# **Name: Abdurrahman Qureshi**

# **Roll No: 210451**

**Practical No: 11**

**Aim: Creating a Doubly Link List and Traversing it**

**CODE:**

#include <stdio.h>

struct node {

int num;

struct node \* preptr;

struct node \* nextptr;

}\*stnode, \*ennode;

void DlListcreation(int n);

void displayDlList();

int main(){

int n;

stnode = NULL;

ennode = NULL;

printf("\n\n Doubly Linked List : Create and display a doubly linked list :\n");

printf("-------------------------------------------------------------------\n");

printf(" Input the number of nodes : ");

scanf("%d", &n);

DlListcreation(n);

displayDlList();

return 0;

}

void DlListcreation(int n){

int i, num;

struct node \*fnNode;

if(n >= 1){

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

if(stnode != NULL){

printf(" Input data for node 1 : "); // assigning data in the first node

scanf("%d", &num);

stnode->num = num;

stnode->preptr = NULL;

stnode->nextptr = NULL;

ennode = stnode;

// putting data for rest of the nodes

for(i=2; i<=n; i++){

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

if(fnNode != NULL){

printf(" Input data for node %d : ", i);

scanf("%d", &num);

fnNode->num = num;

fnNode->preptr = ennode; // new node is linking with the previous node

fnNode->nextptr = NULL;

ennode->nextptr = fnNode; // previous node is linking with the new node

ennode = fnNode; // assign new node as last node

}

else{

printf(" Memory can not be allocated.");

break;

}

}

}

else{

printf(" Memory can not be allocated.");

}

}

}

void displayDlList(){

struct node \* tmp;

int n = 1;

if(stnode == NULL){

printf(" No data found in the List yet.");

}

else{

tmp = stnode;

printf("\n\n Data entered on the list are :\n");

while(tmp != NULL){

printf(" node %d : %d\n", n, tmp->num);

n++;

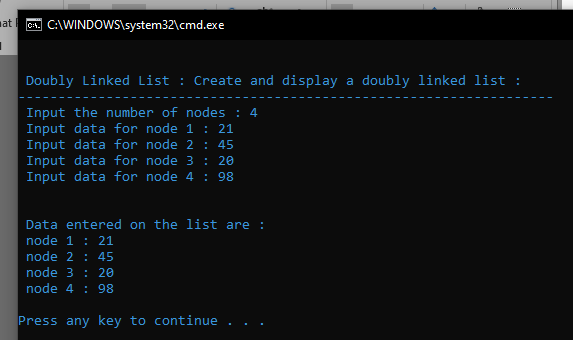
tmp = tmp->nextptr; // current pointer moves to the next node

}

}

}

**OUTPUT:**

****

**Aim: Insertion at start.**

**CODE:**

void DlLinsertNodeAtBeginning(int num)

{

struct node \* newnode;

if(stnode == NULL)

{

printf(" No data found in the list!\n");

}

else

{

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

newnode->num = num;

newnode->nextptr = stnode;

// next address of new node is linking with starting node

newnode->preptr = NULL;

// set previous address field of new node is NULL

stnode->preptr = newnode;

// previous address of starting node is linking with new node

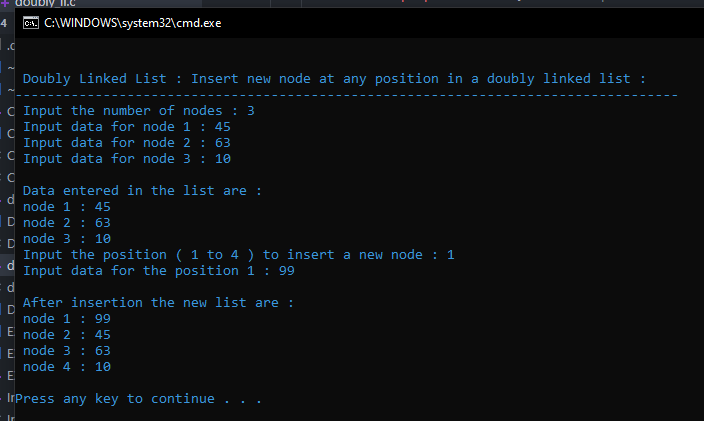
stnode = newnode;

// set the new node as starting node

}

}

**OUTPUT:**



**Aim: Insertion at any index.**

**CODE:**

void DlLinsertNodeAtAny(int num, int pos)

{

int i;

struct node \* newnode, \*tmp;

if(ennode == NULL)

{

printf(" No data found in the list!\n");

}

else

{

tmp = stnode;

i=1;

while(i<pos-1 && tmp!=NULL)

{

tmp = tmp->nextptr;

i++;

}

if(pos == 1)

{

DlLinsertNodeAtBeginning(num);

}

else if(tmp == ennode)

{

DlLinsertNodeAtEnd(num);

}

else if(tmp!=NULL)

{

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

newnode->num = num;

//next address of new node is linking with the next address of temp node

newnode->nextptr = tmp->nextptr;

// previous address of new node is linking with the tmp node

newnode->preptr = tmp;

if(tmp->nextptr != NULL)

{

tmp->nextptr->preptr = newnode;

// n+1th node is linking with new node

}

tmp->nextptr = newnode;

// n-1th node is linking with new node

}

else

{

printf(" The position you entered, is invalid.\n");

}

}

}

**OUTPUT:**



**Aim: Insert at end.**

**CODE:**

void DlLinsertNodeAtEnd(int num){

struct node \* newnode;

if(ennode == NULL){

printf(" No data found in the list!\n");}

Else{

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

newnode->num = num;

newnode->nextptr = NULL; // set next address field of new node is NULL

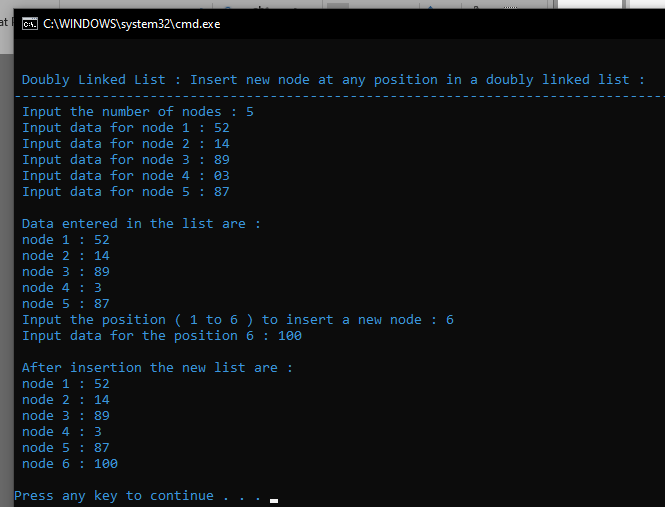
newnode->preptr = ennode; // previous address of new node is linking with ending node

ennode->nextptr = newnode; // next address of ending node is linking with new node

ennode = newnode; // set the new node as ending node

}

}

**OUTPUT:** 

**Aim: Delete first node**

**CODE:**

void DlListDeleteFirstNode(){

struct node \* NodeToDel;

if(stnode == NULL){

printf(" Delete is not possible. No data in the list.\n");

}

else{

NodeToDel = stnode;

stnode = stnode->nextptr; // move the next address of starting node to 2 node

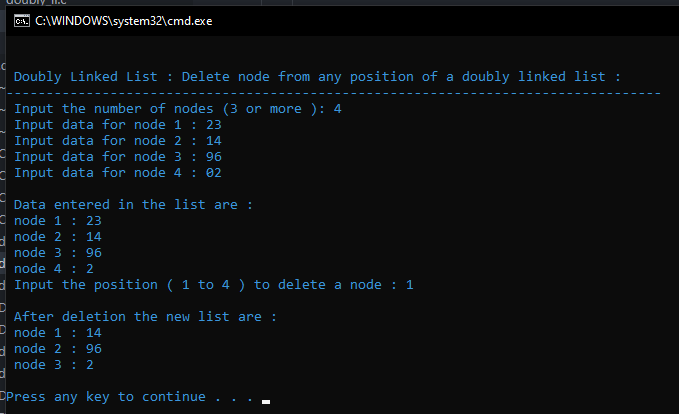
stnode->preptr = NULL; // set previous address of staring node is NULL

free(NodeToDel); // delete the first node from memory

}

}

**OUTPUT:**



**Aim: Delete any node**

**CODE:**

void DlListDeleteAnyNode(int pos){

struct node \*curNode;

int i;

curNode = stnode;

for(i=1; i<pos && curNode!=NULL; i++){

curNode = curNode->nextptr;

}

if(pos == 1){

DlListDeleteFirstNode();

}

else if(curNode == ennode){

DlListDeleteLastNode();

}

else if(curNode != NULL){

curNode->preptr->nextptr = curNode->nextptr;

curNode->nextptr->preptr = curNode->preptr;

free(curNode); //Delete the n node

}

else

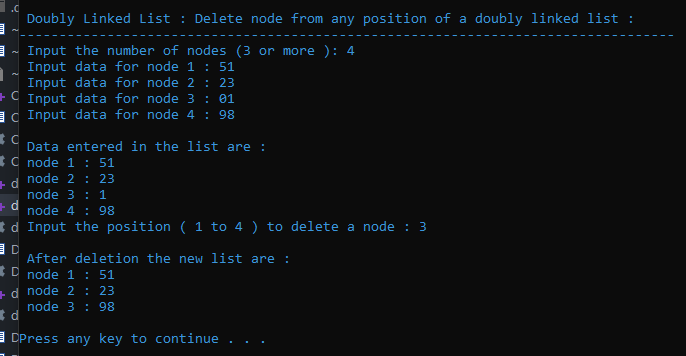
{

printf(" The given position is invalid!\n");

}

}

**OUTPUT:**



**Aim: Delete last node**

**CODE:**

void DlListDeleteLastNode()

{

struct node \* NodeToDel;

if(ennode == NULL)

{

printf(" Delete is not possible. No data in the list.\n");

}

else

{

NodeToDel = ennode;

ennode = ennode->preptr; // move the previous address of the last node to 2nd last node

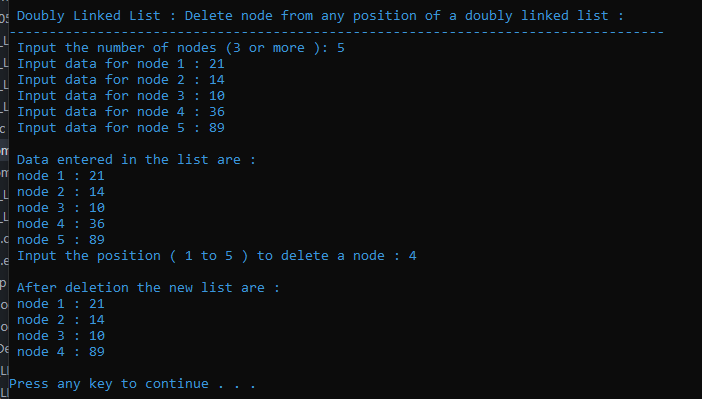
ennode->nextptr = NULL; // set the next address of last node to NULL

free(NodeToDel); // delete the last node

}

}

**OUTPUT:**



**Aim: Find largest and smallest in link list**

**CODE:**

#include<limits.h>

……………..

void printLargestSmallestLinkedList(struct Node\* head) {

int maxElement = INT\_MIN;

int minElement = INT\_MAX;

while (head != NULL) {

if (minElement > head->data)

minElement = head->data;

if (maxElement < head->data)

maxElement = head->data;

head = head->next;

}

cout<<"Smallest element in the linked list is : "<<minElement<<endl;

cout<<"Largest element in the linked list is : "<<maxElement<<endl;

}

**OR**

void findLargest(SingleLL \* ref){

if (ref->head == NULL){

printf("\n Empty linked list");

return;

}

display(ref); // Display linked list

LinkNode \* result = ref->head; // Define some auxiliary variables

LinkNode \* temp = ref->head->next;

while (temp != NULL){ // iterating linked list elements

if (result->data < temp->data){

result = temp; // Get a new big node

} // For smallest (result->data > temp->data){

//result = temp;

//temp = temp->next; }

temp = temp->next; // Visit to next node

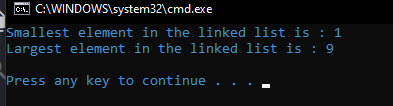
}

printf(" Largest Element : %d\n", result->data);

printf(" Smallest Element : %d\n", result->data);

}

**OUTPUT:**

****