Batch:B2 Roll No.:16010121110

Experiment No.

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

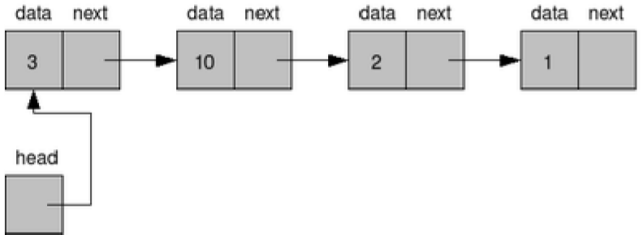
Expected Outcome of Experiment:

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:

abstract typedef linklist{

preconditions: linklist must not be empty

postcondition: makes a dynamic list where insertions and deletions can occur at any node.

}

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

LLType Insert(LLType Head, NodeType NewNode)

// This Algorithm adds a NewNode at the desired position in the linked list. Head is the pointer that points to the first node in the linked list

{ if (head==NULL) // first element in Queue

NewNode->Next = NULL;

head=NewNode;

Else // General case: insertion before head, in the end, in between

}

Algorithm LLType CreateLinkedList()

//This Algorithm creates and returns an empty Linked List, pointed by a pointer -head

{ createNode(head);

head=NULL;

}

3. Algorithm ElementType Delete(LLType Head, ElementType ele)

//This algorithm returns ElementType ele if it exists in the List, an error message otherwise. Temp and current are the a temproary nodes used in the delete process.

{ if (Head==NULL)

Print “Underflow”

exit;

Else

{0. search for element

1. element doesn’t exist in list

2. deletion in unsorted list

3. deletion in sorted list

}

}
Algorithm NodeType Search(LLType Head,
ElementType Key)

//This algorithm returns NodeType node which contains the
‘keyvalue” being searched.

{ if (Head==NULL)

Print “element doesn’t exist”

exit;

Else{
return search(this - > next;
}

Implementation of an application using linked lists:

```
/*********************************************************************
*********
linkedlist in CPP
*********************************************************************
*********/
#include <iostream>

using namespace std;

class node{
   public:
   node* pointer;
   int value;
   node(){

   }
```

```
   public:
   node(node *pointer, int value){
      this->pointer=pointer;
      this->value=value;
   }
   void point(node *pointer){
      this->pointer=pointer;
   }
};
class linkedlist{
      public:

   node *top;
   node * bottom;

   /******
    *
    * Since top and bottom are nodes, edge cases do not come to picture
    * *****/
   linkedlist(){
     top= new node(NULL,-1);
     bottom= new node(NULL,-1);

   }
   node* findbefore(int value){
```

```
    node temp;
    temp.point(bottom);
    while(&temp.pointer!=NULL){
       if(temp.pointer->pointer->value==value){
          return temp.pointer;
       }
       temp.point(temp.pointer->pointer);
    }
    //Do nothing if not found
    cout<<"Sorry node not found";
    return NULL;

}
node* findafter(int value){
    node temp;
    temp.point(bottom->pointer);
    while(&temp!=top){
       if(temp.pointer->value==value){
          return temp.pointer;
       }
       temp.point(temp.pointer->pointer);
    }
    //Do nothing if not found
    cout<<"Sorry node not found";
   return NULL;
```

```
}

void add(int value,int ValueToBeAddedAfter){
   add(new node(NULL,value), ValueToBeAddedAfter);
}
void add(node *toBeAdded, int ValueToBeAddedAfter){
    if(top->pointer!=NULL)
  {top->pointer->point(toBeAdded);} //first addition
  if(bottom->pointer==NULL){
      bottom->point(toBeAdded); //first addition
  }
  else{
 node *NodeToBeAddedAfter = findafter(ValueToBeAddedAfter);
 toBeAdded->point(NodeToBeAddedAfter->pointer);
 NodeToBeAddedAfter->point(toBeAdded);

  }
}
void Delete(int value){

   node *NodeToBeDeleted = findbefore(value);


   NodeToBeDeleted->point(NodeToBeDeleted->pointer->pointer);
```

```
   }
    void display(){
        //cannot display empty list
        node temp;
        temp.point(bottom->pointer);
do{
   cout<<"\n"<<temp.pointer->value;

          temp.point(temp.pointer->pointer);
}
       while(temp.pointer!=NULL);

   }

};



int main()
{

linkedlist *mylink=new linkedlist();
mylink->add(2,1);

mylink->Delete(2);
mylink->add(2,1);
```

```
mylink->add(24,2);


mylink->add(23,2);

mylink->Delete(23);




mylink->display();




cout<<"Hello";


   return 0;

}
```

Conclusion:-

We learned linked list in this experiment. A linked list is a chain of nodes where each node in the list consists of two fields, a data field and a next address field. The data field holds the actual element on the list where the next address field contains the address of next node in the list. Hence, the next address field is simply a reference to the next node.

Post lab questions:

1. Write the differences between linked list and linear array

Size: Since data can only be stored in contiguous blocks of memory in an array, its size cannot be altered at runtime due to the risk of overwriting other data.

However, in a linked list, each node points to the next one such that data can exist at scattered (non-contiguous) addresses; this allows for a dynamic size that can change at runtime.

Memory allocation: For arrays at compile time and at runtime for linked lists. but, a dynamically allocated array also allocates memory at runtime.

Memory efficiency: For the same number of elements, linked lists use more memory as a reference to the next node is also stored along with the data. However, size flexibility in linked lists may make them use less memory overall; this is useful when there is uncertainty about size or there are large variations in the size of data elements;

Memory equivalent to the upper limit on the size has to be allocated (even if not all of it is being used) while using arrays, whereas linked lists can increase their sizes step-by-step proportionately to the amount of data.

Execution time: Any element in an array can be directly accessed with its index. However, in the case of a linked list, all the previous elements must be traversed to reach any element.

Also, better cache locality in arrays (due to contiguous memory allocation) can significantly improve performance. As a result, some operations (such as modifying a certain element) are faster in arrays, while others (such as inserting/deleting an element in the data) are faster in linked lists.

Insertion: In an array, insertion operation takes more time but in a linked list these operations are fast. For example, if we want to insert an element in the array at the end position in the array and the array is full then we copy the array into another array and then we can add an element whereas if the linked list is full then we find the last node and make it next to the new node

Dependency: In an array, values are independent of each other but

In the case of linked list nodes are dependent on each other. one node is dependent on its previous node. If the previous node is lost then we can’t find its next subsequent nodes.

Source - https://www.geeksforgeeks.org/linked-list-vs-array/

2. Name some applications which uses linked list.

Applications of linked list in computer science:

- Implementation of stacks and queues.
- Implementation of graphs: Adjacency list representation of graphs is the most popular which uses a linked list to store adjacent vertices.
- Dynamic memory allocation: We use a linked list of free blocks.
- Maintaining a directory of names.

Source https://www.geeksforgeeks.org/applications-of-linked-list-data-structure/