**Batch: A3 Roll No. : 16010121045**

**Experiment No. : 2**

**Grade: AA / AB / BB / BC / CC / CD /DD**

|  |
| --- |
| **Title: Implementation of different operations on Linked List – creation, insertion, deletion, traversal, searching an element** |

**Objective:** To understand the advantage of linked list over other structures like arrays in implementing the general linear list

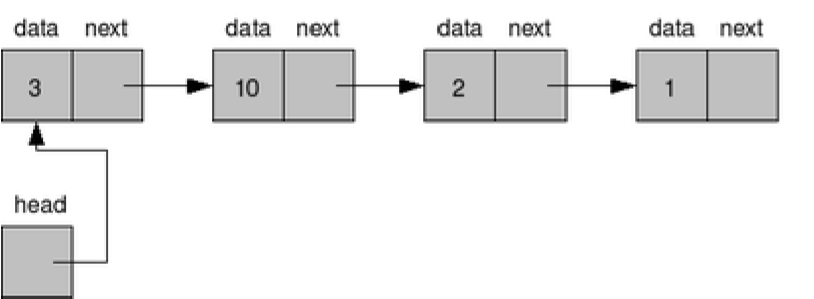
**Expected Outcome of Experiment:**

|  |  |
| --- | --- |
| **CO** | **Outcome** |
| **CO 1** | To understand the advantage of linked list over other structures like arrays in implementing the general linear list |

**Books/ Journals/ Websites referred:**

**Introduction:**

A linear list is a list where each element has a unique successor. There are four common operations associated with linear list: insertion, deletion, retrieval, and traversal. Linear list can be divided into two categories: general list and restricted list. In general list the data can be inserted or deleted without any restriction whereas in restricted list there is restrictions for these operations. Linked list and arrays are commonly used to implement general linear list. A linked list is simply a chain of structures which contain a pointer to the next element. It is dynamic in nature. Items may be added to it or deleted from it at will.



A list item has a pointer to the next element, or to NULL if the current element is the tail (end of the list). This pointer points to a structure of the same type as itself. This Structure that contains elements and pointers to the next structure is called a Node.

**Related Theory: -**

In computer science, a linked list is a linear collection of data elements, whose order is not given by their physical placement in memory. Instead, each element points to the next. It is a data structure consisting of a collection of nodes which together represent a sequence. In its most basic form, each node contains: data, and a reference to the next node in the sequence. This structure allows for efficient insertion or removal of elements from any position in the sequence during iteration.

Like arrays, Linked List is a linear data structure. Unlike arrays, linked list elements are not stored at contiguous location; the elements are linked using pointers

**Linked List ADT:**

Linked List is an *Abstract Data Type (ADT)* that holds a collection of **Nodes,** the nodes  can be accessed in a sequential way. Linked List doesn’t provide a random access to a  Node.

Usually, those Nodes are connected to the next node and/or with the previous one, this gives the linked effect. When the Nodes are connected with only the next pointer the list is called Singly Linked List and when it’s connected by the next and previous the list is called Doubly Linked List.

Value definition:

Abstract Typedef linked\_list/node<<int number, struct linked\_list \*next>>

 Condition: None

Operator Definition

1. Abstract void beginsert<< >>

Pre-condition: none

Post-condition: New node of linked list are linked with it’s next node; and tail as  ptr thus, successively creating circular linked lists. Takes head node as the  argument

2. Abstract void lastinsert<< >>

Pre-condition: none

Post-condition: Node of linked list are linked with it’s tail as ptr thus and tail to  its head

3. Abstract linked\_list/node begin\_delete<< >>

Pre-condition: None

Post-condition: The function then deletes the node aptly at the beginning.

4. Abstract linked\_list/node last\_delete<< >>

Pre-condition: None

Post-condition: The function then deletes the node aptly at the ending.

5. Abstract void display<< >>

Pre-condition: The next node for tail node must point to head

Post-condition: All nodes of the linked list is printed from head to tail;  demonstrating that the linked list previously created is indeed, circualr linked

6. Abstract linked\_list/node searching<<node \*head>>

Pre-condition: None

Post-condition: Requests the user to enter the value of the data stored at a  particular node which is to be searched. The function then searches the linked list  to find the data. Returns the index of the node where the data is stored.

*Algorithm for creation, insertion, deletion, traversal and searching an element in assigned linked list type:*

**Insertion Operation**

beginsert(): Begin

create a new node node

-> data := data if the

list is empty, then head

:= node next of node =

head else temp := head

while next of temp is not head, do

temp := next of temp done next of

node := head next of temp := node

head := node end if

End

**Deletion Operation**

deleteFirst(): Begin

if head is null, then it is

Underflow and return else if

next of head = head, then head

:= null deallocate head else ptr

:= head

while next of ptr is not head, do

ptr := next of ptr next of

ptr = next of head

deallocate head head :=

next of ptr end if

End

**Display List Operation**

display(): Begin

if head is null, then Nothing to

print and return else ptr := head

while next of ptr is not head, do

display data of ptr ptr := next

of ptr display data of ptr end if

End

**Searching:**

Step 1: SET PTR = HEAD

Step 2: Set I = 0

STEP 3: IF PTR = NULL WRITE

"EMPTY LIST"

GOTO STEP 8

END OF IF

STEP 4: REPEAT STEP 5 TO 7 UNTIL PTR != NULL  STEP 5: if ptr → data = item

write i+1 End of IF

STEP 6: I = I + 1

STEP 7: PTR = PTR → NEXT

[END OF LOOP]

STEP 8: EXIT

Implementation of an application using linked lists:

*#include* <stdio.h>

*#include* <stdlib.h>

struct node

{

int value;

struct node \*hash;

};

struct node \*list;

void insertBeg(int *x*)

{

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

ptr->value = x;

ptr->hash = list;

list = ptr;

}

void deleteBeg()

{

*if* (list == NULL)

printf("The list is empty!\n");

*else*

{

struct node \*ptr = list;

list = ptr->hash;

free(ptr);

}

}

void insertEnd(int *x*)

{

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

ptr->value = x;

ptr->hash = NULL;

struct node \*i = list;

*if* (i == NULL)

list = ptr;

*else*

{

*while* (i->hash != NULL)

i = i->hash;

i->hash = ptr;

}

}

void deleteEnd()

{

*if* (list == NULL)

printf("The list is empty!\n");

*else*

{

struct node \*i = list;

struct node \*j = i;

*while* (i->hash != NULL)

{

j = i;

i = i->hash;

}

j->hash = NULL;

free(i);

}

}

void insertAt(int *x*, int *pos*)

{

*if* (list == NULL || pos == 0)

insertBeg(x);

*else*

{

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

ptr->value = x;

ptr->hash = NULL;

struct node \*i = list->hash;

struct node \*j = list;

int counter = 0;

*while* (i != NULL)

{

*if* (++counter == pos)

{

j->hash = ptr;

ptr->hash = i;

*break*;

}

j = i;

i = i->hash;

}

}

}

void deleteAt(int *pos*)

{

*if* (list == NULL)

printf("The list is empty!\n");

*else* *if* (pos == 0)

deleteBeg();

*else*

{

struct node \*i = list->hash;

struct node \*j = list;

int counter = 0;

*while* (i != NULL)

{

*if* (++counter == pos)

{

j->hash = i->hash;

free(i);

*break*;

}

j = i;

i = i->hash;

}

}

}

void printList()

{

*if* (list == NULL)

printf("The list is empty!\n");

*else*

{

struct node \*i = list;

printf("\n\nThe list Elements are :\n");

*while* (i != NULL)

{

printf("%d", i->value);

i = i->hash;

*if* (i != NULL)

printf(" -> ");

}

}

}

void searchList(int *x*)

{

*if* (list == NULL)

printf("The list is empty!\n");

*else*

{

struct node \*i = list;

int pos = 0;

*while* (i != NULL)

{

*if* (i->value == x)

{

printf("Element found at pos: %d\n", pos);

pos = -1;

*break*;

}

i = i->hash;

pos++;

}

*if* (pos != -1)

printf("Element Not Found!\n");

}

}

int main()

{

int t = 0;

*while* (t != 11)

{

printf("\nSelect a function\n");

printf("(1) Print List\n");

printf("(2) Insert at Beginning\n");

printf("(3) Delete Beginning\n");

printf("(4) Insert at End\n");

printf("(5) Delete End\n");

printf("(6) Insert at position\n");

printf("(7) Insert after position\n");

printf("(8) Delete at position\n");

printf("(9) Delete after position\n");

printf("(10) Search Element\n");

printf("(11) Exit\n");

scanf("%d", &t);

*if* (t == 1)

printList();

*else* *if* (t == 2)

{

int x;

printf("Enter element to insert : ");

scanf("%d", &x);

insertBeg(x);

}

*else* *if* (t == 3)

deleteBeg();

*else* *if* (t == 4)

{

int x;

printf("Enter element to insert : ");

scanf("%d", &x);

insertEnd(x);

}

*else* *if* (t == 5)

deleteEnd();

*else* *if* (t == 6)

{

int x, pos;

printf("Enter element to insert : ");

scanf("%d", &x);

printf("Enter element position : ");

scanf("%d", &pos);

insertAt(x, pos);

}

*else* *if* (t == 7)

{

int x, pos;

printf("Enter element to insert : ");

scanf("%d", &x);

printf("Enter position to insert after: ");

scanf("%d", &pos);

insertAt(x, pos + 1);

}

*else* *if* (t == 8)

{

int pos;

printf("Enter position to delete: ");

scanf("%d", &pos);

deleteAt(pos);

}

*else* *if* (t == 9)

{

int pos;

printf("Enter position to delete after: ");

scanf("%d", &pos);

deleteAt(pos + 1);

}

*else* *if* (t == 10)

{

int x;

printf("Enter element to search: ");

scanf("%d", &x);

searchList(x);

}

*else* *if* (t == 11)

*break*;

*else*

printf("Wrong entry!\n");

}

*return* 0;

}

**Conclusion:-**

We have successfully understood algorithm for creation, traversal, insertion, deletion and searching an element in linked list. Written code for application of linked list.

**Post lab questions:**

1. **Write the differences between linked list and linear array**

|  |  |
| --- | --- |
| **Linear Array** | **Linked List** |
| An array is a collection of elements of a  similar data type. | A linked list is a collection of objects  known as a node where node consists of  two parts, i.e., data and address. |
| Array works with a static memory. Here  static memory means that the memory size  is fixed and cannot be changed at the run  time. | The Linked list works with dynamic  memory. Here, dynamic memory means  that the memory size can be changed at  the run time according to our  requirements. |
| Array takes more time while performing  any operation like insertion, deletion, etc. | Linked list takes less time while  performing any operation like insertion,  deletion, etc. |
| In the case of an array, memory is  allocated at compile-time. | In the case of a linked list, memory is  allocated at run time. |
| Memory utilization is inefficient in an  array | Memory utilization is efficient when it  comes to linked lists |

1. **Name some applications which uses linked list.**

The Applications are as follows:

* Implementation of stacks and queues.
* Implementation of graphs: Adjacency list representation of graphs is most popular  which is uses a linked list to store adjacent vertices.
* Dynamic memory allocation: We use a linked list of free blocks.
* Maintaining a directory of names.
* Image viewer – Previous and next images are linked, hence can be accessed by next  and previous button.
* Useful for implementation of a queue. We can maintain a pointer to the last inserted  node and the front can always be obtained as next of last