**Data Structures and Algorithms**

**Lab Journal - Lab 4**

Name: Saad Ahmad

Enrollment #: 01-134222-130

Class/Section: BS-CS-3A

**Objective**

This lab session is aimed at introducing students to singly linked list. In addition, the students will also implement the ‘Queue’ using a linked list and develop a number of utility functions for singly linked lists.

**Task 1:**

Give answers to the following.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1. | Show the contents of a (linear) queue and position of front and rear markers (at each step) once the following sequence of statements is executed.  Queue Q; | | | |
|  | 1. Q.enqueue(10); | 10  Rear 0  Front -1 |  |
| 2. Q.enqueue(20); | 10 20  Rear 1  Front -1 |
| 3. Q.enqueue(30); | 10 20 30  Rear 2  Front -1 |
| 4. Q.dequeue(); | 20 30  Rear 2  Front 0 |
| 5. Q.dequeue(); | 20  Rear 2  Front 1 |
| 6. Q.enqueue(40); | 20 40  Rear 3  Front 1 |
| 7. Q.dequeue() | 40  Rear 3  Front 2 |
| 8. Q.dequeue() | NULL  Rear 3  Front 3 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 2. | Consider a circular QUEUE with N=8 memory cells. Find the number of elements in QUEUE for the following positions of front and rear. | | |  |
|  | **front = 0 ; rear = 4 ;** | 4 |  |
| **front = 2 ; rear = 0 ;** | 5 |  |
| **front = 4 ; rear = 6 ; And two elements are dequeued.** | 0 |  |
|  | | |  |
| 3. | Suppose q is an instance of a circular queue and the queue size is 4. Show the contents of the queue and positions of the front and rear markers once the following sequence of statements is executed. The initial contents of the queue are listed in the following.  q.dequeue(); front **25**   |  | | --- | |  | |  | |  | | 15 |   q.dequeue();  q.enqueue(15); **80**  q.enqueue(25); rear  Front = 2  Rear = 0  25 80 15 | | |  |

**Task 1 :**

Give answers to the following.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1. | The following list of names is assigned (in order) to a linear array INFO. Assign value to LINK and START, so that INFO, LINK and START form an alphabetical list.   |  |  |  | | --- | --- | --- | | START 6 | **INFO** | **LINK** | | 1 | Mary | **&8** | | 2 | Helen | **&7** | | 3 | Barbara | **&5** | | 4 | Paula | **&9** | | 5 | Diana | **&10** | | 6 | Audrey | **&3** | | 7 | Karen | **&1** | | 8 | Nancy | **&4** | | 9 | Ruth | **NULL** | | 10 | Eileen | **&2** | |
| 2. | Given the following linked list, state what does each of the following statements refer to.  2  1  3  first  ptr   |  |  | | --- | --- | | first->data; | 1 | | first->next->next->data; | 3 | | ptr->next->data; | 3 | | ptr->next->next; | NULL | | first->next->next->next; | NULL | |
| 3. | Redraw the following list after the given instructions are executed:  2  1  3  first  ptr  first -> next = first -> next -> next;  first  3  1  2  ptr  ptr -> next -> next = ptr;  first  3  1  2  ptr  ptr->next = NULL;  3  1  2  first  ptr |

**Task 2 :**

Implement the following exercises.

**Exercise 1**

Create a class Linear Queue that implements the functionality of a static linear queue providing all the required operations (Enqueue(), Dequeue(), Empty(), Full() and getFront()).

**Code :**

#include <iostream>

using namespace std;

const int MAXSIZE = 5;

class LinearQueue {

int front;

int back;

int arr[MAXSIZE];

public:

LinearQueue() {

front = -1;

back = -1;

}

void Enqueue(int temp) {

if (Full()) {

cout << "Queue is full" << endl;

}

else {

back++;

arr[back] = temp;

}

if (front == -1) {

front++;

}

}

void Dequeue() {

if (Empty()) {

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

}

else {

front++;

}

if (front > back) {

front = -1;

back = -1;

}

}

bool Empty() {

if (front > back || front == -1) {

return true;

}

else {

return false;

}

}

bool Full() {

if (back == MAXSIZE-1) {

return true;

}

else {

return false;

}

}

int getFront() {

return arr[front];

}

};

int main()

{

LinearQueue q1;

if (q1.Empty()) {

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

}

else {

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

}

q1.Enqueue(5);

q1.Enqueue(4);

q1.Enqueue(3);

q1.Enqueue(2);

q1.Enqueue(1);

q1.Enqueue(0);

q1.Enqueue(8);

cout << q1.getFront() << endl;

q1.Dequeue();

cout << q1.getFront() << endl;

q1.Dequeue();

cout << q1.getFront() << endl;

q1.Dequeue();

cout << q1.getFront() << endl;

q1.Dequeue();

cout << q1.getFront() << endl;

q1.Dequeue();

cout << q1.getFront() << endl;

q1.Dequeue();

cout << q1.getFront() << endl;

q1.Dequeue();

}

**A screenshot of a computer

Description automatically generated**

**Task 3 :**

Implement the following exercises.

**Exercise 1**

|  |
| --- |
| Implement the class Linked List to create a list of integers. You need to provide the implementation of the member functions as described in the following.  **class List**  **{**  **private:**  **Node \* head;**  **public:**  **List();// Constructor**  **~List(); // Destructor**  **bool IsListEmpty();// Checks if the list is empty or not**  **// Inserts a new node with value ‘newV’ after the node containing value ‘oldV’. If a node with value ‘oldV’ does not exist, inserts the new node at the end.**  **void insertafter(int oldV, int newV);**    **// Deletes the node containing the specified value**  **void deleteNode(int value);**  **// Inserts a new node at the start of the list**  **void insert\_begin(int value);**  **// Inserts a new node at the end of the list**  **void insert\_end(int value);**  **// Displays the values stored in the list**  **void traverse();**  **};** |

**Code :**

#include <iostream>

using namespace std;

struct Node {

int data;

Node\* next;

};

class List

{

private:

Node\* head;

public:

List() {

head = NULL;

}

~List() {

delete head;

}

bool IsListEmpty() {

if (head == NULL) {

return true;

}

else {

return false;

}

}

// Inserts a new node with value ‘newV’ after the node containing value ‘oldV’. If a node with value ‘oldV’ does not exist, inserts the new node at the end.

void insertafter(int oldV, int newV) {

Node\* temp = head;

Node\* temp1 = new Node();

temp1->data = newV;

while (temp->data != oldV) {

temp = temp->next;

}

if (temp->data == oldV) {

Node\* temp2 = temp->next;

temp->next = temp1;

temp1->next = temp2;

}

else {

insert\_end(newV);

}

}

// Deletes the node containing the specified value

void deleteNode(int value) {

Node\* temp = head;

Node\* prev = NULL;

while (temp->data != value) {

prev = temp;

temp = temp->next;

}

if (temp->data == value) {

prev->next = temp->next;

delete temp;

}

}

// Inserts a new node at the start of the list

void insert\_begin(int value) {

Node\* temp = new Node();

temp->data = value;

temp->next = head;

head = temp;

}

// Inserts a new node at the end of the list

void insert\_end(int value) {

Node\* temp = new Node();

temp->data = value;

temp->next = NULL;

if (IsListEmpty()) {

head = temp;

}

else {

Node\* check = head;

while (check->next != NULL) {

check = check->next;

}

check->next = temp;

}

}

// Displays the values stored in the list

void traverse() {

Node\* temp = head;

while (temp != NULL) {

cout << temp->data << endl;

temp = temp->next;

}

}

};

int main()

{

List l1;

l1.insert\_begin(5);

l1.insert\_begin(10);

l1.insert\_begin(15);

l1.insert\_begin(20);

l1.insert\_end(50);

l1.insert\_end(90);

cout << "List: " << endl;

l1.traverse();

l1.deleteNode(10);

cout << "After Deletion : " << endl;

l1.traverse();

cout << "Inserting a new node : " << endl;

l1.insertafter(5, 65);

l1.traverse();

}

**Output :**

**A screenshot of a computer

Description automatically generated**

**Exercise 2**

|  |
| --- |
| Linked lists allow efficient implementation of a number of data structures. For instance, Queues can be implemented using a linked list to store the data values. The first node can serve as the ‘front’ while the last node can be regarded as ‘rear’ of the queue (Figure 1). The Enqueue() operation is equivalent to adding a node at the end of the list while the Dequeue() operation removes the first element from the list. Implement the ‘Queue’ class using a linked list for data storage. |

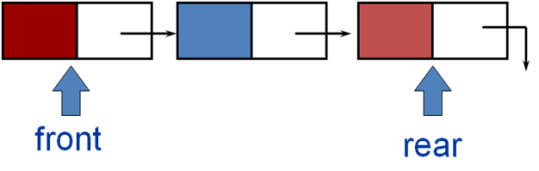
****

Figure 1 Linked list based implementation of queue

#include <iostream>

using namespace std;

struct Node {

int data;

Node\* next;

};

class Queue

{

private:

Node\* front;

Node\* rear;

public:

Queue() {

front = NULL;

rear = NULL;

}

~Queue() {

while (!IsListEmpty())

{

Dequeue();

}

}

bool IsListEmpty() {

if (front == NULL) {

return true;

}

else {

return false;

}

}

void Dequeue() {

if (IsListEmpty()) {

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

}

Node\* temp = front;

front = front->next;

delete temp;

}

void Enqueue(int value) {

Node\* temp = new Node();

temp->data = value;

temp->next = NULL;

if (IsListEmpty()) {

front = temp;

rear = temp;

}

else {

rear->next = temp;

rear = temp;

}

}

// Displays the values stored in the list

void traverse() {

Node\* temp = front;

while (temp != NULL) {

cout << temp->data << endl;

temp = temp->next;

}

}

};

int main()

{

Queue q1;

q1.Enqueue(10);

q1.Enqueue(20);

q1.Enqueue(30);

q1.Enqueue(40);

q1.traverse();

q1.Dequeue();

cout << endl;

q1.traverse();

q1.Dequeue();

cout << endl;

q1.traverse();

q1.Dequeue();

cout << endl;

q1.traverse();

q1.Dequeue();

cout << endl;

q1.traverse();

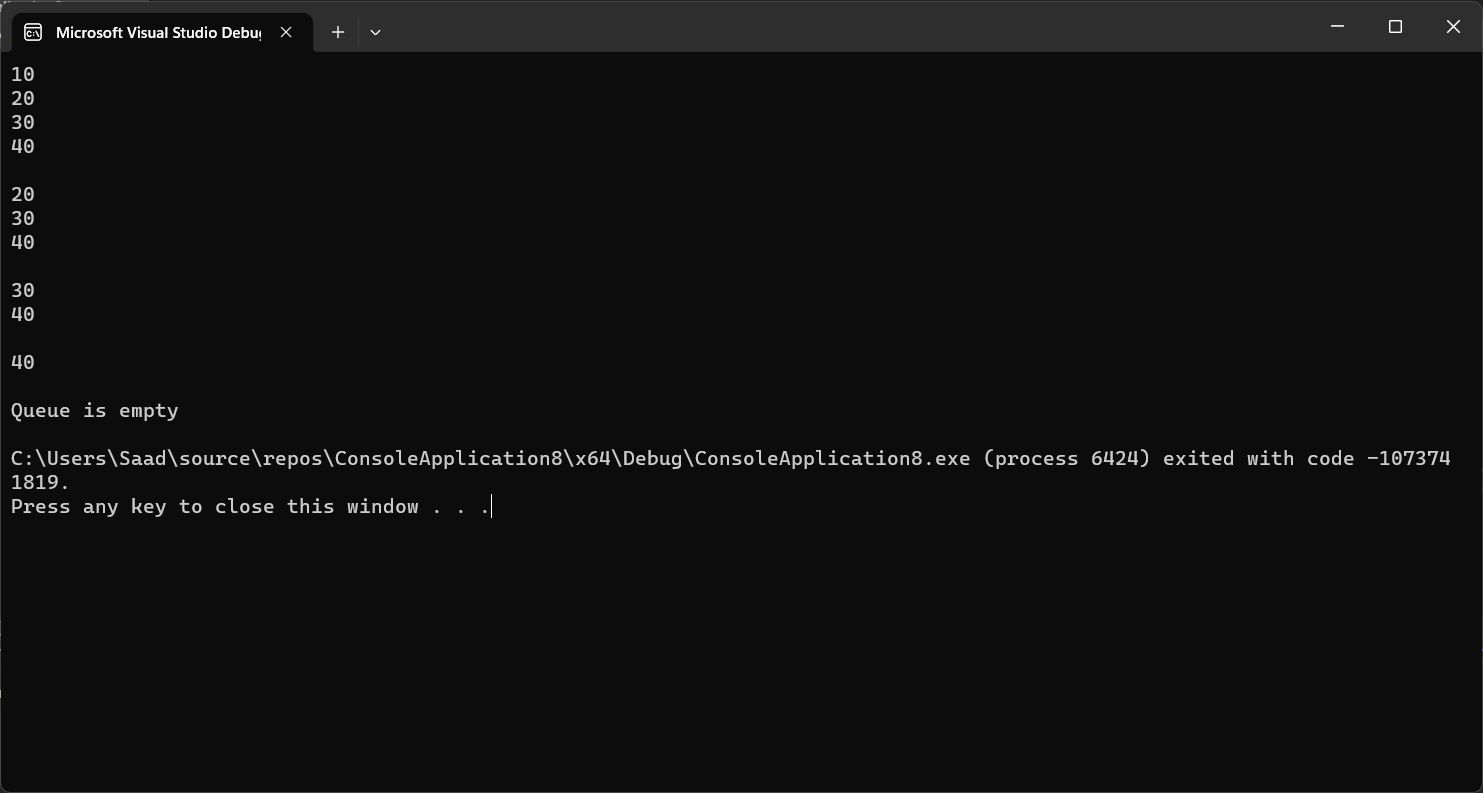
q1.Dequeue();

cout << endl;

q1.traverse();

}

Output:



**Exercise 3**

|  |
| --- |
| Write the following C++ functions to realize the indicated functionality on a singly linked list of integers.   * A function which accepts a pointer to the first node and returns the maximum value in the list. * A function that counts the total number of nodes in the list * A function to search a given value in the list and return the node number where the queried value is found * A function to display all elements of a list * A function swap(Node \*p1, Node \*p2) that swaps the data in the nodes p1 and p2. |

**Code :**

#include <iostream>

using namespace std;

struct Node {

int data;

Node\* next;

};

class List

{

private:

Node\* head;

public:

List() {

head = NULL;

}

~List() {

delete head;

}

bool IsListEmpty() {

if (head == NULL) {

return true;

}

else {

return false;

}

}

void insert\_begin(int value) {

Node\* temp = new Node();

temp->data = value;

temp->next = head;

head = temp;

}

int maxNum(Node\* temp) {

int data =0;

while (temp->next != NULL) {

if (temp->data > data) {

data = temp->data;

}

temp = temp->next;

}

return data;

}

void display() {

Node\* temp = head;

while (temp != NULL) {

cout << temp->data << endl;

temp = temp->next;

}

}

void Total() {

Node\* temp = head;

int counter = 0;

while (temp != NULL) {

counter++;

temp = temp->next;

}

cout << "Total number of nodes in the list is : " << counter << endl;

}

void Search(int value) {

Node\* temp = head;

int counter = 0;

while (temp->data != value) {

counter++;

temp = temp->next;

}

cout << temp->data << " is at " << counter << " index." << endl;

}

Node\* getHead() {

return head;

}

void swap(Node\* p1, Node\* p2) {

int data = p1->data;

p1->data = p2->data;

p2->data = data;

}

};

int main()

{

List l1;

l1.insert\_begin(5);

l1.insert\_begin(10);

l1.insert\_begin(25);

l1.insert\_begin(20);

Node\* temp = l1.getHead();

cout << l1.maxNum(temp) << endl;

l1.Total();

l1.Search(10);

cout << "Before swapping " << endl;

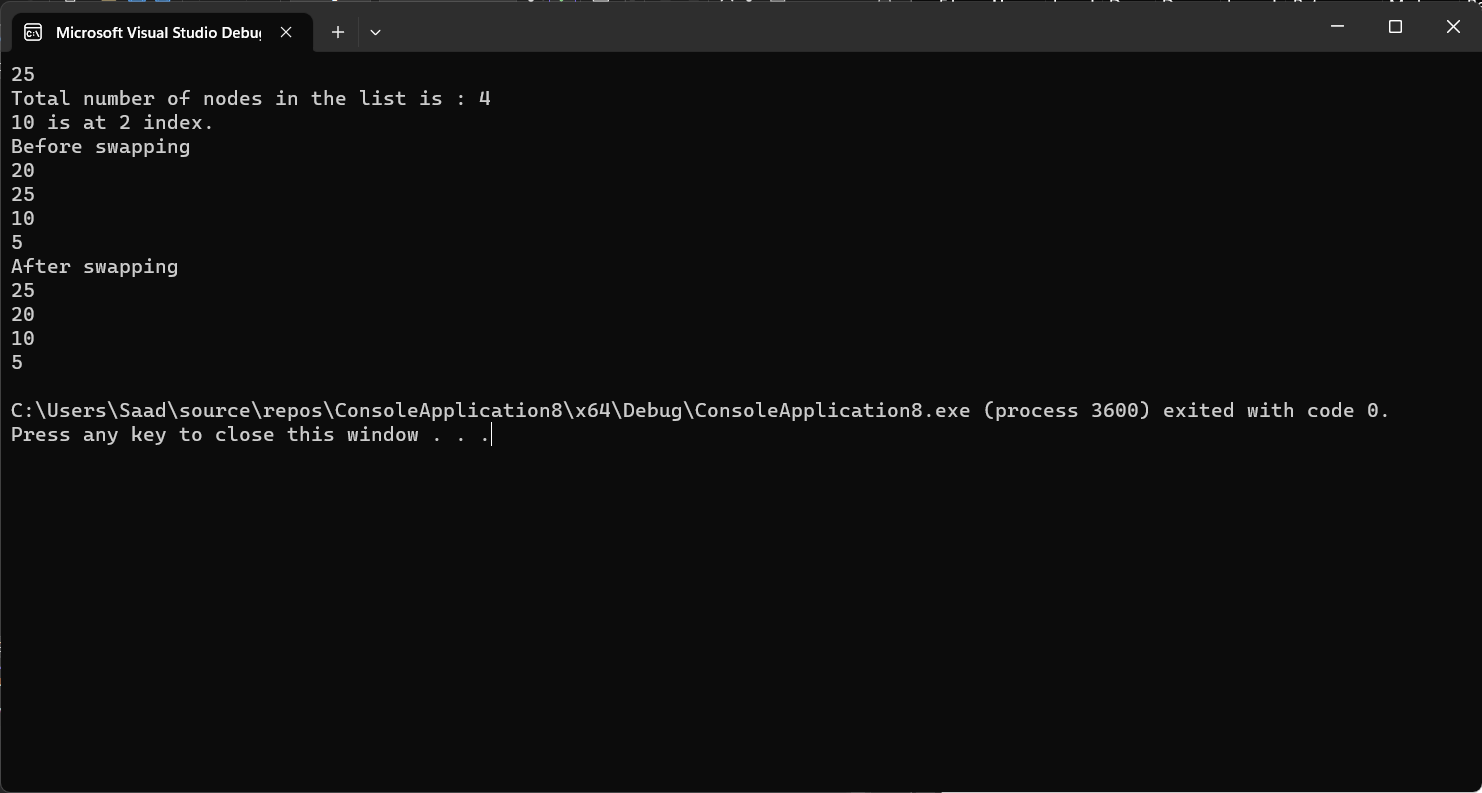
l1.display();

l1.swap(temp, temp->next);

cout << "After swapping" << endl;

l1.display();

}

****

**Implement the given exercises and get them checked by your instructor. If you are unable to complete the tasks in the lab session, deposit this journal alongwith your programs (printed or handwritten) before the start of the next lab session.**

|  |  |  |
| --- | --- | --- |
| **S No.** | **Exercise** | **Checked By:** |
| 1. | Exercise 1 |  |
| 2. | Exercise 2 |  |
| 3. | Exercise 3 |  |

+++++++++++++++++++++++++