Laboratory Manual

For

Data Structures and Algorithms

(CT 409)

B.Tech (IT)

SEM IV

June 2010

Faculty of Technology
Dharmsinh Desai University
Nadiad.
www.ddu.ac.in

Table of Contents

Sample experiment

1 AIM: To implement the operations(insert,insert after,insert before ,delete after ,delete before and display the whole link list.

2 TOOLS/APPARATUS: Turbo C/C++

3 STANDARD PROCEDURES:

3.1 Analyzing the Problem:

To insert the set of data, we use the link list data structure, we would like to perform the operations like insert data into link list,insert data after the node whose value is specified, insert data before the node whose value is specified , delete data after the node whose value is specified, delete data before the node whose value is specified and display all the data of link list.

3.2 Designing the Solution:
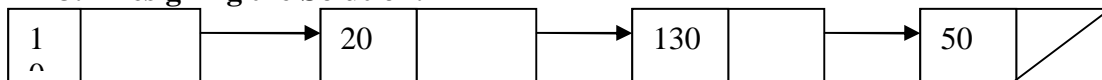
Sturcture of a Link List Node

DATA	LINK
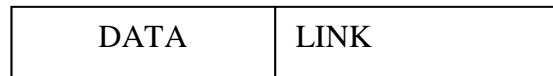
There are following steps for analysis.

1) Generate the class for Node of Link list
2) Create the constructor in this class to initialize the Node of Link list
Create the implementation class for implementing link list operations
Operations:

3.3 Implementing the Solution

3.3.1 Source Code

```
//**************************************
// Operations on Singly Link List//
**************************************
#include<conio.h>
#include<iostream.h>
#include<stdlib.h>
#include<alloc.h>
class LinkListNode
{
public:
int data;
LinkListNode *link;
LinkListNode(int val)
{
data=val;
link=NULL;
}
};
```

Department of Information Technology,Faculty of Technology,D.D.University,Nadiad.

1

```
class  ImpLinkListNode
{
friend class LinkListNode;
public:
LinkListNode *first;        //declaration of head LinkListNode
lkimp()
{
    first=NULL;

}
    void add_LinkListNode(int d)  //constructur for link list
      {

                    if(first==NULL)
                    {
                    first=new LinkListNode(d);
                    }
                    else
                    {
                    LinkListNode *l=new LinkListNode(d);
                            LinkListNode *p=first;
                            while(p->link!=NULL)
                            {
                                    p=p->link;
                             }
                             p->link=l;

                     }

        }
    void delete_LinkListNode(int delval) //for delete the LinkListNode whose value is
    specified
        {
        LinkListNode *p=first;
        LinkListNode *prev;
        while(p!=NULL && p->data!=delval)
        {
        prev=p;
         p=p->link;

        }
        prev->link=p->link;

        }
```

Department of Information Technology,Faculty of Technology,D.D.University,Nadiad.

2

```
void delete_after(int LinkListNodeval) //LinkListNode delete after
{

        LinkListNode *p=first;
         LinkListNode *prev,*next;

        while(p->link!=NULL && p->data!=LinkListNodeval)
        {
        p=p->link;
        }
        prev=p->link;
        next=p->link->link;
        p->link=next;



}


    void delete_before(int LinkListNodeval) //delete a LinkListNode before a
specified LinkListNode
    {

        LinkListNode *p=first;
         LinkListNode *prev,*next;

        while(p->link->link!=NULL && p->link->link->data!=LinkListNodeval)
        {

        prev=p->link;
        p=p->link->link;
        }

        p->link=p->link->link;

 }


    void insert_after(int LinkListNodeval,int newval) //For enter new LinkListNode
after specified value
    {
    LinkListNode *newLinkListNode =new LinkListNode(newval);
    LinkListNode *p=first;
    while(p->link!=NULL && p->data!=LinkListNodeval)
    {
```

Department of Information Technology,Faculty of Technology,D.D.University,Nadiad.

3

```cpp
            p=p->link;
            }
        LinkListNode *link=p;
        newLinkListNode->link=p->link;
        p->link=newLinkListNode;

        }
    void insert_before(int LinkListNodeval,int newval) //For enter new LinkListNode
before specified value
      {
        LinkListNode *newLinkListNode =new LinkListNode(newval);
        LinkListNode *p=first;
        LinkListNode *prev;
        while(p->link!=NULL && p->data!=LinkListNodeval)
        {
        prev=p;
        p=p->link;
        }
        newLinkListNode->link=prev->link;
        prev->link=newLinkListNode;
      }

        void display() //display the linklist
        {
        LinkListNode *p=first;
        while(p!=NULL)
        {
        cout<<"Value ="<<p->data<<"\n";
          p=p->link;

        }
        }
    } ;
    void main()
    {
    clrscr();
    lkimp l;
    l.add_LinkListNode(10);
    l.add_LinkListNode(20);
    l.add_LinkListNode(30);
    l.add_LinkListNode(40);
    l.add_LinkListNode(50);
    l.add_LinkListNode(60);
    l.display();
    cout<<"\n after deletion of 40\n" ;
    l.delete_LinkListNode(40);
```
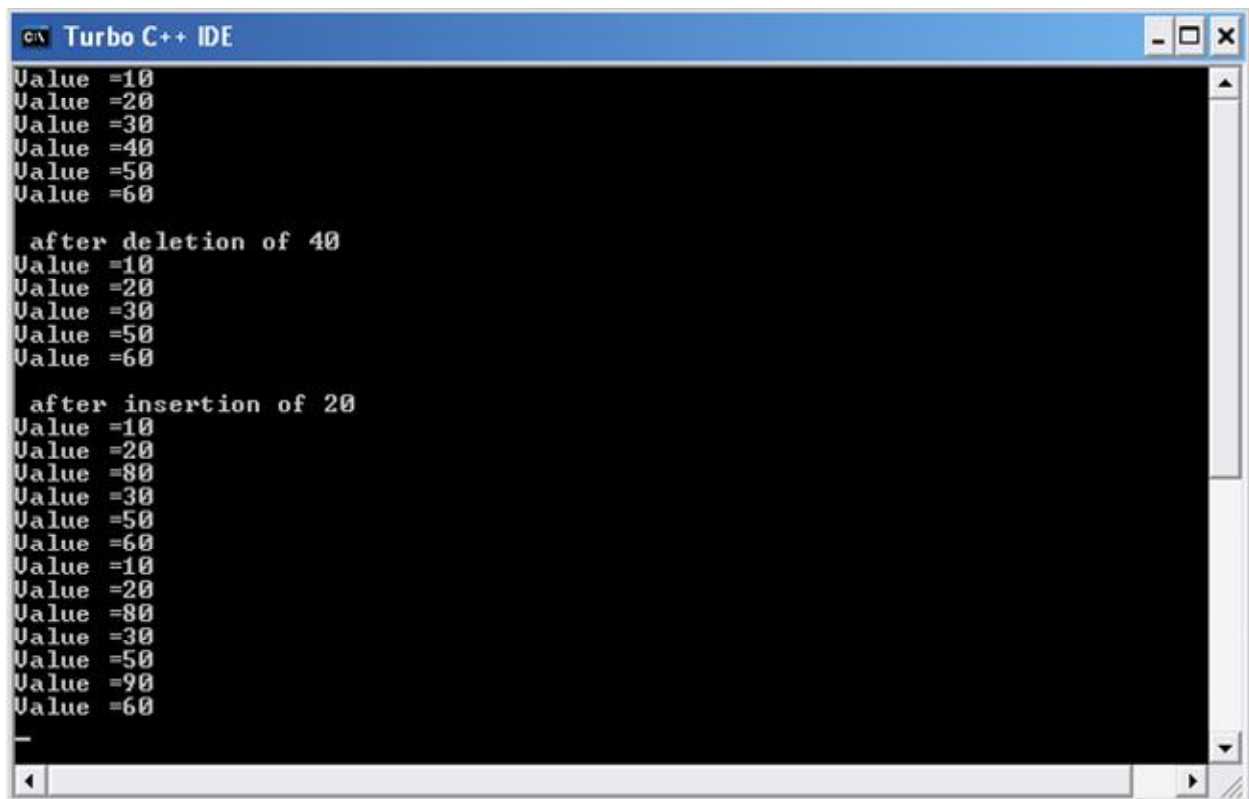
Department of Information Technology,Faculty of Technology,D.D.University,Nadiad.

4

```
        l.display();
        cout<<"\n after insertion of 20\n" ;
        l.insert_after(20,80);
        l.display();

        l.insert_before(40,90);

        l.delete_before(30);

        l.display();
        }
```

3.3.2 Compilation /Running and Debugging the Solution

            - Compile using Alt+F9
            - Run using Ctl+F9
            - View output using Alt+F5

3.4 Testing the Solution

4 Conclusions
By this experiment; we can conclude that basic working of singly link list operations.

Department of Information Technology,Faculty of Technology,D.D.University,Nadiad.

5

EXPERIMENT-1

Aim: Implement the Polynomial representation using a Array.
Tools / Apparatus: O.S.: Microsoft Windows (any) / Linux / DOS
                    Packages: Turbo/Borland/GNU - C/C++
Procedure:

The following steps are required to make the addition of two polynomial..

1) Design the structure of node using C-programming structure.
   struct term
   {
   int coeff;
   int expo;
   struct term *next;
   };

2) Insert term at the end of polynomial using insert end function given below.
   void insertend(TERM **p,int c,int e);
   Pass co-efficient, exponent and term as an argument.

3) Implement polynomial addition function for addition of two polynomial and store in
   third polynomial as a result.
      a. If (T1.exp == T2.exp) than add T1.coeff and T2.coeff and store in T3.coeff .
         And T1.exp++, T2.exp++

      b. If (T1.exp > T2.exp) than T3.exp =T1.exp & T3.coeff=T1.coeff and T1.exp++

      c. Else T3.exp =T1.exp & T3.coeff=T2.coeff and T2.exp++.

   4) Display the result of Term T3 polynomial.

EXPERIMENT-2

Aim: Implement the Application of Stack Infix to Postfix
Tools / Apparatus: O.S.: Microsoft Windows (any) / Linux / DOS
                    Packages: Turbo/Borland/GNU - C/C++
Procedure:

POLISH(Q,P)
   suppose Q is an arithmetic expression written in infix notation,This
   algorithm finds the equivalent postfix expression

   1) Push "(" onto STACK, and add ")" to the end of Q

   2)scan Q from left to right and repeat steps 3 to 6 for each element
   of Q until the stack is empty

   3) if an operand is encountered add it to P

   4) if a left parenthesis is encountered push it onto the stack

   5) if an operator is encountered , then

   (a) Repeatedly pop from STACK and add to P each operator
   (on top of the STACK) which has same precedence as or
   higher precedence than the operator encountered

   (b) Add the encountered operator to the stack
   [end of IF structure]

   6) if a right parenthesis is encountered, then:

   (a) repeatedly pop from the stack and add to P each operator
   (on top of the STACK) until a left parenthesis is encountered

   (b) remove the left parenthesis [do not add left parenthesis to P]

   [End of IF structure]
   [End of step 2 loop]

   7) Exit

EXPERIMENT-3

Aim: Make the basic operations of circular Queue

Tools / Apparatus: O.S.: Microsoft Windows (any) / Linux / DOS

Packages: Turbo/Borland/GNU - C/C++

Procedure:

CQINSERT(CQ,F,R,MAX,NAME) : Given F and R, pointers to the front and rear elements of a circular queue CQ consisting of MAX elements and an element NAME. This procedure inserts X at the rear end of the queue prior to the first invocation of the procedure, F and R have been set to -1. Here 'temp' is a temporary variable within this procedure.

STEP 1 : [Incrementing rear pointer]
Temp <-- R
R <-- (R+1) % MAX

STEP 2 : [Check for overflow condition]
If (R==F)
then R <-- temp
write(overflow)
Return

STEP 3 : [Insert element]
Q[R] <-- NAME
Return

CQDELETE(CQ,F,R) : Given F and R, pointer to the front and the queue CQ to which they correspond. This procedure deletes and returns the element at the front-end of the queue.

STEP 1 : [Check for underflow condition]
if(R==F)
then Write('Underflow')
Return

STEP 2 : [Increment the front pointer]
F <-- (F+1)%MAX

STEP 3 : [Delete element]
Write(Q[F])
Return

Department of Information Technology,Faculty of Technology,D.D.University,Nadiad.

8

EXPERIMENT-4

Aim: Implement the Polynomial representation using a Link -List.
Tools / Apparatus: O.S.: Microsoft Windows (any) / Linux / DOS
                    Packages: Turbo/Borland/GNU - C/C++

Procedure:

1) Implement singly link list with two data members co efficient and exponent.
       struct ele {
       struct ele *next;
       int coef;
       int power;
       };
2) Follow the same process as per polynomial addition using an array.
    a. If (T1.exp == T2.exp) than add T1.coeff and T2.coeff and store in T3.coeff .
        And T1.exp++, T2.exp++

    b. If (T1.exp > T2.exp) than T3.exp =T1.exp & T3.coeff=T1.coeff and T1.exp++

    c. Else T3.exp =T1.exp & T3.coeff=T2.coeff and T2.exp++.

3) Display result of Term T3.

EXPERIMENT-5

Aim: Implement the Doubly Link-List
Tools / Apparatus: O.S.: Microsoft Windows (any) / Linux / DOS
                    Packages: Turbo/Borland/GNU - C/C++
Procedure:

1) Data type declaration as per given below.
       record Node {
         data // The data being stored in the node
         next // A reference to the next node; null for last node
         prev // A reference to the previous node; null for first node
        }
        record List {
       Node firstNode // points to first node of list; null for empty list
       Node lastNode // points to last node of list; null for empty list
        }
  2) Implement Insert before and Insert after function as per given below.
       Function insertAfter(*List* list, *Node* node, *Node* newNode)
       newNode.prev := node
         newNode.next := node.next
         if node.next == null
            list.lastNode := newNode
         else
            node.next.prev := newNode
         node.next := newNode
       function insertBefore(*List* list, *Node* node, *Node* newNode)
         newNode.prev := node.prev
         newNode.next := node
         if node.prev is null
            list.firstNode := newNode
         else
            node.prev.next := newNode
         node.prev   := newNode
  2) Implement the remove function as per given below.
       function remove(*List* list, *Node* node)
         if node.prev == null
            list.firstNode := node.next
         else
            node.prev.next := node.next
         if node.next == null
            list.lastNode := node.prev
         else
            node.next.prev := node.prev
         destroy node

Department of Information Technology,Faculty of Technology,D.D.University,Nadiad.

10

EXPERIMENT-6

Aim: Implement the Binary Tree Traversal
Tools / Apparatus: O.S.: Microsoft Windows (any) / Linux / DOS
Packages: Turbo/Borland/GNU - C/C++

Procedure:

1) Make the structure of Binary Tree Traversal for integer numbers.
```
struct node {
    int data;
    struct node* left;
    struct node* right;
}
```
2) implement inorder(), preorder() and postorder() traversal for it.
   Preorder
   1) Visit the root.
   2) Traverse the left subtree.
   3) Traverse the right subtree.
   Inorder
   1) Traverse the left subtree.
   2) Visit the root.
   3) Traverse the right subtree.
   Postorder
   1) Traverse the left subtree.
   2) Traverse the right subtree.
   3) Visit the root.

EXPERIMENT-7

Aim: Find the Shortest Path using Diskstra's Algo
Tools / Apparatus: O.S.: Microsoft Windows (any) / Linux / DOS
                    Packages: Turbo/Borland/GNU - C/C++
Procedure:

1. Assign to every node a distance value. Set it to zero for our initial node and to infinity for all other nodes.
2. Mark all nodes as unvisited. Set initial node as current.
3. For current node, consider all its unvisited neighbors and calculate their *tentative* distance (from the initial node). For example, if current node (A) has distance of 6, and an edge connecting it with another node (B) is 2, the distance to B through A will be 6+2=8. If this distance is less than the previously recorded distance (infinity in the beginning, zero for the initial node), overwrite the distance.
4. When we are done considering all neighbors of the current node, mark it as visited. A visited node will not be checked ever again; its distance recorded now is final and minimal.
5. If all nodes have been visited, finish. Otherwise, set the unvisited node with the smallest distance (from the initial node) as the next "current node" and continue from step 3.

.

Department of Information Technology,Faculty of Technology,D.D.University,Nadiad.

12

EXPERIMENT-8

Aim: Implement the Shorting using Quick Short method.
Implement the Static Hashing using any one method
Tools / Apparatus: O.S.: Microsoft Windows (any) / Linux / DOS
Packages: Turbo/Borland/GNU - C/C++

Procedure:

The steps are:

1. Pick an element, called a *pivot*, from the list.
2. Reorder the list so that all elements with values less than the pivot come before the pivot, while all elements with values greater than the pivot come after it (equal values can go either way). After this partitioning, the pivot is in its final position. This is called the partition operation.
3. Recursively sort the sub-list of lesser elements and the sub-list of greater elements.

The base case of the recursion are lists of size zero or one, which never need to be sorted.In simple pseudocode, the algorithm might be expressed as this:

```
 function quicksort(array)

  var list less, greater

 if length(array) ≤ 1
    return array
 select and remove a pivot value pivot from array
 for each x in array
    if x ≤ pivot then append x to less
    else append x to greater
 return concatenate(quicksort(less), pivot, quicksort(greater))
```

EXPERIMENT-9

Aim: Implement the Shorting using Merge Short method
Tools / Apparatus: O.S.: Microsoft Windows (any) / Linux / DOS
Packages: Turbo/Borland/GNU - C/C++
Procedure:

Conceptually, a merge sort works as follows

If the list is of length 0 or 1, then it is already sorted. Otherwise:

1) Divide the unsorted list into two sublists of about half the size.
2) Sort each sublist recursively by re-applying merge sort.
3) Merge the two sublists back into one sorted list.

Merge sort incorporates two main ideas to improve its runtime:

1. A small list will take fewer steps to sort than a large list.
2. Fewer steps are required to construct a sorted list from two sorted lists than two unsorted lists. For example, you only have to traverse each list once if they're already sorted (see the merge function below for an example implementation).

Example: Using merge sort to sort a list of integers contained in an array:

Suppose we have an array A with n indices ranging from A_0 to A_{n-1}. We apply merge sort to $A(A_0..A_{c-1})$ and $A(A_c..A_{n-1})$ where c is the integer part of $n / 2$. When the two halves are returned they will have been sorted. They can now be merged together to form a sorted array.

In a simple pseudocode form, the algorithm could look something like this:

```
function merge_sort(m)
    if length(m) ≤ 1
        return m
    var list left, right, result

    var integer middle = length(m) / 2
    for each x in m up to middle
        add x to left
    for each x in m after middle
        add x to right
    left = merge_sort(left)
    right = merge_sort(right)
    result = merge(left, right)
    return result
```

EXPERIMENT-10

Aim: Implement the Static Hashing using any one method.
Tools / Apparatus: O.S.: Microsoft Windows (any) / Linux / DOS
 Packages: Turbo/Borland/GNU - C/C++
Procedure:

Steps-:

1) Store the string of data in array.
2) Take static Hash array of 26 character size.
3) Make the modulo hash function and store data at respective hash location in static hash array.

 Hash location = int(character) mode 26
4) If the location is already filled than fill the next location with overflow condition.
5) Display the open loop addressing hash table with static hash array.

EXPERIMENT-11

Aim: Arrange words in dictionary order using Binary Search Tree In order Traversal. Implement binary search tree for word representation and make in order traversal for shorting in dictionary order.

Tools / Apparatus: O.S.: Microsoft Windows (any) / Linux / DOS
                                        Packages: Turbo/Borland/GNU - C/C++

Procedure:

Follow the following steps to arrange words in dictionary order using Binary Search Tree.

1) Generate the binary search tree node using template concept.
2) Insert each node in following order.
   If word in temporally node is less than the root node word than insert in left side to root.
   Else insert to right side.

3) Make the Inorder traversal and display the words. You will get in dictionary order.

EXPERIMENT-12

Aim: Implement generalizes representation of Link List for polynomial, matrix …etc type of application.
Tools / Apparatus: O.S.: Microsoft Windows (any) / Linux / DOS
                         Packages: Turbo/Borland/GNU - C/C++
Procedure:

Follow the steps to generate the generalize list.

1) Implement List, List Iterator and List Node class as per given in text book.
   - List class is used for implementing the list as a general format using template concept.
    - generate the list nodes which accept all data type members.
   - Implement add, create, first, last … etc functions in List Iterator class.

2) Use header file List.h for representing matrix, polynomial …etc.

3) Make the instant of list iterator class and use for making matrix or polynomial equation using generalize list class.

4) Use normal addition or multiplication function for polynomial or metrix addition.

References

- Data Structures& Algorithms, Aho, Ullman, Addision Wesley
- An Introduction to Data Strucures with applications, Tremblay, Sorenson, McGraw Hill.
- The art of Computer Programming Vol.I & III, Kunth, Addision Wesley.
- Data Structures using C and C++ Yedidyah Langsam, Tenenbaum

Department of Information Technology,Faculty of Technology,D.D.University,Nadiad.

18