Batch:C1 Roll No:16010122257

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:

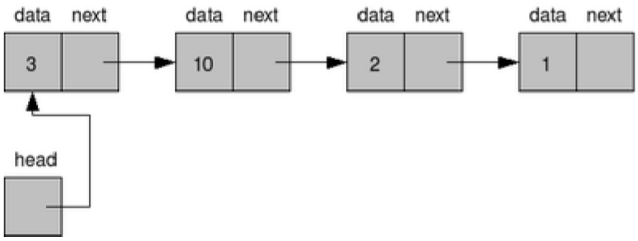
CO	Outcome
CO 1	To understand the advantage of linked list over other structures like arrays in implementing the general linear list

Books/ Journals/ Websites referred:

1)Ma'am's classroom notes
2) https://www.geeksforgeeks.org/data-structures/linked-list/
3) https://dotnettutorials.net/lesson/polynomial-representation-using-linked-list-in-c/

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:

StructNodeType{

ElementType Element;

structNodeType *Next;

}

Algorithm LLTypeCreateLinkedList()

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

{ createNode(head);

head=NULL;

}

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:

APPLICATION:To create two linked lists to read two polynomial equations,then create resultant polynomial after addition, multiplication, division, multiplying one polynomial by a constant number etc.

CODE:

```
#include <iostream>

#include <map>

#include <sstream>


class Poly {

private:

   std::map<int, int> terms;


public:

   void insertTerm(int coeff, int expt) {

      terms[expt] += coeff;

   }


   void display() const {

      bool firstTerm = true;
```

```
for (auto it = terms.rbegin(); it != terms.rend(); ++it) {
    int coeff = it->second;
    int expt = it->first;

    if (coeff != 0) {
        if (firstTerm) {
            std::cout << coeff;
            firstTerm = false;
        } else {
            if (coeff > 0) {
                std::cout << " + " << coeff;
            } else {
                std::cout << " - " << -coeff;
            }
        }

        if (expt != 0) {
            std::cout << "x";
            if (expt != 1) {
                std::cout << "^" << expt;
            }
        }
    }
}
std::cout << std::endl;
```

```
}

Poly add(const Poly& other) const {
    Poly res;
    for (const auto& term : terms) {
        res.insertTerm(term.second, term.first);
    }
    for (const auto& term : other.terms) {
        res.insertTerm(term.second, term.first);
    }
    return res;
}

Poly multiply(const Poly& other) const {
    Poly res;
    for (const auto& term1 : terms) {
        for (const auto& term2 : other.terms) {
            int newCoeff = term1.second * term2.second;
            int newExp = term1.first + term2.first;
            res.insertTerm(newCoeff, newExp);
        }
    }
    return res;
}
```

```
    Poly divide(int divisor) const {
        Poly res;
        for (const auto& term : terms) {
            res.insertTerm(term.second / divisor, term.first);
        }
        return res;
    }

    Poly multiplyByConst(int constnt) const {
        Poly res;
        for (const auto& term : terms) {
            res.insertTerm(term.second * constnt, term.first);
        }
        return res;
    }
};

int main() {
    Poly p1, p2;

    std::cout << "Enter your first polynomial:" << std::endl;
    std::string input;
    std::getline(std::cin, input);
    std::istringstream stream(input);
    int coeff, expt;
```

```
while (stream >> coeff >> expt && !(coeff == 0 && expt == 0)) {
    p1.insertTerm(coeff, expt);
}

std::cout << "Enter your second polynomial:" << std::endl;
std::getline(std::cin, input);
std::istringstream stream2(input);
while (stream2 >> coeff >> expt && !(coeff == 0 && expt == 0)) {
    p2.insertTerm(coeff, expt);
}

std::cout << "The first polynomial you entered is: ";
p1.display();

std::cout << "The second Polynomial you entered is : ";
p2.display();

Poly sum = p1.add(p2);
std::cout << "The sum of entered polynomials is: ";
sum.display();

Poly pdt = p1.multiply(p2);
std::cout << "The product of entered polynomials is: ";
pdt.display();
```

```
    std::cout << "Enter a constant to divide Polynomial 1:" << std::endl;
    int divisor;
    std::cin >> divisor;
    Poly quotient = p1.divide(divisor);
    std::cout << "The quotient of entered polynomials is: ";
    quotient.display();

    std::cout << "Enter a constant to multiply with Polynomial 1:" << std::endl;
    int constant;
    std::cin >> constant;
    Poly scaled = p1.multiplyByConst(constant);
    std::cout << "Polynomial 1 multiplied by " << constant << ": ";
    scaled.display();

    return 0;
}
```

OUTPUT:

```
Output                                                    Clear
/tmp/U3h8MFYCgr.o
Enter your first polynomial:
2 5 4 8 -7 6 8 3 -2 1 6 0
Enter your second polynomial:
5 4 3 2 1 1 -9 0
The first polynomial you entered is: 4x^8 - 7x^6 + 2x^5 + 8x^3 - 2x + 6
The second Polynomial you entered is : 5x^4 + 3x^2 + 1x - 9
The sum of entered polynomials is: 4x^8 - 7x^6 + 2x^5 + 5x^4 + 8x^3 + 3x^2 - 1x - 3
The product of entered polynomials is: 20x^12 - 23x^10 + 14x^9 - 57x^8 + 39x^7 + 65x^6 - 4x^5 +
    38x^4 - 78x^3 + 16x^2 + 24x - 54
Enter a constant to divide Polynomial 1:
4
The quotient of entered polynomials is: 1x^8 - 1x^6 + 2x^3 + 1
Enter a constant to multiply with Polynomial 1:
6
Polynomial 1 multiplied by 6: 24x^8 - 42x^6 + 12x^5 + 48x^3 - 12x + 36
```

Conclusion:-Through this experiment,we thus gained hands-on experience of creating linked lists for mathematical purposes.Successfully implemented a Linked List application to perform operations on polynomials and thus successfully completed the experiment.

Post lab questions:

1. Write the differences between linked list and linear array

ARRAY:

- An array is a grouping of data elements of equivalent data type.
- It stores the data elements in a contiguous memory zone.
- In the case of an array, memory size is fixed, and it is not possible to change it during the run time.
- The memory is assigned at compile time.
- It is easier and faster to access the element in an array.

LINKED LIST:

- A linked list is a group of entities called a node. The node includes two segments: data and address.
- It stores elements randomly, or we can say anywhere in the memory zone.
- In the linked list, the placement of elements is allocated during the run time.
- The memory is assigned at run time.
- In a linked list, the process of accessing elements takes more time.

2. Name some applications which uses linked list.

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.