**Muhammad Abdullah**

**19F-0916 | SE (3A)**

Data Structures Lab

DS Lab 2

**QUESTION # 1**

**PROGRAM**

#include <iostream>

using namespace std; // BY M.ABDULLAH

// Doubly Link List with almost all functionlities

// Question # 1

struct Node

{

int data;

Node \*NextNode, \*PreviousNode;

};

class Fun

{

public:

Fun()

{

Head = NULL;

}

Node \*Head;

void Add\_Node\_First(int val);

void Add\_Node\_Last(int val);

void Add\_Node\_Random(int val);

void Delete\_Node\_First();

void Delete\_Node\_Last();

void Delete\_Node\_Random();

void Search\_Node\_Data(int val);

void Sort\_List();

void Reverse();

void Count\_List();

void Reverse\_List();

void Display();

};

void Fun::Add\_Node\_First(int val)

{

Node \*temp = NULL, \*current = Head;

if (Head == NULL)

{

temp = new Node;

temp->data = val;

temp->NextNode = NULL;

temp->PreviousNode = NULL;

Head = temp;

}

else

{

temp = new Node;

temp->data = val;

temp->NextNode = current;

current->PreviousNode = temp;

temp->PreviousNode = NULL;

Head = temp;

}

}

void Fun::Add\_Node\_Last(int val)

{

Node \*temp = NULL, \*current = Head;

if (Head != NULL)

{

while (current->NextNode != NULL)

{

current = current->NextNode;

}

temp = new Node;

temp->data = val;

current->NextNode = temp;

temp->PreviousNode = current;

temp->NextNode = NULL;

}

else

cout << endl << "!!! Create Link List First !!!" << endl;

}

void Fun::Add\_Node\_Random(int val)

{

Node \*temp = NULL, \*current = Head;

int opt = 0, counter = 0;

cout << endl << "Enter Postion to add a Node : ";

cin >> opt;

if (Head == NULL)

{

cout << endl << "Create List First !!" << endl;

}

else

{

Node \*temp1 = Head;

while (temp1 != NULL)

{

counter++;

temp1 = temp1->NextNode;

}

if (opt == 1)

{

if (Head == NULL)

{

temp = new Node;

temp->data = val;

temp->NextNode = NULL;

temp->PreviousNode = Head;

Head = temp;

}

else

{

temp = new Node;

temp->data = val;

current = Head;

current->PreviousNode = temp;

temp->NextNode = current;

temp->PreviousNode = Head;

Head = temp;

}

}

else if (opt == counter)

{

while (current->NextNode != NULL)

{

current = current->NextNode;

}

temp = new Node;

temp->data = val;

current->NextNode = temp;

temp->PreviousNode = current;

temp->NextNode = NULL;

}

else if (opt > 1 && opt < counter)

{

current = Head;

for (int i = 2; i < counter - 1; i++)

{

if (i == opt)

{

temp = new Node;

temp->data = val;

temp->NextNode = current->NextNode;

temp->PreviousNode = current;

current->NextNode->PreviousNode = temp;

current->NextNode = temp;

break;

}

current = current->NextNode;

}

}

}

}

void Fun::Delete\_Node\_First()

{

Node \*temp = NULL;

if (Head != NULL)

{

temp = new Node;

temp = Head;

Head = Head->NextNode;

Head->NextNode->PreviousNode = Head;

free(temp);

cout << endl << "First Node has been deleted !" << endl;

}

else

cout << "Link List is Empty ! " << endl;

}

void Fun::Delete\_Node\_Last()

{

Node \*temp = NULL, \*current = Head;

if (Head != NULL)

{

while (current->NextNode->NextNode != NULL)

{

current = current->NextNode;

}

temp = new Node;

temp = current->NextNode->NextNode;

current->NextNode = NULL;

free(temp);

cout << endl << "Last Node has been deleted !" << endl;

}

else

cout << "Link List is Empty ! " << endl;

}

void Fun::Delete\_Node\_Random()

{

Node \*temp = NULL, \*current = Head;

if (Head == NULL)

{

cout << endl << "Link List is Empty !" << endl;

}

else

{

int opt = 0, counter = 0;

cout << "Enter the Position of Node to Delete it : ";

cin >> opt;

while (current != NULL)

{

counter++;

current = current->NextNode;

}

if (opt == 1)

{

temp = new Node;

temp = Head;

Head = Head->NextNode;

Head->NextNode->PreviousNode = Head;

free(temp);

cout << endl << opt << " Node has been deleted !" << endl;

}

else if (opt == counter)

{

current = Head;

while (current->NextNode->NextNode != NULL)

{

current = current->NextNode;

}

temp = new Node;

temp = current->NextNode->NextNode;

current->NextNode = NULL;

free(temp);

cout << endl << opt << " Node has been deleted !" << endl;

}

else if (opt > 1 && opt < counter)

{

current = Head;

for (int i = 2; i < counter - 1; i++)

{

if (i == opt)

{

temp = new Node;

temp = current->NextNode;

current->NextNode = temp->NextNode;

temp->NextNode->PreviousNode = current;

free(temp);

cout << endl << opt << " Node has been deleted !" << endl;

break;

}

current = current->NextNode;

}

}

}

}

void Fun::Search\_Node\_Data(int val)

{

if (Head == NULL)

{

cout << endl << "Link List is Empty !" << endl;

}

else

{

Node \*current = Head;

int counter = 1;

while (current != NULL)

{

if (current->data == val)

{

cout << endl << "Value found at " << counter << " Node" << endl;

}

current = current->NextNode;

counter++;

}

}

}

void Fun::Sort\_List()

{

Node \*current = Head;

int count = 1;

while (count == 1)

{

current = Head;

count = 0;

while (current->NextNode->NextNode != NULL)

{

if (current->data > current->NextNode->data)

{

swap(current->data, current->NextNode->data);

count = 1;

}

current = current->NextNode;

}

}

cout << endl << "Linked List has Been Sorted !!" << endl;

}

void Fun::Reverse()

{

Node \*current = Head, \*temp = NULL;

if (Head == NULL)

{

cout << endl << "Create Link List First !!" << endl;

}

else

{

while (current != NULL)

{

temp = current->PreviousNode;

current->PreviousNode = current->NextNode;

current->NextNode = temp;

current = current->PreviousNode;

}

Head = temp;

cout << endl << "Link List has been reversed !! " << endl;

}

}

void Fun::Count\_List()

{

Node\*temp = Head;

int counter = 0;

if (Head == NULL)

{

cout << endl << "Empty Link List !!" << endl;

}

else

{

while (temp != NULL)

{

counter++;

temp = temp->NextNode;

}

cout << endl << "Total Number of List is/are = " << counter << endl;

}

}

void Fun::Reverse\_List()

{

Node \*temp = Head;

if (Head == NULL)

cout << endl << "Link List is Empty !! " << endl;

else

{

while (temp->NextNode != NULL)

{

temp = temp->NextNode;

}

cout << endl << "DATA : ";

while (temp != Head)

{

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

temp = temp->PreviousNode;

}

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

}

}

void Fun::Display()

{

Node \*temp = Head;

cout << endl << "DATA : ";

while (temp != NULL)

{

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

temp = temp->NextNode;

}

cout << endl;

}

int main()

{

Fun List;

int opt = 1, val = 0;

while (opt != 0)

{

system("cls");

cout << " -----------------------------------" << endl;

cout << " | Press 1 to Add Node on Start |" << endl;

cout << " | Press 2 to Add Node on Last |" << endl;

cout << " | Press 3 to Add Node Randomly |" << endl << " |\t\t\t\t |" << endl;

cout << " | Press 4 to Delete First Node |" << endl;

cout << " | Press 5 to Delete Last Node |" << "\t\t\tDOUBLY LINK LIST" << endl;

cout << " | Press 6 to Delete Random Node |" << endl << " |\t\t\t\t |" << endl;

cout << " | Press 7 to Search Data in Node |" << endl;

cout << " | Press 8 to Sort Link List |" << endl;

cout << " | Press 9 to Reverse the List |" << endl;

cout << " | Press 10 to Count the Total Link|" << endl << " |\t\t\t\t |" << endl;

cout << " | Press 11 to Display Link List |" << endl;

cout << " | Press 12 to Display Reverse |" << endl << " |\t\t\t\t |" << endl;

cout << " | Press 0 to Exit from the system |" << endl;

cout << " -----------------------------------" << endl;

cout << endl << " Option Choosen : ";

cin >> opt;

cout << endl;

switch (opt)

{

case 1:

{

cout << "Enter any data to Enter in List : ";

cin >> val;

List.Add\_Node\_First(val);

break;

}

case 2:

{

cout << "Enter any data to Enter in List : ";

cin >> val;

List.Add\_Node\_Last(val);

break;

}

case 3:

{

cout << "Enter any data to Enter in List : ";

cin >> val;

List.Add\_Node\_Random(val);

break;

}

case 4:

{

List.Delete\_Node\_First();

cout << endl;

system("pause");

break;

}

case 5:

{

List.Delete\_Node\_Last();

cout << endl;

system("pause");

break;

}

case 6:

{

List.Delete\_Node\_Random();

cout << endl;

system("pause");

break;

}

case 7:

{

cout << "Enter value to search in Link List : ";

cin >> val;

List.Search\_Node\_Data(val);

cout << endl;

system("pause");

break;

}

case 8:

{

List.Sort\_List();

cout << endl;

system("pause");

break;

}

case 9:

{

List.Reverse();

cout << endl;

system("pause");

break;

}

case 10:

{

List.Count\_List();

cout << endl;

system("pause");

break;

}

case 11:

{

List.Display();

cout << endl;

system("pause");

break;

}

case 12:

{

List.Reverse\_List();

cout << endl;

system("pause");

break;

}

case 0:

{

opt = 0;

cout << endl << "You have exited from Link List !" << endl;

break;

}

default:

cout << "Invalid Entry, Press any key to try again !!" << endl;

system("pause");

break;

}

}

cout << endl << endl;

system("pause");

}

**A screen shot of a computer

Description automatically generated**

**QUESTION # 2**

**PROGRAM**

#include <iostream>

using namespace std; // BY M.ABDULLAH

// Circular Doublt link list

// Question 2

struct Node

{

int data;

Node \*NextNode, \*PreviousNode;

};

class Fun

{

public:

Fun()

{

Head = NULL, Tail = NULL;

}

Node \*Head, \*Tail;

void Add\_Node\_First(int val);

void Add\_Node\_Last(int val);

void Add\_Node\_Random(int val);

void Delete\_Node\_First();

void Delete\_Node\_Last();

void Delete\_Node\_Random();

void Search\_Node\_Data(int val);

void Display();

};

void Fun::Add\_Node\_First(int val)

{

Node \*temp = NULL, \*current = NULL;

if (Head == NULL)

{

temp = new Node;

temp->data = val;

temp->NextNode = temp;

temp->PreviousNode = temp;

Head = temp;

Tail = temp;

}

else

{

current = Head;

temp = new Node;

temp->data = val;

temp->NextNode = current;

current->PreviousNode = temp;

if (current->NextNode == current)

{

current->NextNode = temp;

temp->PreviousNode = current;

}

else

{

while (current != Tail)

{

current = current->NextNode;

}

temp->PreviousNode = current;

current->NextNode = temp;

}

Head = temp;

}

}

void Fun::Add\_Node\_Last(int val)

{

Node \*temp = NULL, \*current = Head;

if (Head == NULL || Tail == NULL)

{

cout << endl << "Create Link List First !!" << endl;

}

else

{

while (current != Tail)

{

current = current->NextNode;

}

temp = new Node;

temp->data = val;

current->NextNode = temp;

temp->PreviousNode = current;

temp->NextNode = Head;

Tail = temp;

}

}

void Fun::Add\_Node\_Random(int val)

{

Node \*temp = NULL, \*current = Head;

int opt = 0, counter = 0;

cout << endl << "Enter Postion to add a Node : ";

cin >> opt;

if (Head == NULL)

{

cout << endl << "Create List First !!" << endl;

}

else

{

Node \*temp1 = Head;

if (temp1 == Tail)

{

counter++;

}

else

{

while (temp1 != Tail)

{

counter++;

temp1 = temp1->NextNode;

}

counter++;

}

if (opt == 1)

{

if (Head == NULL)

{

temp = new Node;

temp->data = val;

temp->NextNode = temp;

temp->PreviousNode = temp;

Head = temp;

Tail = temp;

}

else

{

current = Head;

temp = new Node;

temp->data = val;

temp->NextNode = current;

current->PreviousNode = temp;

if (current->NextNode == current)

{

current->NextNode = temp;

temp->PreviousNode = current;

}

else

{

while (current != Tail)

{

current = current->NextNode;

}

temp->PreviousNode = current;

current->NextNode = temp;

}

Head = temp;

}

}

else if (opt == counter)

{

while (current != Tail)

{

current = current->NextNode;

}

temp = new Node;

temp->data = val;

current->NextNode = temp;

temp->PreviousNode = current;

temp->NextNode = Head;

Tail = temp;

}

else if (opt > 1 && opt < counter)

{

current = Head;

for (int i = 2; i < counter; i++)

{

if (i == opt)

{

temp = new Node;

temp->data = val;

temp->NextNode = current->NextNode;

temp->PreviousNode = current;

current->NextNode->PreviousNode = temp;

current->NextNode = temp;

break;

}

current = current->NextNode;

}

}

}

}

void Fun::Delete\_Node\_First()

{

Node \*temp = NULL;

if (Head != NULL)

{

temp = new Node;

temp = Head;

if (temp == Tail)

{

free(temp);

Head = NULL, Tail = NULL;

}

else

{

Head = temp->NextNode;

Tail->PreviousNode = temp->NextNode;

free(temp);

}

cout << endl << "First Node has been deleted !" << endl;

}

else

cout << "Link List is Empty ! " << endl;

}

void Fun::Delete\_Node\_Last()

{

Node \*temp = NULL, \*current = Head;

if (Head != NULL)

{

if (current == Tail)

{

temp = new Node;

temp = Head;

free(temp);

Head = NULL, Tail = NULL;

}

else

{

while (current->NextNode != Tail)

{

current = current->NextNode;

}

temp = new Node;

temp = current->NextNode;

Tail = current;

current->NextNode = Head;

free(temp);

cout << endl << "Last Node has been deleted !" << endl;

}

}

else

cout << "Link List is Empty ! " << endl;

}

void Fun::Delete\_Node\_Random()

{

Node \*temp = NULL, \*current = Head;

if (Head == NULL)

{

cout << endl << "Link List is Empty !" << endl;

}

else

{

int opt = 0, counter = 0;

cout << "Enter the Position of Node to Delete it : ";

cin >> opt;

Node \*temp1 = Head;

if (temp1 == Tail)

{

counter++;

}

else

{

while (temp1 != Tail)

{

counter++;

temp1 = temp1->NextNode;

}

counter++;

}

if (opt == 1)

{

temp = new Node;

temp = Head;

if (temp == Tail)

{

free(temp);

Head = NULL, Tail = NULL;

}

else

{

Head = temp->NextNode;

Tail->PreviousNode = temp->NextNode;

free(temp);

}

cout << endl << "First Node has been deleted !" << endl;

}

else if (opt == counter)

{

if (current == Tail)

{

temp = new Node;

temp = Head;

free(temp);

Head = NULL, Tail = NULL;

}

else

{

while (current->NextNode != Tail)

{

current = current->NextNode;

}

temp = new Node;

temp = current->NextNode;

Tail = current;

current->NextNode = Head;

free(temp);

cout << endl << "Last Node has been deleted !" << endl;

}

}

else if (opt > 1 && opt < counter)

{

current = Head;

for (int i = 2; i < counter; i++)

{

if (i == opt)

{

temp = new Node;

temp = current->NextNode;

current->NextNode = temp->NextNode;

temp->NextNode->PreviousNode = current;

free(temp);

cout << endl << opt << " Node has been deleted !" << endl;

break;

}

current = current->NextNode;

}

}

}

}

void Fun::Search\_Node\_Data(int val)

{

if (Head == NULL)

{

cout << endl << "Link List is Empty !" << endl;

}

else

{

Node \*current = Head;

int counter = 1;

if (current == Tail)

{

if (current->data == val)

{

cout << endl << "Value found at " << counter << " Node" << endl;

}

}

else

{

while (current != Tail)

{

if (current->data == val)

{

cout << endl << "Value found at " << counter << " Node" << endl;

}

current = current->NextNode;

counter++;

}

}

}

}

void Fun::Display()

{

Node \*temp = Head;

cout << endl << "DATA : ";

if (Head == NULL)

{

cout << endl << "Empty Link List " << endl;

}

else if (temp == Tail)

{

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

}

else

{

while (temp != Tail)

{

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

temp = temp->NextNode;

}

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

}

cout << endl;

}

int main()

{

Fun List;

int opt = 1, val = 0;

while (opt != 0)

{

system("cls");

cout << " -----------------------------------" << endl;

cout << " | Press 1 to Add Node on Start |" << endl;

cout << " | Press 2 to Add Node on Last |" << endl;

cout << " | Press 3 to Add Node Randomly |" << endl << " |\t\t\t\t |" << endl;

cout << " | Press 4 to Delete First Node |" << endl;

cout << " | Press 5 to Delete Last Node |" << "\t\t\tDOUBLY CIRCULAR LINK LIST" << endl;

cout << " | Press 6 to Delete Random Node |" << endl << " |\t\t\t\t |" << endl;

cout << " | Press 7 to Search Data in Node |" << endl << " |\t\t\t\t |" << endl;

cout << " | Press 9 to Display Link List |" << endl << " |\t\t\t\t |" << endl;

cout << " | Press 0 to Exit from the system |" << endl;

cout << " -----------------------------------" << endl;

cout << endl << " Option Choosen : ";

cin >> opt;

cout << endl;

switch (opt)

{

case 1:

{

cout << "Enter any data to Enter in List : ";

cin >> val;

List.Add\_Node\_First(val);

break;

}

case 2:

{

cout << "Enter any data to Enter in List : ";

cin >> val;

List.Add\_Node\_Last(val);

cout << endl;

system("pause");

break;

}

case 3:

{

cout << "Enter any data to Enter in List : ";

cin >> val;

List.Add\_Node\_Random(val);

cout << endl;

system("pause");

break;

}

case 4:

{

List.Delete\_Node\_First();

cout << endl;

system("pause");

break;

}

case 5:

{

List.Delete\_Node\_Last();

cout << endl;

system("pause");

break;

}

case 6:

{

List.Delete\_Node\_Random();

cout << endl;

system("pause");

break;

}

case 7:

{

cout << "Enter value to search in Link List : ";

cin >> val;

List.Search\_Node\_Data(val);

cout << endl;

system("pause");

break;

}

case 9:

{

List.Display();

cout << endl;

system("pause");

break;

}

case 0:

{

opt = 0;

cout << endl << "You have exited from Link List !" << endl;

break;

}

default:

cout << "Invalid Entry, Press any key to try again !!" << endl;

system("pause");

break;

}

}

cout << endl << endl;

system("pause");

}

**A screenshot of a computer screen

Description automatically generated**

**QUESTION # 3**

**PROGRAM**

#include <iostream>

using namespace std; // BY M.ABDULLAH

struct Node

{

int data;

Node \*NextNode, \*PreviousNode;

};

class Fun

{

public:

Fun()

{

Head = NULL;

}

Node \*Head;

void Add\_Node\_First(int val);

void Add\_Node\_Last(int val);

void Add\_Node\_Random(int val);

void Split\_List();

void Display();

};

void Fun::Add\_Node\_First(int val)

{

Node \*temp = NULL, \*current = Head;

if (Head == NULL)

{

temp = new Node;

temp->data = val;

temp->NextNode = NULL;

temp->PreviousNode = Head;

Head = temp;

}

else

{

temp = new Node;

temp->data = val;

temp->NextNode = current;

current->PreviousNode = temp;

temp->PreviousNode = Head;

Head = temp;

}

}

void Fun::Add\_Node\_Last(int val)

{

Node \*temp = NULL, \*current = Head;

if (Head != NULL)

{

while (current->NextNode != NULL)

{

current = current->NextNode;

}

temp = new Node;

temp->data = val;

current->NextNode = temp;

temp->PreviousNode = current;

temp->NextNode = NULL;

}

else

cout << endl << "!!! Create Link List First !!!" << endl;

}

void Fun::Add\_Node\_Random(int val)

{

Node \*temp = NULL, \*current = Head;

int opt = 0, counter = 0;

cout << endl << "Enter Postion to add a Node : ";

cin >> opt;

if (Head == NULL)

{

cout << endl << "Create List First !!" << endl;

}

else

{

Node \*temp1 = Head;

while (temp1 != NULL)

{

counter++;

temp1 = temp1->NextNode;

}

if (opt == 1)

{

if (Head == NULL)

{

temp = new Node;

temp->data = val;

temp->NextNode = NULL;

temp->PreviousNode = Head;

Head = temp;

}

else

{

temp = new Node;

temp->data = val;

current = Head;

current->PreviousNode = temp;

temp->NextNode = current;

temp->PreviousNode = Head;

Head = temp;

}

}

else if (opt == counter)

{

while (current->NextNode != NULL)

{

current = current->NextNode;

}

temp = new Node;

temp->data = val;

current->NextNode = temp;

temp->PreviousNode = current;

temp->NextNode = NULL;

}

else if (opt > 1 && opt < counter)

{

current = Head;

for (int i = 2; i < counter - 1; i++)

{

if (i == opt)

{

temp = new Node;

temp->data = val;

temp->NextNode = current->NextNode;

temp->PreviousNode = current;

current->NextNode->PreviousNode = temp;

current->NextNode = temp;

break;

}

current = current->NextNode;

}

}

}

}

void Fun::Split\_List()

{

Node \*current = Head;

int counter = 1;

if (Head == NULL)

{

cout << endl << "Create Link List First !!" << endl;

}

else

{

cout << endl << "Spilitted List 1st : ";

while (current != NULL)

{

if (counter % 2 != 0)

{

cout << " " << current->data << " ";

}

current = current->NextNode;

counter++;

}

current = Head;

counter = 1;

cout << endl << "Splitted List 2nd : ";

while (current != NULL)

{

if (counter % 2 == 0)

{

cout << " " << current->data << " ";

}

current = current->NextNode;

counter++;

}

}

}

void Fun::Display()

{

Node \*temp = Head;

cout << endl << "DATA : ";

while (temp != NULL)

{

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

temp = temp->NextNode;

}

cout << endl;

}

int main()

{

Fun List;

int opt = 1, val = 0;

while (opt != 0)

{

system("cls");

cout << " -----------------------------------" << endl;

cout << " | Press 1 to Add Node on Start |" << endl;

cout << " | Press 2 to Add Node on Last |" << endl;

cout << " | Press 3 to Add Node Randomly |" << endl << " |\t\t\t\t |" << endl;

cout << " | Press 4 to Split Link List |" << endl << " |\t\t\t\t |" << endl;

cout << " | Press 5 to Display Link List |" << endl << " |\t\t\t\t |" << endl;

cout << " | Press 0 to Exit from the system |" << endl;

cout << " -----------------------------------" << endl;

cout << endl << " Option Choosen : ";

cin >> opt;

cout << endl;

switch (opt)

{

case 1:

{

cout << "Enter any data to Enter in List : ";

cin >> val;

List.Add\_Node\_First(val);

break;

}

case 2:

{

cout << "Enter any data to Enter in List : ";

cin >> val;

List.Add\_Node\_Last(val);

break;

}

case 3:

{

cout << "Enter any data to Enter in List : ";

cin >> val;

List.Add\_Node\_Random(val);

break;

}

case 4:

{

List.Split\_List();

cout << endl;

system("pause");

break;

}

case 5:

{

List.Display();

cout << endl;

system("pause");

break;

}

case 0:

{

opt = 0;

cout << endl << "You have exited from Link List !" << endl;

break;

}

default:

cout << "Invalid Entry, Press any key to try again !!" << endl;

system("pause");

break;

}

}

cout << endl << endl;

system("pause");

}

**A screenshot of a computer screen

Description automatically generated**

**A screenshot of a computer screen

Description automatically generated**

**QUESTION # 4**

**PROGRAM**

#include <iostream>

using namespace std; // BY M.ABDULLAH

// Equally Divided List

// Question 4

struct Node

{

int data;

Node \*NextNode, \*PreviousNode;

};

class Fun

{

public:

Fun()

{

Head = NULL, Tail = NULL;

}

Node \*Head, \*Tail;

void Add\_Node\_First(int val);

void Add\_Node\_Last(int val);

void Add\_Node\_Random(int val);

void Divide\_List();

void Display();

};

void Fun::Add\_Node\_First(int val)

{

Node \*temp = NULL, \*current = NULL;

if (Head == NULL)

{

temp = new Node;

temp->data = val;

temp->NextNode = temp;

temp->PreviousNode = temp;

Head = temp;

Tail = temp;

}

else

{

current = Head;

temp = new Node;

temp->data = val;

temp->NextNode = current;

current->PreviousNode = temp;

if (current->NextNode == current)

{

current->NextNode = temp;

temp->PreviousNode = current;

}

else

{

while (current != Tail)

{

current = current->NextNode;

}

temp->PreviousNode = current;

current->NextNode = temp;

}

Head = temp;

}

}

void Fun::Add\_Node\_Last(int val)

{

Node \*temp = NULL, \*current = Head;

if (Head == NULL || Tail == NULL)

{

cout << endl << "Create Link List First !!" << endl;

}

else

{

while (current != Tail)

{

current = current->NextNode;

}

temp = new Node;

temp->data = val;

current->NextNode = temp;

temp->PreviousNode = current;

temp->NextNode = Head;

Tail = temp;

}

}

void Fun::Add\_Node\_Random(int val)

{

Node \*temp = NULL, \*current = Head;

int opt = 0, counter = 0;

cout << endl << "Enter Postion to add a Node : ";

cin >> opt;

if (Head == NULL)

{

cout << endl << "Create List First !!" << endl;

}

else

{

Node \*temp1 = Head;

if (temp1 == Tail)

{

counter++;

}

else

{

while (temp1 != Tail)

{

counter++;

temp1 = temp1->NextNode;

}

counter++;

}

if (opt == 1)

{

if (Head == NULL)

{

temp = new Node;

temp->data = val;

temp->NextNode = temp;

temp->PreviousNode = temp;

Head = temp;

Tail = temp;

}

else

{

current = Head;

temp = new Node;

temp->data = val;

temp->NextNode = current;

current->PreviousNode = temp;

if (current->NextNode == current)

{

current->NextNode = temp;

temp->PreviousNode = current;

}

else

{

while (current != Tail)

{

current = current->NextNode;

}

temp->PreviousNode = current;

current->NextNode = temp;

}

Head = temp;

}

}

else if (opt == counter)

{

while (current != Tail)

{

current = current->NextNode;

}

temp = new Node;

temp->data = val;

current->NextNode = temp;

temp->PreviousNode = current;

temp->NextNode = Head;

Tail = temp;

}

else if (opt > 1 && opt < counter)

{

current = Head;

for (int i = 2; i < counter; i++)

{

if (i == opt)

{

temp = new Node;

temp->data = val;

temp->NextNode = current->NextNode;

temp->PreviousNode = current;

current->NextNode->PreviousNode = temp;

current->NextNode = temp;

break;

}

current = current->NextNode;

}

}

}

}

void Fun::Divide\_List()

{

Node \*current = Head;

int counter = 0;

if (Head == NULL)

{

cout << endl << "Link List is Empty !!" << endl;

}

else

{

while (current != Tail)

{

current = current->NextNode;

counter++;

}

counter++;

current = Head;

while (current != Tail)

{

if (counter % 2 == 0)

{

int j = 0;

cout << "First Half List : ";

for (int i = 1; i <= counter / 2; i++)

{

cout << " " << current->data << " ";

current = current->NextNode;

j++;

}

cout << endl << "Second Half List : ";

for (int i = j; i >= counter / 2 && i < counter; i++)

{

cout << " " << current->data << " ";

current = current->NextNode;

}

break;

}

else

{

int j = 0;

cout << "First Half List : ";

for (int i = 1; i <= counter / 2; i++)

{

cout << " " << current->data << " ";

current = current->NextNode;

j++;

}

cout << " " << current->data << " ";

current = current->NextNode;

cout << endl << "Second Half List : ";

for (int i = j; i >= counter / 2 && i < counter-1; i++)

{

cout << " " << current->data << " ";

current = current->NextNode;

}

cout << " " << '0' << " ";

break;

}

}

}

}

void Fun::Display()

{

Node \*temp = Head;

cout << endl << "DATA : ";

if (Head == NULL)

{

cout << endl << "Empty Link List " << endl;

}

else if (temp == Tail)

{

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

}

else

{

while (temp != Tail)

{

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

temp = temp->NextNode;

}

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

}

cout << endl;

}

int main()

{

Fun List;

int opt = 1, val = 0;

while (opt != 0)

{

system("cls");

cout << " -----------------------------------" << endl;

cout << " | Press 1 to Add Node on Start |" << endl;

cout << " | Press 2 to Add Node on Last |" << endl;

cout << " | Press 3 to Add Node Randomly |" << endl << " |\t\t\t\t |" << endl;

cout << " | Press 4 to Divide Link List |" << endl << " |\t\t\t\t |" << endl;

cout << " | Press 9 to Display Link List |" << endl << " |\t\t\t\t |" << endl;

cout << " | Press 0 to Exit from the system |" << endl;

cout << " -----------------------------------" << endl;

cout << endl << " Option Choosen : ";

cin >> opt;

cout << endl;

switch (opt)

{

case 1:

{

cout << "Enter any data to Enter in List : ";

cin >> val;

List.Add\_Node\_First(val);

break;

}

case 2:

{

cout << "Enter any data to Enter in List : ";

cin >> val;

List.Add\_Node\_Last(val);

cout << endl;

system("pause");

break;

}

case 3:

{

cout << "Enter any data to Enter in List : ";

cin >> val;

List.Add\_Node\_Random(val);

cout << endl;

system("pause");

break;

}

case 4:

{

List.Divide\_List();

cout << endl << endl;

system("pause");

break;

}

case 9:

{

List.Display();

cout << endl;

system("pause");

break;

}

case 0:

{

opt = 0;

cout << endl << "You have exited from Link List !" << endl;

break;

}

default:

cout << "Invalid Entry, Press any key to try again !!" << endl;

system("pause");

break;

}

}

cout << endl << endl;

system("pause");

}

**A screen shot of a computer

Description automatically generated**

**A screenshot of a computer screen

Description automatically generated**

**QUESTION # 5**

**PROGRAM**

#include <iostream>

using namespace std; // BY M.ABDULLAH

//Merging X and Y list

// Question 5

struct Node

{

int data;

Node \*NextNode, \*PreviousNode;

};

class Fun

{

public:

Fun()

{

Head = NULL, Tail = NULL;

}

Node \*Head, \*Tail;

void Add\_Node\_First(int val);

void Merge\_List(Fun, Fun);

void Display(int);

};

void Fun::Add\_Node\_First(int val)

{

Node \*temp = NULL, \*current = NULL;

if (Head == NULL)

{

temp = new Node;

temp->data = val;

temp->NextNode = temp;

temp->PreviousNode = temp;

Head = temp;

Tail = temp;

}

else

{

current = Head;

temp = new Node;

temp->data = val;

temp->NextNode = current;

current->PreviousNode = temp;

if (current->NextNode == current)

{

current->NextNode = temp;

temp->PreviousNode = current;

}

else

{

while (current != Tail)

{

current = current->NextNode;

}

temp->PreviousNode = current;

current->NextNode = temp;

}

Head = temp;

}

}

void Fun::Merge\_List(Fun one, Fun two)

{

Node \* THead = two.Head;

Node\*current = one.Head, \*current2 = THead;;

if (Head == NULL)

{

cout << endl << "Link List is Empty !!" << endl;

}

else

{

while (current != one.Tail)

{

current = current->NextNode;

}

current->NextNode = THead;

}

cout << endl << "After Merging the Sorted Linked Lists : ";

current = one.Head;

cout << "(";

while (current != one.Tail)

{

cout << " x" << current->data << ",";

current = current->NextNode;

cout << " y" << current2->data << ",";

current2 = current2->NextNode;

}

cout << " x" << current->data << ",";

cout << " y" << current->data << " )";

}

void Fun::Display(int a)

{

Node \*temp = Head;

cout << endl << "DATA : ";

if (a == 1)

{

if (Head == NULL)

{

cout << endl << "Empty Link List " << endl;

}

else if (temp == Tail)

{

cout << " x" << temp->data << ",";

}

else

{

cout << "(";

while (temp != Tail)

{

cout << " x" << temp->data << ",";

temp = temp->NextNode;

}

cout << " x" << temp->data << " )";

}

}

else

{

if (Head == NULL)

{

cout << endl << "Empty Link List " << endl;

}

else if (temp == Tail)

{

cout << " y" << temp->data << ",";

}

else

{

cout << "(";

while (temp != Tail)

{

cout << " y" << temp->data << ",";

temp = temp->NextNode;

}

cout << " y" << temp->data << " )";

}

}

cout << endl;

}

int main()

{

Fun X, Y;

int opt = 1, val = 0;

while (opt != 0)

{

system("cls");

cout << " -----------------------------------" << endl;

cout << " | Press 1 to Add Node on List 1 |" << endl;

cout << " | Press 2 to Add Node on List 2 |" << endl << " |\t\t\t\t |" << endl;

cout << " | Press 5 to Merge Link List |" << endl << " |\t\t\t\t |" << endl;

cout << " | Press 6 to Display Link List 1 |" << endl;

cout << " | Press 7 to Display Link List 2 |" << endl << " |\t\t\t\t |" << endl;

cout << " | Press 0 to Exit from the system |" << endl;

cout << " -----------------------------------" << endl;

cout << endl << " Option Choosen : ";

cin >> opt;

cout << endl;

switch (opt)

{

case 1:

{

cout << "Enter any data to Enter in List : ";

cin >> val;

X.Add\_Node\_First(val);

break;

}

case 2:

{

cout << "Enter any data to Enter in List : ";

cin >> val;

Y.Add\_Node\_First(val);

break;

}

case 5:

{

X.Merge\_List(X, Y);

cout << endl;

system("pause");

break;

}

case 6:

{

X.Display(1);

cout << endl;

system("pause");

break;

}

case 7:

{

Y.Display(2);

cout << endl;

system("pause");

break;

}

case 0:

{

opt = 0;

cout << endl << "You have exited from Link List !" << endl;

break;

}

default:

cout << "Invalid Entry, Press any key to try again !!" << endl;

system("pause");

break;

}

}

cout << endl << endl;

system("pause");

}

**A screenshot of a computer screen

Description automatically generated**

**A screenshot of a computer screen

Description automatically generated**

**A screenshot of a computer screen

Description automatically generated**

**QUESTION # 7**

**PROGRAM**

#include <iostream>

using namespace std; // BY M.ABDULLAH

//Checking Link list whether it is circular or not

// Question 7

struct Node

{

int data;

Node \*NextNode, \*PreviousNode;

};

class Fun

{

public:

Fun()

{

Head = NULL, Tail = NULL;

}

Node \*Head, \*Tail;

void Add\_Node\_First(int val);

void Sort\_List();

void Merge\_List(Fun,Fun);

void Display();

};

void Fun::Add\_Node\_First(int val)

{

Node \*temp = NULL, \*current = NULL;

if (Head == NULL)

{

temp = new Node;

temp->data = val;

temp->NextNode = temp;

temp->PreviousNode = temp;

Head = temp;

Tail = temp;

}

else

{

current = Head;

temp = new Node;

temp->data = val;

temp->NextNode = current;

current->PreviousNode = temp;

if (current->NextNode == current)

{

current->NextNode = temp;

temp->PreviousNode = current;

}

else

{

while (current != Tail)

{

current = current->NextNode;

}

temp->PreviousNode = current;

current->NextNode = temp;

}

Head = temp;

}

}

void Fun::Sort\_List()

{

Node \*current = Head;

int count = 1;

while (count == 1)

{

current = Head;

count = 0;

while (current != Tail)

{

if (current->data > current->NextNode->data)

{

swap(current->data, current->NextNode->data);

count = 1;

}

current = current->NextNode;

}

}

cout << endl << "Linked List has Been Sorted !!" << endl;

}

void Fun::Merge\_List(Fun one ,Fun two)

{

Node \* THead = two.Head;

Node\*current = one.Head;

if (Head == NULL)

{

cout << endl << "Link List is Empty !!" << endl;

}

else

{

while (current != one.Tail)

{

current = current->NextNode;

}

current->NextNode = THead;

}

cout << endl << "After Merging the Sorted Linked Lists : ";

current = one.Head;

while (current != two.Tail)

{

cout << " " << current->data << " ";

current = current->NextNode;

}

cout << " " << current->data << " ";

}

void Fun::Display()

{

Node \*temp = Head;

cout << endl << "DATA : ";

if (Head == NULL)

{

cout << endl << "Empty Link List " << endl;

}

else if (temp == Tail)

{

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

}

else

{

while (temp != Tail)

{

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

temp = temp->NextNode;

}

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

}

cout << endl;

}

int main()

{

Fun List1,List2;

int opt = 1, val = 0;

while (opt != 0)

{

system("cls");

cout << " -----------------------------------" << endl;

cout << " | Press 1 to Add Node on List 1 |" << endl;

cout << " | Press 2 to Add Node on List 2 |" << endl << " |\t\t\t\t |" << endl;

cout << " | Press 3 to Sort Link List 1 |" << endl;

cout << " | Press 4 to Sort Link List 2 |" << endl;

cout << " | Press 5 to Merge Link List |" << endl << " |\t\t\t\t |" << endl;

cout << " | Press 6 to Display Link List 1 |" << endl << " |\t\t\t\t |" << endl;

cout << " | Press 7 to Display Link List 2 |" << endl << " |\t\t\t\t |" << endl;

cout << " | Press 0 to Exit from the system |" << endl;

cout << " -----------------------------------" << endl;

cout << endl << " Option Choosen : ";

cin >> opt;

cout << endl;

switch (opt)

{

case 1:

{

cout << "Enter any data to Enter in List : ";

cin >> val;

List1.Add\_Node\_First(val);

break;

}

case 2:

{

cout << "Enter any data to Enter in List : ";

cin >> val;

List2.Add\_Node\_First(val);

break;

}

case 3:

{

List1.Sort\_List();

cout << endl;

system("pause");

break;

}

case 4:

{

List2.Sort\_List();

cout << endl;

system("pause");

break;

}

case 5:

{

List1.Merge\_List(List1, List2);

cout << endl;

system("pause");

break;

}

case 6:

{

List1.Display();

cout << endl;

system("pause");

break;

}

case 7:

{

List2.Display();

cout << endl;

system("pause");

break;

}

case 0:

{

opt = 0;

cout << endl << "You have exited from Link List !" << endl;

break;

}

default:

cout << "Invalid Entry, Press any key to try again !!" << endl;

system("pause");

break;

}

}

cout << endl << endl;

system("pause");

}

**LINK LIST 1**

**A screenshot of a computer screen

Description automatically generated**

**LINK LIST 1 SORTED**

**A screenshot of a computer screen

Description automatically generated**

**LINK LIST 2**

**A screenshot of a computer screen

Description automatically generated**

**LINK LIST 2 SORTED**

**A screenshot of a computer screen

Description automatically generated**

**MERGED LISTS**

**Graphical user interface

Description automatically generated**

**QUESTION # 8**

**PROGRAM**

#include <iostream>

using namespace std; // BY M.ABDULLAH

//Checking Link list whether it is circular or not

// Question 8

struct Node

{

int data;

Node \*NextNode, \*PreviousNode;

};

class Fun

{

public:

Fun()

{

Head = NULL, Tail = NULL;

}

Node \*Head, \*Tail;

void Add\_Node\_First(int val);

void Check\_List();

void Display();

};

void Fun::Add\_Node\_First(int val)

{

Node \*temp = NULL, \*current = NULL;

if (Head == NULL)

{

temp = new Node;

temp->data = val;

temp->NextNode = temp;

temp->PreviousNode = temp;

Head = temp;

Tail = temp;

}

else

{

current = Head;

temp = new Node;

temp->data = val;

temp->NextNode = current;

current->PreviousNode = temp;

if (current->NextNode == current)

{

current->NextNode = temp;

temp->PreviousNode = current;

}

else

{

while (current != Tail)

{

current = current->NextNode;

}

temp->PreviousNode = current;

current->NextNode = temp;

}

Head = temp;

}

}

void Fun::Check\_List()

{

Node\*current = Head, \*temp = NULL;

if (Head == NULL)

{

cout << endl << "Create Link List First !!" << endl;

}

else

{

while (current != NULL || current != Tail)

{

if (current->NextNode == NULL)

{

cout << endl << "It is not a Circular Linked List !!" << endl;

break;

}

if (current->NextNode == Tail)

{

cout << endl << "It is a Circular Linked List !!" << endl;

break;

}

current = current->NextNode;

}

}

}

void Fun::Display()

{

Node \*temp = Head;

cout << endl << "DATA : ";

if (Head == NULL)

{

cout << endl << "Empty Link List " << endl;

}

else if (temp == Tail)

{

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

}

else

{

while (temp != Tail)

{

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

temp = temp->NextNode;

}

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

}

cout << endl;

}

int main()

{

Fun List;

int opt = 1, val = 0;

while (opt != 0)

{

system("cls");

cout << " -----------------------------------" << endl;

cout << " | Press 1 to Add Node on Start |" << endl;

cout << " | Press 2 to Check Link List |" << endl << " |\t\t\t\t |" << endl;

cout << " | Press 9 to Display Link List |" << endl << " |\t\t\t\t |" << endl;

cout << " | Press 0 to Exit from the system |" << endl;

cout << " -----------------------------------" << endl;

cout << endl << " Option Choosen : ";

cin >> opt;

cout << endl;

switch (opt)

{

case 1:

{

cout << "Enter any data to Enter in List : ";

cin >> val;

List.Add\_Node\_First(val);

break;

}

case 2:

{

List.Check\_List();

cout << endl;

system("pause");

break;

}

case 9:

{

List.Display();

cout << endl;

system("pause");

break;

}

case 0:

{

opt = 0;

cout << endl << "You have exited from Link List !" << endl;

break;

}

default:

cout << "Invalid Entry, Press any key to try again !!" << endl;

system("pause");

break;

}

}

cout << endl << endl;

system("pause");

}

**A screenshot of a computer screen

Description automatically generated**