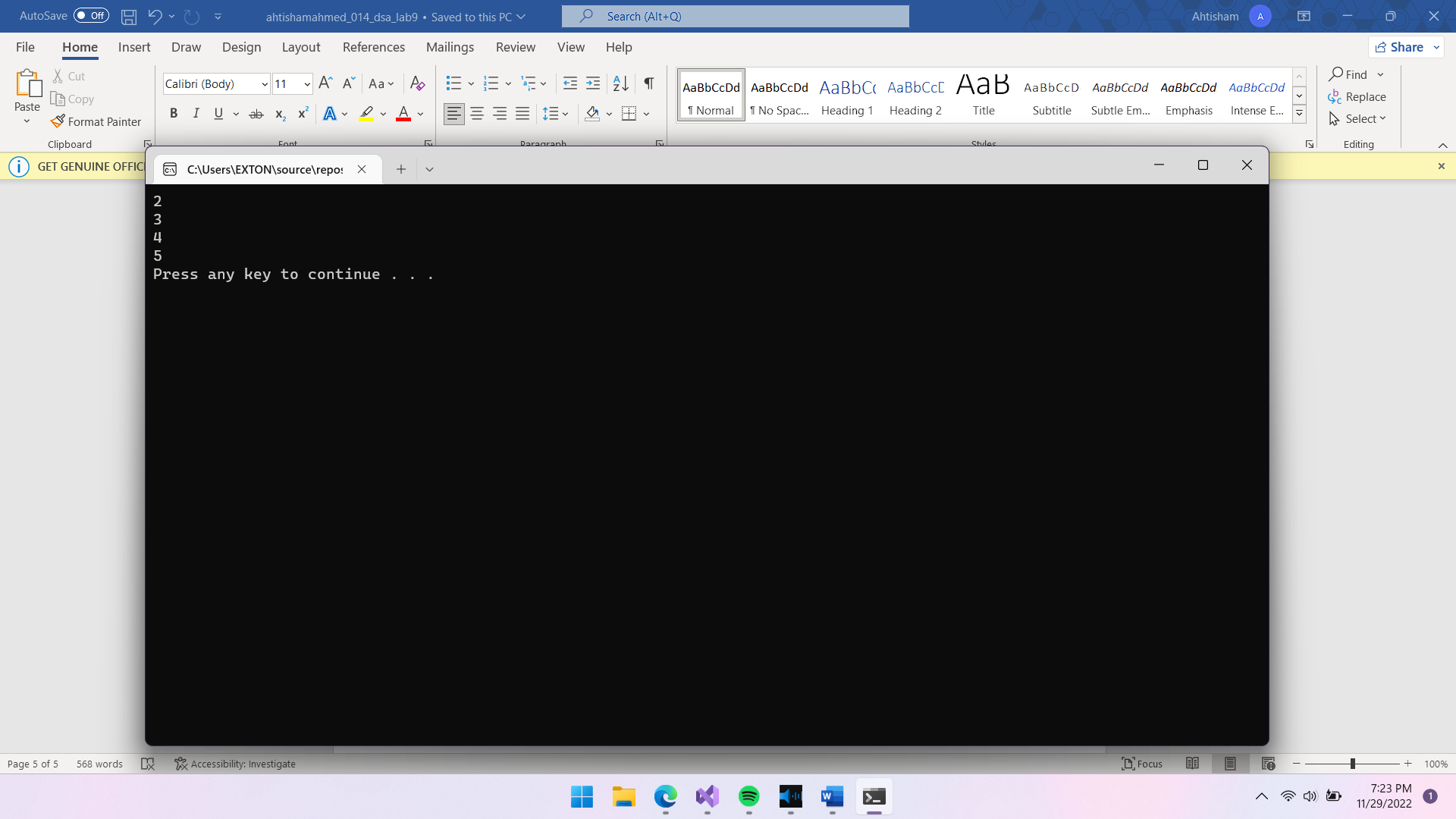
STANDARD DISPLAY:



REVERSE DISPLAY:



DEQUEUE:



Task # 2  
Create a new project having two interfaces one for Queue using Array and the   
other one is for Queue using Link List. User will choose the one he/she   
wants to use.

#include <iostream>

using namespace std;

class node {

public:

int data;

node\* next;

};

node\* front = NULL;

node\* back = NULL;

node\* top\_temp = NULL;

class list {

public:

node\* create\_node() {

node\* newnode = new node;

return newnode;

}

void read\_data() {

node\* newnode = create\_node();

cout << "ENTER THE DATA : ";

cin >> newnode->data; cout << endl;

enqueue(newnode);

system("CLS");

}

node\* find\_front() {

node\* temp = back;

if (temp == NULL) {

return NULL;

}

else {

while (temp->next != NULL) {

temp = temp->next;

}

return temp;

}

}

void enqueue(node\* newnode) {

node\* temp\_front = find\_front();

if (front == NULL) {

front = newnode;

back = front;

front->next = back;

back->next = NULL;

}

else {

back->next = newnode;

back = newnode;

back->next = NULL;

}

}

void dequeue() {

//node\* find\_sec = find\_sec\_in\_queue();

if (front == NULL) {

cout << "THE LIST IS EMPTY" << endl;

}

else if (front->next == NULL) {

delete(front);

back = NULL;

front = NULL;

}

else {

node\* temp = front;

front = front->next;

delete(temp);

}

}

node\* find\_sec\_in\_queue() {

node\* temp\_front = find\_front();

if (temp\_front == NULL) {

return NULL;

}

else if (back->next == NULL) {

return back;

}

else {

node\* temp = back;

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

temp = temp->next;

}

return temp;

}

}

void display\_original() {

node\* temp\_front = find\_front();

if (temp\_front == NULL) {

cout << "THE LIST IS EMPTY" << endl;

}

else {

node\* temp = front;

while (temp != NULL) {

cout << temp->data << endl;

temp = temp->next;

}

}

system("pause");

system("CLS");

}

void ins\_stack(int inp) {

node\* newnode = create\_node();

newnode->data = inp;

if (top\_temp == NULL) {

top\_temp = newnode;

top\_temp->next = NULL;

}

else {

newnode->next = top\_temp;

top\_temp = newnode;

}

}

void display\_reverse() {

node\* temp\_front = find\_front();

if (temp\_front == NULL) {

cout << "THE LIST IS EMPTY" << endl;

}

else {

node\* temp = front;

while (temp != NULL) {

ins\_stack(temp->data);

temp = temp->next;

}

node\* temp\_disp\_stack = top\_temp;

while (temp\_disp\_stack != NULL) {

cout << temp\_disp\_stack->data << endl;

temp\_disp\_stack = temp\_disp\_stack->next;

}

top\_temp = NULL;

}

system("pause");

system("CLS");

}

bool is\_empty() {

node\* temp\_front = find\_front();

if (temp\_front == NULL) {

return 1;

}

else {

return 0;

}

}

};

//A R R A Y

class list\_array {

public:

static int front\_arr;

static int back\_arr;

int read\_data() {

int i;

cout << "ENTER THE DATA : "; cin >> i; cout << endl;

return i;

}

void insert(int\* arr, int data\_inp, int size\_arr) {

if (back\_arr == size\_arr || back\_arr == size\_arr - 1) {

cout << "QUEUE OVERFLOW" << endl;

}

else if (front\_arr == -1) {

front\_arr++;

back\_arr++;

arr[back\_arr] = data\_inp;

}

else if (front\_arr > 0) {

back\_arr++;

arr[back\_arr] = data\_inp;

}

else {

int traverse = 0;

while (traverse != back\_arr) {

traverse++;

}

arr[traverse + 1] = data\_inp;

back\_arr++;

}

}

void display(int\* arr) {

int traverse = front\_arr;

while (traverse != back\_arr + 1) {

cout << arr[traverse] << endl;

traverse++;

}

}

void dequeue(int\* arr) {

if (front\_arr == -1) {

cout << "Queue Underflow" << endl;

}

else if (front\_arr == back\_arr) {

front\_arr = -1;

back\_arr = -1;

}

else {

front\_arr++;

}

}

};

int list\_array::front\_arr = -1;

int list\_array::back\_arr = -1;

int main() {

list l; list\_array l\_a;

int option;

int arr[10];

int size = sizeof(arr) / sizeof(arr[0]);

while (1) {

cout << "ENTER 1 TO IMPLEMENT QUEUE USING LINKED LIST" << endl;

cout << "ENTER 2 TO IMPLEMENT QUEUE USING ARRAY" << endl;

cin >> option; cout << endl;

switch (option) {

case 1:

{

int opt;

cout << "ENTER 1 TO INSERT IN QUEUE" << endl;

cout << "ENTER 2 TO DISPLAY IN ORDER" << endl;

cout << "ENTER 3 TO DEQUEUE" << endl;

cout << "ENTER 4 TO CHECK IF QUEUE IS EMPTY" << endl;

cout << "ENTER 5 TO DISPLAY IN REVERSE" << endl;

cin >> opt;

switch (opt) {

case 1:

{

system("CLS");

l.read\_data();

break;

}

case 2:

{

system("CLS");

l.display\_original();

break;

}

case 3:

{

system("CLS");

l.dequeue();

break;

}

case 4:

{

system("CLS");

if (l.is\_empty()) {

cout << "THE LIST IS EMPTY" << endl;

}

else {

cout << "THE LIST IS NOT EMPTY" << endl;

}

break;

}

case 5:

{

system("CLS");

l.display\_reverse();

break;

}

}

break;

}

//array

case 2:

{

int opt;

cout << "ENTER 1 TO INSERT IN QUEUE" << endl;

cout << "ENTER 2 TO DISPLAY IN ORDER" << endl;

cout << "ENTER 3 TO DEQUEUE" << endl;

cin >> opt; cout << endl;

switch (opt) {

case 1:

{

system("CLS");

int inp = l\_a.read\_data();

l\_a.insert(arr, inp, size);

break;

}

case 2:

{

system("CLS");

l\_a.display(arr);

break;

}

case 3:

{

system("CLS");

l\_a.dequeue(arr);

break;

}

}

break;

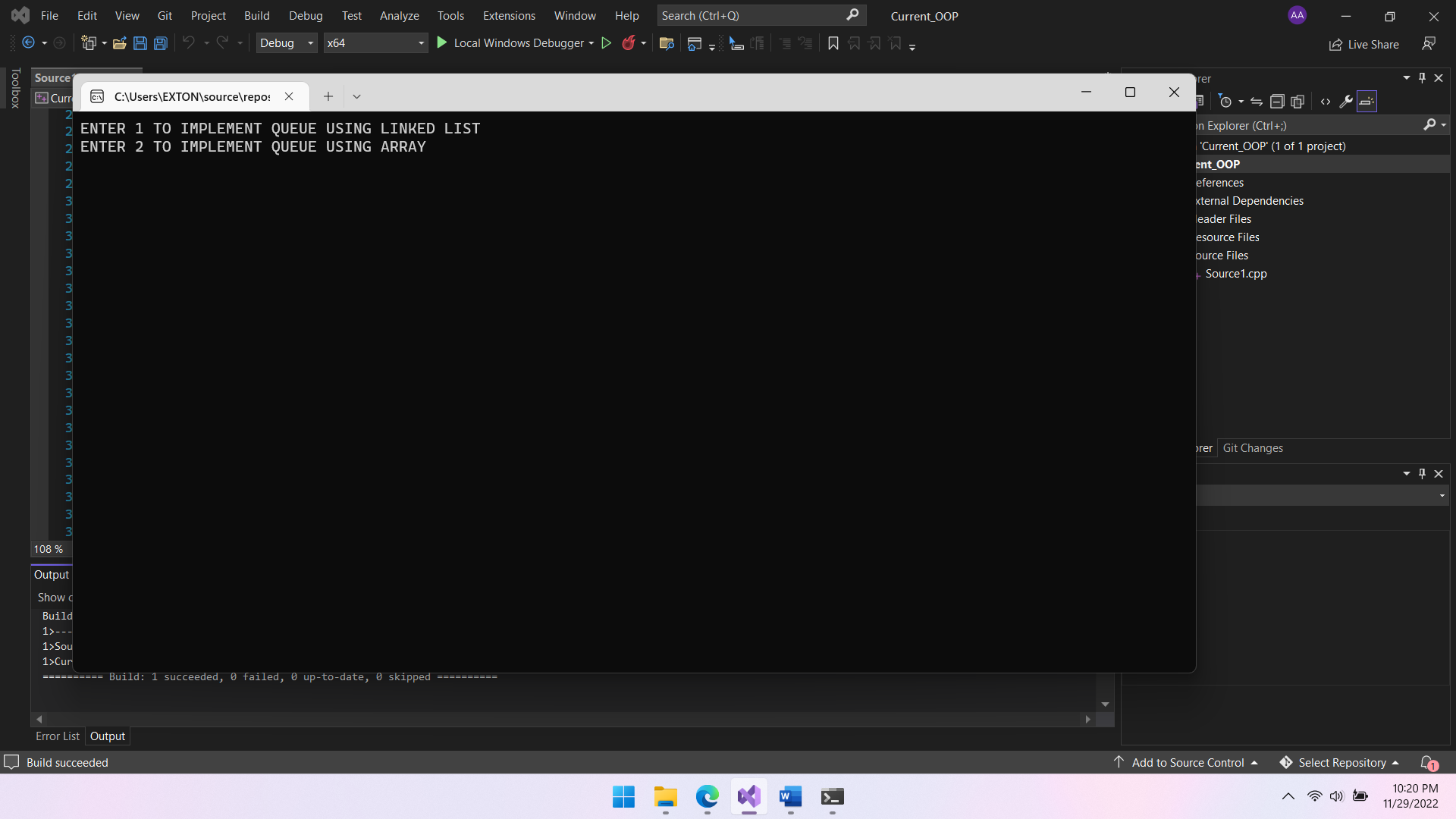
}

}

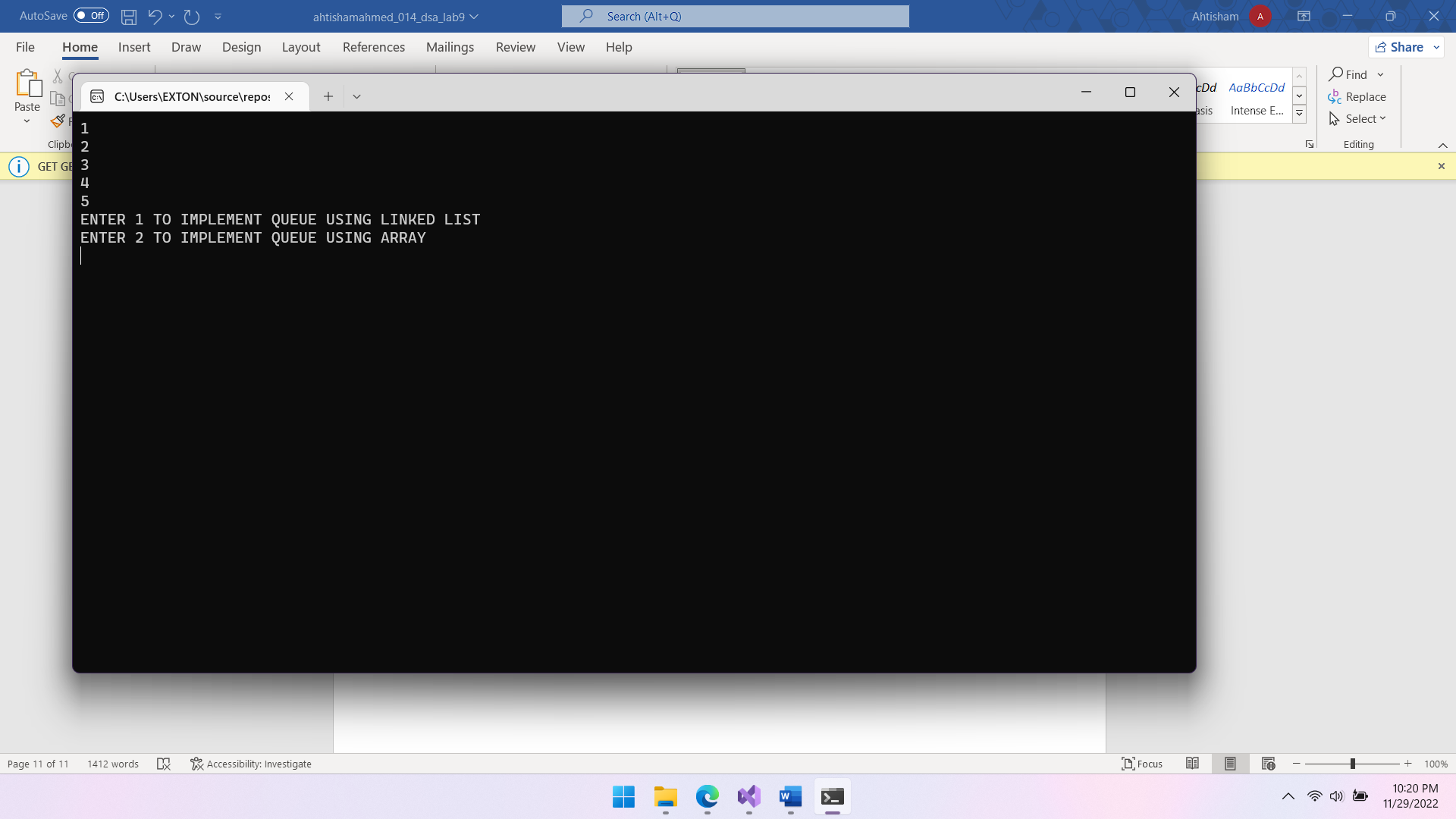
}

}

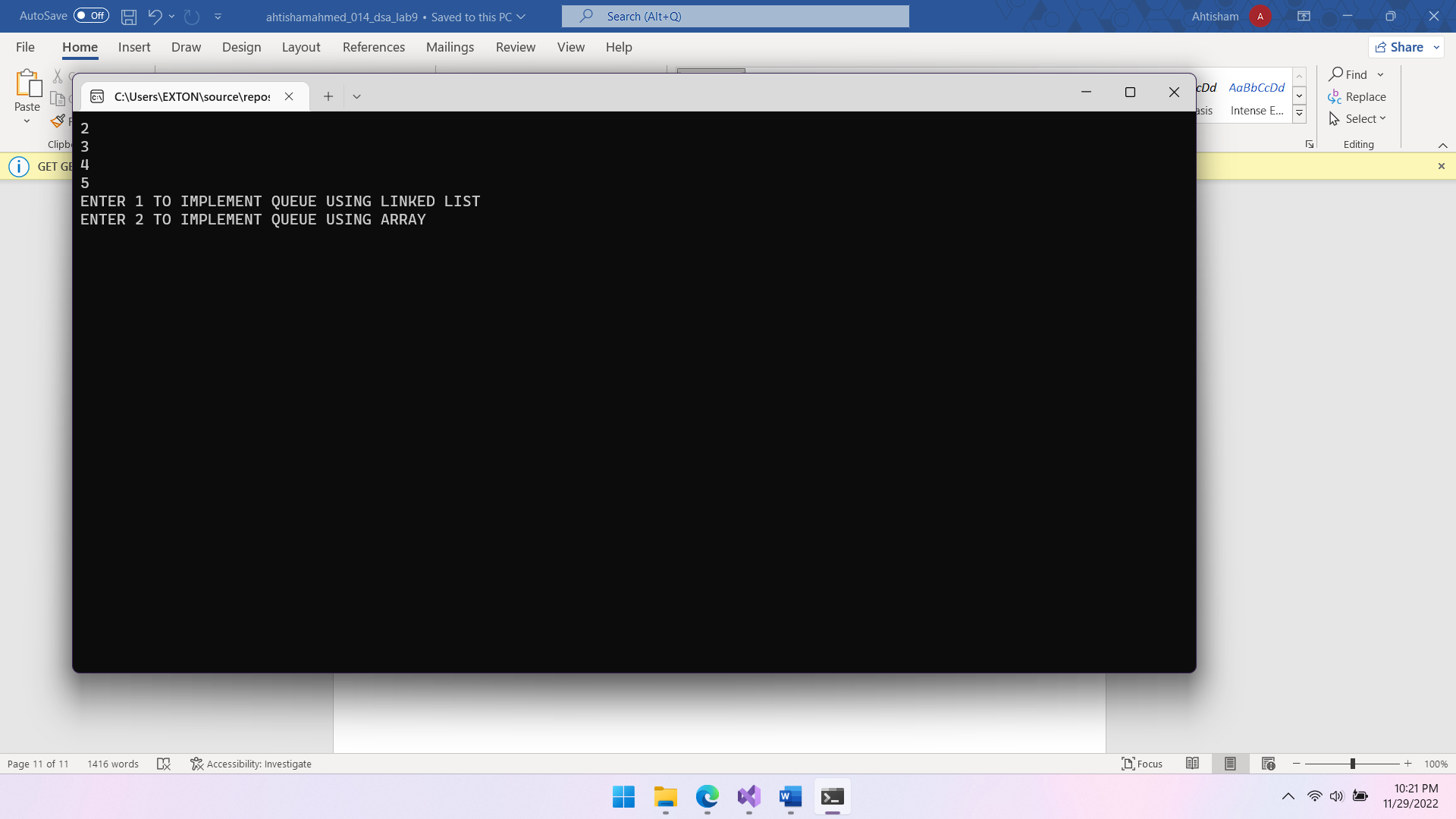
MENU:



QUEUE AFTER INSERTION: [ARRAY]



QUEUE AFTER DEQUEUE: [ARRAY]



\*\*\*\*\*\*\*\*\*\*LINKED LIST IS THE SAME AS PREVIOUS QUESTION\*\*\*\*\*\*\*\*\*\*\*\*\*

Home Task  
A palindrome is a string that reads the same forwards as backwards. Using only a fixed number   
of stacks and queues, write a code to determine if a string is a palindrome. Assume that the string   
is read from standard input one character at a time. The code should output true or false as   
appropriate.

#include <iostream>

using namespace std;

class stack\_arr {

public:

static int front;

static int back;

void read(char\* arr, int size) {

char inp;

cout << "Enter the character : "; cin >> inp; cout << endl;

if (front == -1) {

front++;

back++;

arr[front] = inp;

}

else if (front == back) {

back++;

arr[back] = inp;

}

else if(back<size) {

back++;

arr[back] = inp;

}

else if (back >= size) {

cout << "QUEUE OVERLOADED" << endl;

}

}

bool check(char\* arr, int size) {

int first = 0;

int i = back;

int second = back;

while (first != second) {

if (arr[first] != arr[second]) {

return 0;

}

else {

return 1;

}

first++;

second--;

}

}

void display(char\* arr) {

int i = 0;

while (i != back+1) {

cout << arr[i] << endl;

i++;

}

}

};

int stack\_arr::front = -1;

int stack\_arr::back = -1;

int main() {

char arr[20];

stack\_arr l;

int opt;

int size = sizeof(arr) / sizeof(arr[0]);

while (1) {

cout << "ENTER 1 TO INPUT A CHARACTER" << endl;

cout << "ENTER 2 TO CHECK IF THE DATA IN QUEUE IS A PLAINDROME" << endl;

cout << "ENTER 3 TO DISPLAY" << endl;

cin >> opt; cout << endl;

switch (opt) {

case 1:

{

l.read(arr, size);

cout << l.back << endl;

break;

}

case 2:

{

if (l.check(arr, 20)) {

cout << "PALINDROME" << endl;

}

else {

cout << "NOT A PALINDROME" << endl;

}

break;

}

case 3:

{

l.display(arr);

break;

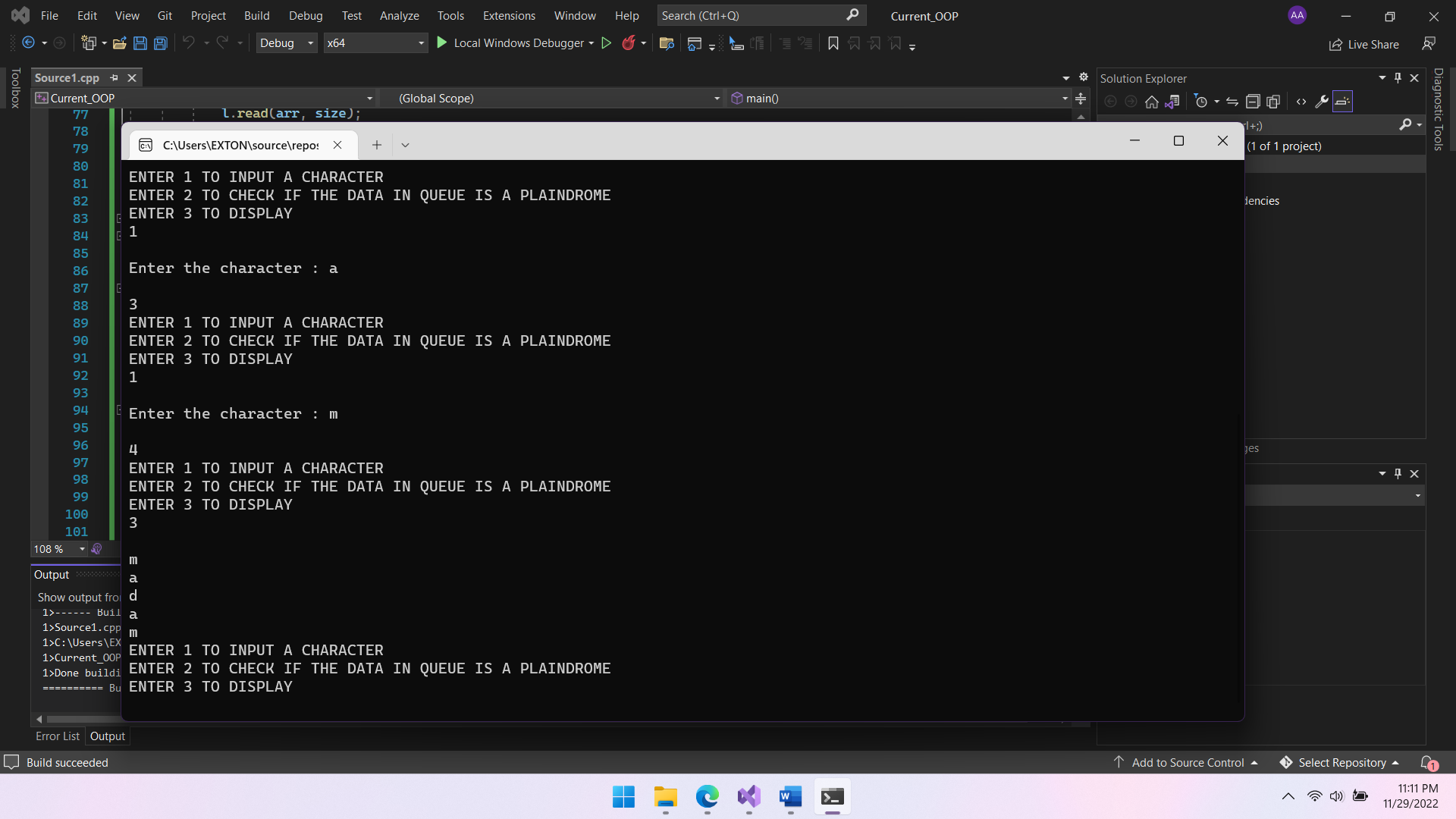
}

}

}

}

INPUT:



OUTPUT:

