# **Name: Abdurrahman Qureshi**

# **Roll No: 210451**

Practical No: 11

Aim: Creating a Link List and Traversing a Link List

**CODE:**

#include <stdio.h>

#include <stdlib.h>

struct node {

int data;

struct node \*next;

}\*head;

void createList(int n);

void traverseList();

int main()

{

int n;

printf("Enter the total number of nodes: ");

scanf("%d", &n);

createList(n);

printf("\nData in the list \n");

traverseList();

return 0;

}

void createList(int n)

{

struct node \*newNode, \*temp;

int data, i;

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

if(head == NULL)

{

printf("Unable to allocate memory.");

exit(0);

}

printf("Enter the data of node 1: ");

scanf("%d", &data);

head->data = data;

head->next = NULL;

temp = head;

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

{

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

if(newNode == NULL)

{

printf("Unable to allocate memory.");

break;

}

printf("Enter the data of node %d: ", i);

scanf("%d", &data);

newNode->data = data;

newNode->next = NULL;

temp->next = newNode;

temp = temp->next;

}

}

void traverseList()

{

struct node \*temp;

if(head == NULL)

{

printf("List is empty.");

return;

}

temp = head;

while(temp != NULL)

{

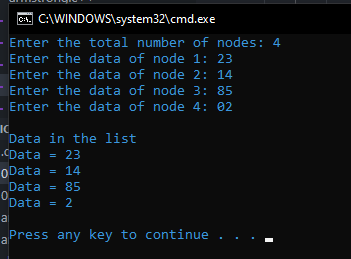
printf("Data = %d\n", temp->data);

temp = temp->next;

}

}

**OUTPUT:**



Aim: Insertion in Link List – First

**CODE:**

void at\_first(struct Node\*\* head\_ref, int new\_data)

{

struct Node\* new\_node = (struct Node\*) malloc(sizeof(struct Node));

new\_node->data = new\_data;

new\_node->next = head;

head = new\_node;

}

Aim: Insertion in Link List – At index

**CODE:**

|  |
| --- |
| void insert\_After(struct Node\* prev\_node, int new\_data){      if (prev\_node == NULL) {          printf("the given previous node cannot be NULL");          return;      }      struct Node\* new\_node = (struct Node\*)malloc(sizeof(struct Node));      new\_node->data = new\_data;      new\_node->next = prev\_node->next;       prev\_node->next = new\_node;  }  Aim: Insertion in Link List – At end  **CODE:**  void append(struct Node\*\* head\_ref, int new\_data){  struct Node\* new\_node = (struct Node\*) malloc(sizeof(struct Node));  struct Node \*last = \*head\_ref;  new\_node->data = new\_data;  new\_node->next = NULL;  if (\*head\_ref == NULL){  \*head\_ref = new\_node;  return;  }  while (last->next != NULL)  last = last->next;  last->next = new\_node;  return;}  **OUTPUT:**    Aim: Deletion in Link List – At first  **CODE:**  struct Node \* deleteFirst(struct Node \* head){  struct Node \* ptr = head;  head = head->next;  free(ptr);  return head;}  Aim: Deletion in Link List – At index  **CODE:**  struct Node \* deleteAtIndex(struct Node \* head, int index){  struct Node \*p = head;  struct Node \*q = head->next;  for (int i = 0; i < index-1; i++){  p = p->next;  q = q->next;}  p->next = q->next;  free(q);  return head;}  Aim: Deletion in Link List – At last  **CODE:** |

struct Node \* deleteAtLast(struct Node \* head){

struct Node \*p = head;

struct Node \*q = head->next;

while(q->next !=NULL){

p = p->next;

q = q->next;

}

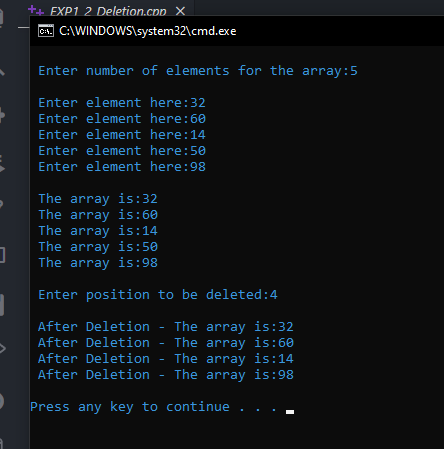
p->next = NULL;

free(q);

return head;

}

**OUTPUT:**

****

Aim: Searching a number in Link List via recursion

**CODE:**

int searchRecursive(int key, struct node \*curNode, int index){

if (curNode == NULL)

return -1;

else if (curNode->data == key)

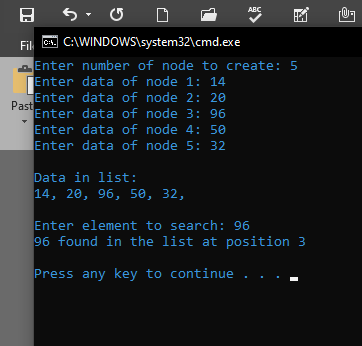
return index;

else

return searchRecursive(key, curNode->next, index + 1);

}

**OUTPUT:**

****