**M. Ali. Arslan**

**19F-0348**

**Assignment-02**

**Task # 01**

#include <iostream>

#include <stack>

using namespace std;

bool checkbalance(string input) {

stack<char> s;

char ch;

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

{

if (input[i] == '(' || input[i]=='[' || input[i] == '{')

{

s.push(input[i]);

continue;

}

if (s.empty())

return false;

switch (input[i])

{

case ')':

ch = s.top();

s.pop();

if (ch == '{' || ch == '[')

return false;

break;

case '}':

ch = s.top();

s.pop();

if (ch == '(' || ch == '[')

return false;

break;

case ']':

ch = s.top();

s.pop();

if (ch == '(' || ch == '{')

return false;

break;

}

}

return (s.empty());

}

int main()

{

string str;

cout << "Input: ";

cin >> str;

if (checkbalance(str) == 1)

{

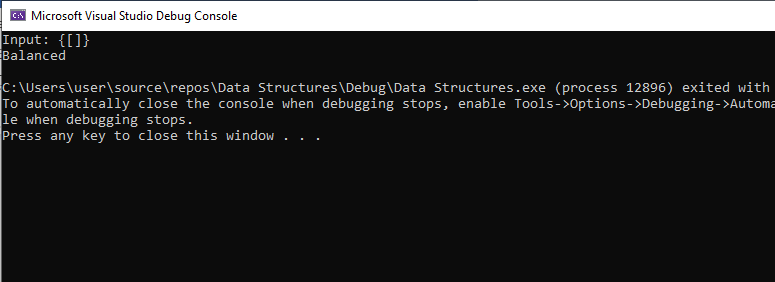
cout << "Balanced" << endl;

}

else {

cout << "Not Balanced" << endl;

}

}

**Task # 02**

#include <iostream>

using namespace std;

struct Node

{

int index;

int data;

Node\* link;

};

Node\* front = NULL;

Node\* rear = NULL;

class Queue

{

private:

int count,a=0,sum=0;

public:

Queue()

: count(0)

{}

bool isEmpty() {

if (front == NULL && rear == NULL)

{

return true;

}

else

{

return false;

}

};

void enqueue(int d)

{

Node\* ptr = new Node;

if (isEmpty())

{

ptr->data = d;

sum = sum + d;

front = rear = ptr;

a = a + 1;

}

else

{

ptr->data = d;

sum = sum + d;

rear->link = ptr;

rear = ptr;

a = a + 1;

}

ptr->index = ++count;

};

int dequeue()

{

int data;

if (isEmpty())

{

cout << "Queue is empty, cannot dequeue!" << endl;

return -1;

}

else if (front == rear)

{

data = front->data;

cout << "Dequeued: " << data << endl;

a--;

sum = sum - data;

front = rear = NULL;

return data;

}

else

{

Node\* ptr = front;

data = front->data;

front = front->link;

ptr = NULL;

cout << "Dequeued: " << data << endl;

a--;

sum = sum - data;

return data;

}

};

void display()

{

if (isEmpty())

{

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

}

else

{

cout << front->data << endl;

}

};

int Return\_Middle()

{

if (isEmpty())

{

cout << "Stack is empty cannot return middle value!" << endl;

return -1;

}

else if (count == 1)

{

cout << "Middle Element: " << front->data << endl;

return front->data;

}

else

{

int midMod = count % 2;

int midIndex = 0;

Node\* midPtr = front;

if (midMod == 0)

{

midIndex = (count / 2);

}

else

{

midIndex = (count / 2) + 1;

}

while (midPtr != NULL)

{

if (midIndex == midPtr->index)

{

cout << "Middle Element: " << midPtr->data << endl;

return midPtr->data;

}

midPtr = midPtr->link;

}

}

}

int sizeofqueue()

{

int size = a;

cout << "\nSize of queue is: " << size << endl;

return size;

}

int sumofqueue()

{

cout << "\nThe Sum of Queue is: " << sum << endl;

return sum;

}

};

int main()

{

Queue q1;

int selection = 0;

int data = 0;

cout << "\t\t\t\t\Queue\n\n";

while (selection != -1)

{

cout << "1) IsEmpty()\n2) Enqueue()\n3) Dequeue()\n4) Display()\n5) Return Middle()\n6) SizeofQueue()\n7) SumofQueue()\n8) Exit()\n\n Enter Option: ";

cin >> selection;

switch (selection)

{

case 1:

if (q1.isEmpty())

{

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

}

else

{

cout << "Queue is not empty!" << endl;

}

break;

case 2:

cout << "Enter the value you want to Enqueue: " << endl;

cin >> data;

q1.enqueue(data);

break;

case 3:

q1.dequeue();

break;

case 4:

q1.display();

break;

case 5:

q1.Return\_Middle();

break;

case 6:

q1.sizeofqueue();

break;

case 7:

q1.sumofqueue();

break;

case 8:

selection = -1;

break;

default:

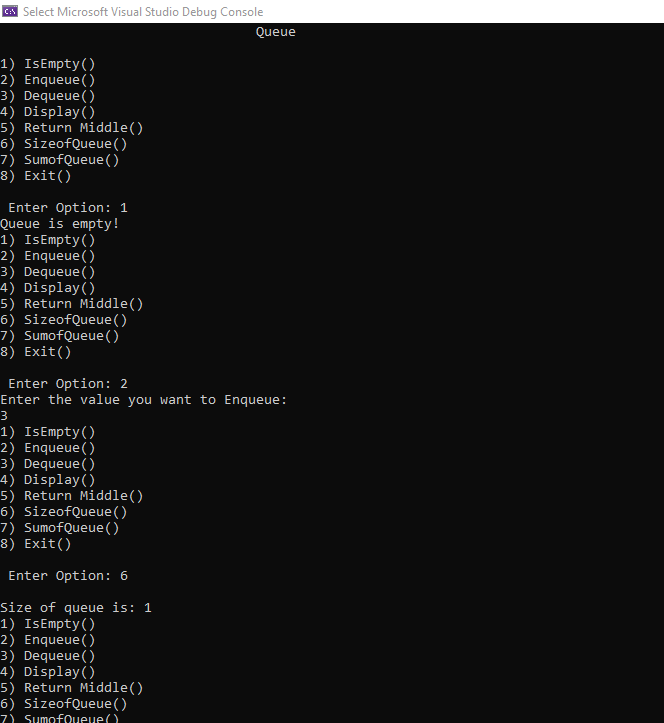
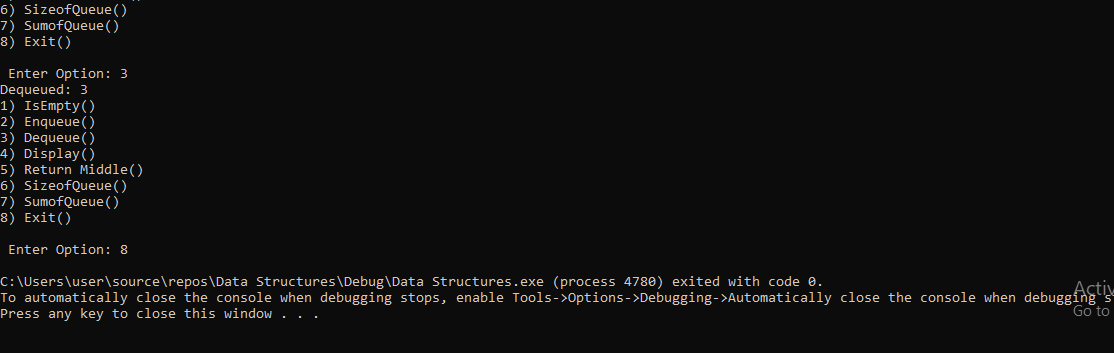
cout << "Invalid input!" << endl;

break;

}

}

}

**Task # 03**

#include<iostream>

#define capacity 6

using namespace std;

int queue[capacity];

int front = -1, rear = -1;

int isFull() {

if ((front == rear + 1) || (front == 0 && rear == capacity - 1)) {

return 1;

}

return 0;

}

int isEmpty() {

if (front == -1)

{

return 1;

}

return 0;

}

void enqueue(int value) {

if (isFull())

cout << "Overflow condition\n" << endl;

else

{

if (front == -1)

front = 0;

rear = (rear + 1) % capacity;

queue[rear] = value;

cout << value << "\tEnqueued in Queue!" << endl;

}

cout << endl;

}

int dequeue() {

int variable;

if (isEmpty()) {

cout << "Queue is Empty" << endl;

return -1;

}

else

{

variable = queue[front];

if (front == rear) {

front = rear = -1;

}

else {

front = (front + 1) % capacity;

}

cout << variable << "\tDequeued from circular queue!" << endl;

return 1;

}

cout << endl;

}

void print() {

int i;

if (isEmpty())

cout << "Nothing to dequeue!" << endl;

else

{

cout << "The queue looks like:" << endl;

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

{

cout << queue[i] << endl;

}

cout << queue[i];

}

cout << endl;

}

int main() {

int selection=0;

while (selection != -1)

{

cout << "1) IsEmpty()\n2) Enqueue()\n3) Dequeue()\n4) Display()\n5) IsFull()\n6) Exit()\n\n Enter Option: ";

cin >> selection;

switch (selection)

{

case 1:

isEmpty();

break;

case 2:

enqueue(15);

enqueue(20);

enqueue(25);

enqueue(30);

enqueue(35);

break;

case 3:

dequeue();

dequeue();

case 4:

print();

break;

case 5:

isFull();

break;

case 6:

selection = -1;

break;

default:

cout << "Invalid input!" << endl;

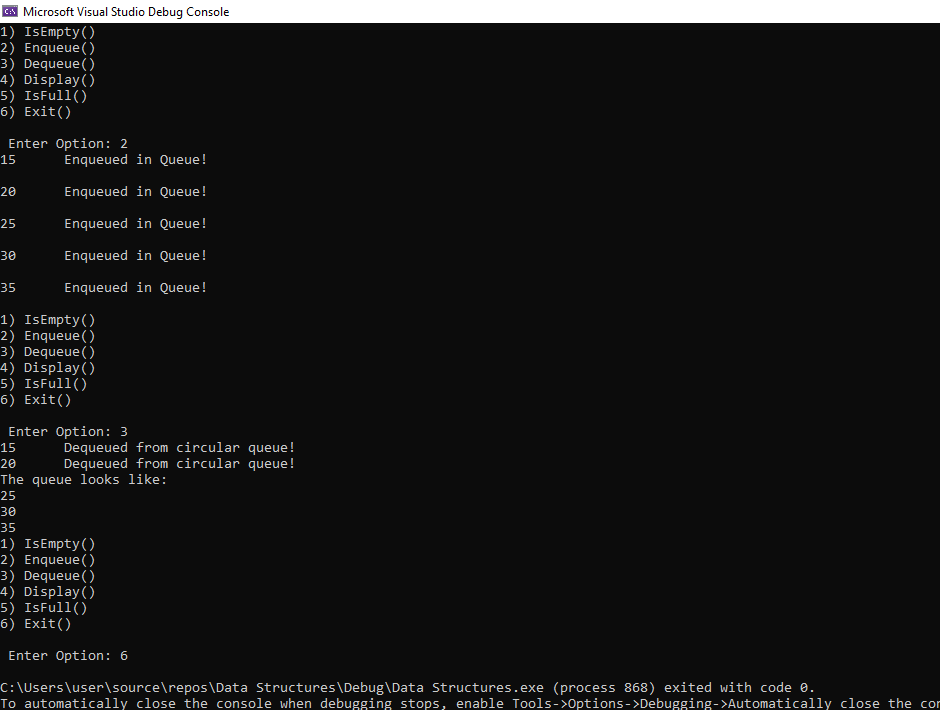
break;

}

}

return 0;

}



**Task # 04**

#include<iostream>

using namespace std;

struct node {

int value;

struct node\* next;

}\*head = NULL;

void insert()

{

int a;

struct node\* prev, \* temp, \* n;

n = (struct node\*)malloc(sizeof(struct node));

cin >> a;

n->value = a;

n->next = NULL;

if (head == NULL)

{

head = n;

}

else

{

if (a < head->value) {

n->next = head;

head = n;

return;

}

prev = temp = head;

while (temp != NULL && temp->value < a) {

prev = temp;

temp = temp->next;

}

n->next = prev->next;

prev->next = n;

}

}

int search() {

int a;

cin >> a;

struct node\* temp;

temp = head;

int i = 1;

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

if (temp->value == a)return i;

i++;

temp = temp->next;

}

return -1;

}

void delet()

{

int a;

a = search() - 1;

if (a == 0) {

head = head->next;

return;

}

struct node\* temp, \* temp1;

temp = head;

for (; a - 1 > 0; a--)

{

temp = temp->next;

}

temp1 = temp->next;

temp->next = temp1->next;

}

void display()

{

struct node\* temp = head;

while (temp != NULL)

{

head;

cout << temp->value;

temp = temp->next;

}

printf("\n");

}

void update() {

delet();

insert();

}

int main()

{

while (1)

{

cout << "Enter your function number \n 1. Insert \n 2. Delete \n 3. Update \n 4. Search \n 5. Exit\n";

int choice;

cout << "Option: ";

cin >> choice;

switch (choice)

{

case 1:

cout << "Enter the value: ";

insert();

break;

case 2:

if (head == NULL) {

cout << "Queue is empty";

}

else {

cout << "Enter the element: ";

delet();

}

break;

case 3:

cout << "Enter the value to be updated & to be updated to: ";

update();

break;

case 4:

cout << "Enter the value to be searched: ";

search();

break;

default:

return 0;

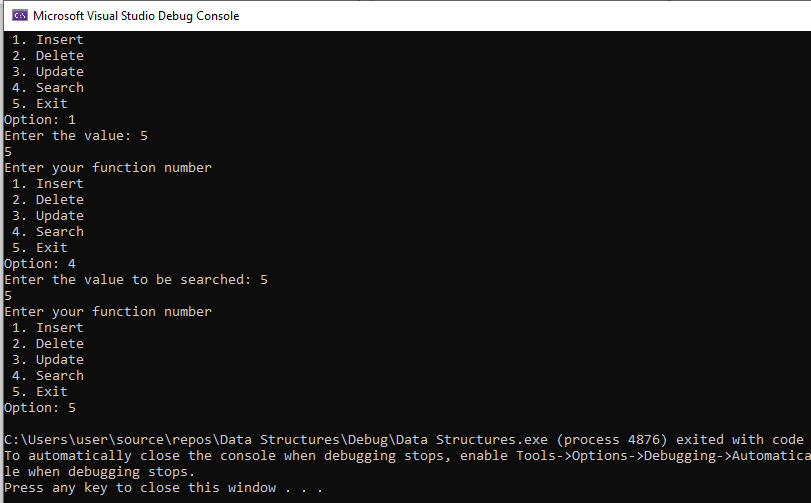
}

display();

}

return 0;

}



**Task # 05**

#include<iostream>

#include<queue>

using namespace std;

int main()

{

int num, temp, pos = 0, time = 0;

queue<int> cOrder;

cin >> num;

/\*const int Size = num;\*/

int\* iOrder = new int[num];

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

{

cin >> temp;

cOrder.push(temp);

}

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

{

cin >> iOrder[i];

}

cout << endl;

while (pos < num)

{

if (iOrder[pos] == cOrder.front())

{

time++;

pos++;

cOrder.pop();

}

else

{

time++;

int pushTolast = cOrder.front();

cOrder.pop();

cOrder.push(pushTolast);

}

}

cout << "Total time taken = " << time << " unit(s)\n" << endl;

return 0;

}

