LINKED LIST CONCEPTS

Objectives:

In this lab, student will be able to:

- Understand list as a data structure
- Design programs using linked list concepts

I. SOLVED EXERCISE:

1) Implement stack using singly linked list.

Description: Implementing a stack using linked list is performed by calling insert beginning for Push operation and calling delete beginning for a pop operation. Push is performed by adding a node to the beginning of the list and updating the header. Pop operation is defined as deleting a node from the beginning of the list and updating the header accordingly.

Algorithm: Push

Step1: Create a new node say

newnode =(struct node *)malloc(sizeof(struct node));

Step2: Check whether a node has been created or not if newnode is

NULL node is not created

i.e. if(newnode= =NULL)

{ printf("Out of memory");

exit(0); } else

Step3: Insert the data in data field

newnode->data = element;

Step4: Check whether the list is empty, if so then hold the

address of newnode **top**.

i.e if(top= =NULL) { top =newnode;

top->link=NULL; }

else

Step5: add the new node to the top end

newnode->link=top; And update the top i.e top=newnode;

Step6: Stop.

Algorithm: POP

Step1: Make temp to point to the top element

temp=top;

Step2: make the node next to the list pointer as the beginning of list

top = top->link; //equivalent to top=top-1 in array implementation

Step3: separate temp from chain

temp->link=NULL

delete the first node pointed by temp

free (temp);

Step4: stop

Program:

File name: stack_sll_fun.h

```
typedef struct node
{       int info;
        struct node *link;
}NODE;

NODE* push(NODE *list,int x)
{       NODE *new,*temp;
        new=(NODE*) malloc(sizeof(NODE));
        new->link=list;
        new->info=x;
        return(new);
}

NODE* pop(NODE *list)
{       NODE *prev,*temp;
        if(list==NULL)
        {
                printf("\nStack Underflow\n");
```

```
                return(list);
        }
        temp=list;
        printf("Deleted element is %d",temp→info);
        free(temp);
        list = list→link;
        return(list);
}

void display(NODE *list)
{
        NODE *temp;
        printf("\n\nSTACK:");
        if(list==NULL)
        {
                printf("  Stack is empty");

        printf("\n\n**********************************************");
                return;
        }
        temp=list;
        while(temp!=NULL)
        {
                printf("%5d",temp->info);
                temp=temp->link;
        }
        printf(" <- TOP");
        printf("\n\n**********************************************");
}

int getchoice()
{
        int ch;
```

```
        printf("*********************************************\n\n");
        printf("--------------------Menu--------------------\n");
        printf("1. Push\n2. Pop\n3. Display\n4. Exit\n");
        printf("Enter your choice:");
        scanf("%d",&ch);
        return(ch);
    }
```

File name: stack_sll.c

```
    #include<stdio.h>
    #include "stack_sll_fun.h"
    int main()
    {
        NODE *list;
        int x,ch;
        list=NULL;
        while(1)
        {
            ch=getchoice();
            switch(ch)
            {
                case 1: printf("Enter the element to be pushed:");
                    scanf("%d",&x);
                    list=push(list,x);
                    display(list);
                    break;
                case 2: list=pop(list);
                    display(list);
                   break;
                case 3: display(list);
                    getch();
                    break;
                case 4: exit(1);
```

```
                                default: printf("\nInvalid choice");
                printf("\n\n***********************************************");
                                }
                }
                return 0;
}
```

Sample Input and Output

```
**********************************************
--------------------Menu--------------------
1. Push
2. Pop
3. Display
4. Exit
Enter your choice:1
Enter the element to be pushed:23

STACK:   23 <- TOP

**********************************************
--------------------Menu--------------------
1. Push
2. Pop
3. Display
4. Exit
Enter your choice:
1
Enter the element to be pushed:34

STACK:   23   34 <- TOP

**********************************************
--------------------Menu--------------------
```

1. Push

2. Pop

3. Display

4. Exit

Enter your choice:

2

Popped element is 34

STACK: 23 <- TOP

****

LINKED LIST APPLICATIONS

Objectives:

In this lab, student will be able to:

- Identify the need for list data structure in a given problem.
- Develop c programs applying linked concepts

I. SOLVED EXERCISE:

1) Given two polynomials, write a program to perform the addition of two polynomials represented using doubly circular linked list with header and display the result.

Description: Suppose we wish to manipulate polynomials of the form $p(x) = c_1*x\text{^}e_1 + c_2*x\text{^}e_2 + \ldots + c_n* x\text{^} e_n$, where $e_1 > e_2 > \ldots > e_n \geq 0$. Such a polynomial can be represented by a linked list in which each cell has three fields: one for the coefficient c_i, one for the exponent e_i, and one for the pointer to the next cell. For example, the polynomial $4x^2+2x+4$, can be viewed as list of the following pairs (4,2),(2,1),(4,0). Therefore, we can use a linked list in which each node will have four fields to store coefficient, exponent and two link fields. The right link will be pointing to the next node and left link will be pointing to the previous node. The last node's right link will point to the header in a circular list and the left link of header is made to point to the last node. A dummy node is maintained as head to the list.

Algorithm: Add polynomials

Step1: Take references to the header of first and second polynomial list

one=h1->rlink; two=h2->rlink; and h3 is a pointer to the resulting list.

Step2: Traverse through the lists by checking the exponents until either of the pointer is not null, i.e, While(one!=h1 && two!=h2) do the following

Step3: If the exponents of botht the lists are equals, add the coefficients and insert added coefficient and the exponent to a resultant list i.e

```
if((one->ex)==(two->ex))
    {       h3=add(h3,((one->info)+(two->info)),one->ex);
            one=one->rlink;
            two=two->rlink;
    }
```

Step4: Else if exponent of first list pointer is greater, then insert first list pointer exponent and coefficient to result list. Update the first lists pointer only to the next node and continue with step2.

```
            h3=add(h3,one->info,one->ex);
        one=one->rlink;
```

Step5: Else insert second list pointer exponent and coefficient to result list. Update the second lists pointer only to the next node and continue with step2.

```
        h3=add(h3,two->info,two->ex);
        two=two->rlink;
```

Step6: If there are any terms left in second list copy it to the resultant list i.e

```
while(two!=h2)
    {       h3=add(h3,two->info,two->ex);
            two=two->rlink;
    }
```

Step7: If there are any terms left in first list copy it to the resultant list i.e

```
while(one!=h1)
    {       h3=add(h3,one->info,one->ex);
            one=one->rlink;
    }
```

Step8: return a pointer to the resultant list h3

Program:

File name: poly_add_dll_fun.h

```
struct node
{       int info;
        int ex;
        struct node *llink;
        struct node *rlink;
};
typedef struct node *NODE;
NODE add(NODE head,int n,int e)
{
       NODE temp,last;
        temp=(NODE)malloc(sizeof(struct node));
        temp->info=n;
        temp->ex=e;
        last=head->llink;
        temp->llink=last;
        last->rlink=temp;
        temp->rlink=head;
        head->llink=temp;
        return head;
}
NODE sum(NODE h1,NODE h2,NODE h3)
{
        NODE one,two;
        one=h1->rlink;
        two=h2->rlink;
        while(one!=h1 && two!=h2)
        {       if((one->ex)==(two->ex))
```

```
                {       h3=add(h3,((one->info)+(two->info)),one->ex);
                        one=one->rlink;
                        two=two->rlink;
                }
                else if(one->ex>two->ex)
                {       h3=add(h3,one->info,one->ex);
                        one=one->rlink;
                }
                else
                {       h3=add(h3,two->info,two->ex);
                        two=two->rlink;
                }
        }
        while(two!=h2)
                {       h3=add(h3,two->info,two->ex);
                        two=two->rlink;
                }

        while(one!=h1)
                {       h3=add(h3,one->info,one->ex);
                        one=one->rlink;
                }
        return h3;
}

void display(NODE head)
{       printf("\ncontents of list are\n");
        NODE temp=NULL;
        temp=head->rlink;
        while(temp!=head)
        {       printf("%d %d\t",temp->info,temp->ex);
                temp=temp->rlink;
        }
```

```
}
```

File name: ploy_add_dll.c

```
#include<stdio.h>
#include<stdlib.h>
#include “poly_add_dll_fun.h”
int main()
{       int m,n,e,k;
        NODE h1,h2,h3,h4;
        h1=(NODE)malloc(sizeof(struct node));
        h2=(NODE)malloc(sizeof(struct node));
        h3=(NODE)malloc(sizeof(struct node));
        h4=(NODE)malloc(sizeof(struct node));
        h1->rlink=h1;
        h1->llink=h1;
        h2->rlink=h2;
        h2->llink=h2;
        h3->rlink=h3;
        h3->llink=h3;
        h4->rlink=h4;
        h4->llink=h4;
        printf("\nnumber of nodes in list1\n");
        scanf("%d",&n);
        while(n>0)
        {       scanf("%d",&m);
                scanf("%d",&e);
                h1=add(h1,m,e);
                n--;
        }
        display(h1);
```

```
        printf("\nnumber of nodes in list2\n");
        scanf("%d",&k);
        while(k>0)
        {       scanf("%d",&m);
                scanf("%d",&e);
                h2=add(h2,m,e);
                k--;
        }
        display(h2);
        printf("\nthe sum is\n");
        h3=sum(h1,h2,h3);
        display(h3);
        return 1;
 }
```

Sample Input and Output

```
number of nodes in list1
3
3 3 3 2 4 1
contents of list are
3 3     3 2     4 1
number of nodes in list2
3
2 3 2 2 1 1
contents of list are
2 3     2 2     1 1
the sum is
contents of list are
5 3     5 2     5 1
```

TREE CONCEPTS

Objectives:

In this lab, student will be able to:

- Understand tree as a data structure
- Design programs using tree concepts

I. SOLVED EXERCISE:

1) Create a binary tree using recursion and display its elements using all the traversal methods.

Description: To create and maintain the information stored in a binary tree, we need an operation that inserts new nodes into the tree. We use the following insertion approach. A new node is made root for the first time and there after a new node is inserted either to the left or right of the node. -1 one is entered to terminate the insertion process. This is recursively done for left subtree and the right subtree respectively.

Program:

File name: binary_tree_recursion_fun_1.h

```
typedef struct node
{
   int data;
   struct node *lchild;
   struct node *rchild;     } *NODE;

NODE Create_Binary_Tree()
{
   NODE temp;
   int ele;
   printf("Enter the element to inserted (-1 for no data):");
   scanf("%d",&ele);
   if(ele==-1)
```

```
        return NULL;
    temp=(NODE*)malloc(sizeof(struct node));
    temp->data=ele;
    printf("Enter lchild child of %d:\n",ele);
    temp->lchild=create();

    printf("Enter rchild child of %d:\n",ele);
    temp->rchild=create();
    return temp;
}
void inorder(NODE *ptr)
{
        if(ptr!=NULL)
        {
                inorder(ptr->lchild);
                printf("%5d",ptr->info);
                inorder(ptr->rchild);
        }
}

void postorder(NODE *ptr)
{
        if(ptr!=NULL)
        {
                postorder(ptr->lchild);
                postorder(ptr->rchild);
                printf("%5d",ptr->info);
        }
}

void preorder(NODE *ptr)
{
        if(ptr!=NULL)
```

```
        {
                printf("%5d",ptr->info);
                preorder(ptr->lchild);
                preorder(ptr->rchild);
        }
}
```

File name: binary_tree.c

```
#include<stdio.h>
#include "binary_tree_recursion_fun_1.h"
int main()
{
        int n,x,ch,i;
        NODE *root;
        root=NULL;
        while(1)
        {
        printf("**********Output**********\n\n");
                printf("----------Menu----------\n");
                printf(" 1. Insert\n 2. All traversals\n 3. Exit\n");
                printf("Enter your choice:");
                scanf("%d",&ch);
                switch(ch)
                {
                        case 1: printf("Enter node :\n");
                                scanf("%d",&x);
                                root=Create_Binary_Tree();
                                break;
                        case 2: printf("\nInorder traversal:\n");
                                inorder(root);
                                printf("\nPreorder traversal:\n");
                                preorder(root);
                                printf("\nPostorder traversal:\n");
                                postorder(root);
```

```
        printf("\n\n*********************************************");
                                        break;
                                case 3: exit(0);
                        }
                }
                return 0;
}
```

Sample Input and Output

```
**********Output**********

----------Menu----------
 1. Insert
 2. All traversals
 3. Exit
Enter your choice:1
Enter node:
20      25      -1      30      40      -1      -1      -1
**********Output**********

Enter your choice:2

Inorder traversal:      25      30      20      40      28
Preorder traversal:     20      25      30      28      40
Postorder traversal:    30      25      40      28      20

            *********************************************
```

Questions for Lab7

1) Implement a queue using singly linked list without header node.
2) Perform UNION and INTERSECTION set operations on singly linked lists with header node.

Questions for Lab8

1) Add two long positive integers represented using circular doubly linked list with header node.

2) Write a menu driven program to do the following using iterative functions:
   i) To create a BST for a given set of integer numbers
   ii) To delete a given element from BST.
   iii) Display the elements using iterative in-order traversal.