Lab1 and Lab2 (Refer DSA lab 2 instructions, DSA lab 3 and 4 instructions videos)

SOLVING PROBLEMS USING ARRAYS, POINTERS AND DYNAMIC MEMORY ALLOCATION FUNCTIONS

Objectives:

In this lab, student will be able to:

i) Familiarize with syntax and usage of pointers and pointer to arrays and dynamic memory management functions

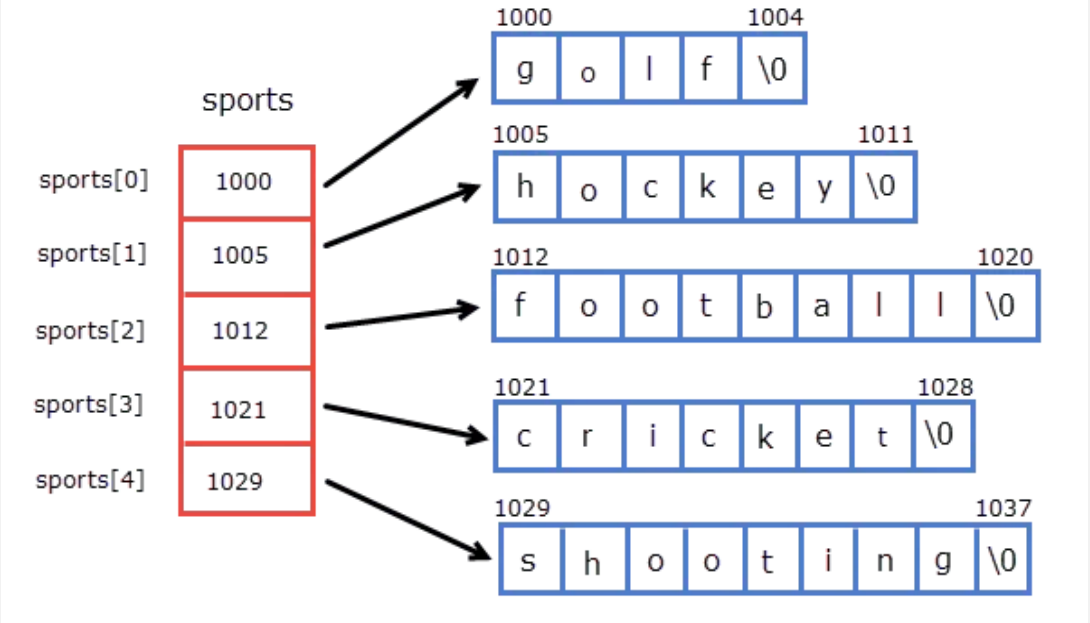
ii) Write C programs making use of pointer concepts and dynamic memory allocation functions

I. SOLVED EXERCISE:

1) Write a program to read n names of different sports and store them using array pointers. Use dynamic memory allocation and deallocation functions. The program should display all the names and deallocate the dynamic memory at the end of the program.

Description: The following figure illustrates the use of array of character pointers to store the names of different sports. First, an array of character pointers is created. Then, the name of different sports is read from the user and memory is allocated as per the input string length. This representation is memory efficient in comparison to the storage of multiple strings using 2D array of characters.

Fig. Memory representation of array of pointers



Program:

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
int main(){
   int i,n;
   char *sports[10];
   char str[100];

   printf("\n enter the number of sports \n");
   scanf("%d", &n);
   printf("\nenter the names of sports:\n");
   for (i = 0; i < n; i++)
   {
      scanf("%s", str);
      //allocating memory equal to the length of string + 1
     //Last 1 byte to accommodate the '\0'
      sports[i] = (char*) calloc(strlen(str)+1, sizeof(char));
      strcpy(sports[i],str);
   }
```

```
    printf("\nGiven list of sports: \n");
    for (i = 0; i < n; i++)
            printf("%s\n", sports[i]);

   //Deallocate the dynamic memory
    for (i = 0; i < n; i++)
        free(sports[i]);

    return 0;
}
```

SOLVING PROBLEMS USING RAGGED ARRAYS, STRUCTURES AND POINTERS

Objectives:

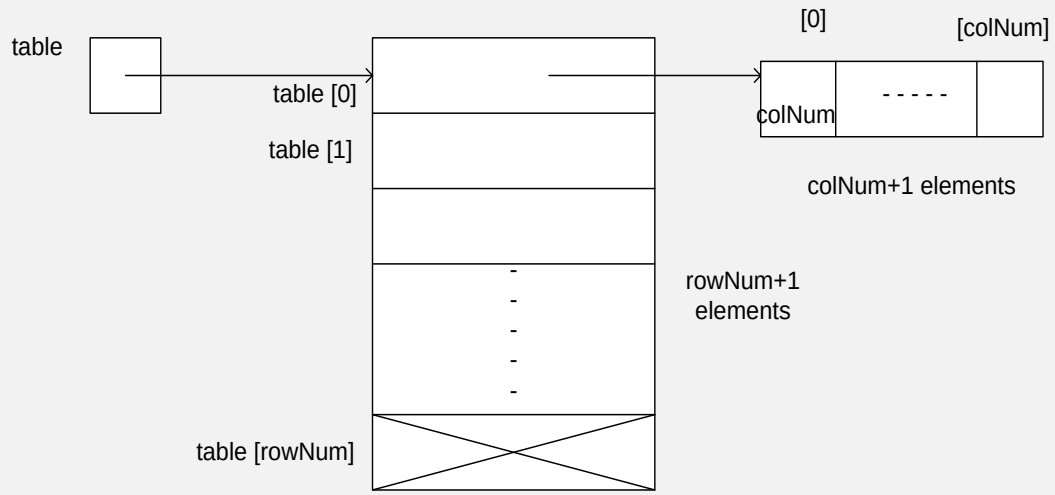
In this lab, student will be able to:

- Familiarize the usage of ragged arrays
- Write programs using structures and pointers
- Familiarize of dynamic memory allocation for structures

I. SOLVED EXERCISE:

1) Write a C program to implement a ragged array dynamically.

Description: In a ragged array the *table* pointer points to the first pointer in an array of pointers. Each array pointer points to a second array of integers, the first element of which is the number of elements in the list. A sample ragged array structure is shown below.



Algorithm: Construct a ragged array

Step 1: Declare a ragged array as a variable *table*.

Step 2: Ask the user for row size and set a variable – *rowNum*

Step 3: Allocate space for *(rowNum+1)* pointers as row pointers. The last row pointer will hold NULL

Step 4: Ask the user for column size and set a variable – *colNum*

Step 5: Allocate space for *(colNum+1)* data elements. The first element will hold value contained in colNum itself.

Step 6: Repeat step 3 for all rows

Step 7 : Display ragged array contents. Step 8: Stop

Program:

```
#include<stdio.h>
#include<stdlib.h>
int main(){
int rowNum, colNum, i, j;
int **table;
printf("\n enter the number of rows \n");
scanf("%d", &rowNum);
table = (int **) calloc(rowNum+1, sizeof(int *));
for (i = 0; i < rowNum; i++)  /* this will tell which row we are in */
{
        printf("enter size of %d row", i+1);
        scanf("%d", &colNum);
        table[i] = (int *) calloc(colNum+1, sizeof(int));
            printf("\n enter %d row elements ", i+1);
                    for (j = 1; j <= colNum;  j++)
                     {          scanf("%d", &table[i][j]);                          }
        table[i][0] = colNum;
        printf("size of row number [%d] = %d", i+1, table[i][0]);
}
table[i] = NULL;
for (i = 0; i < rowNum; i++) /* this will tell which row we are in */
{
        printf("displaying %d row elements\n", i+1);
              for (j = 0; j <= *table[i];  j++)
                              {
```

```
                    printf("%5d", table[i][j]);
                    }
printf("\n");
}
//freeup the memory
for (i = 0; i < rowNum; i++) {
    free(table[i]);
  }
free(table);
return 0;
}
```

Sample input and output:

```
enter the number of rows: 3
enter size of row 1: 4
enter row 1 elements: 10 11 12 13
enter size of row 2: 5
enter row 2 elements: 20 21 22 23 24
enter size of row 3
enter row 3 elements: 30 31 32
displaying
10 11 12 13
20 21 22 23 24
30 31 32
```

SOLVING PROBLEMS USING RECURSION

Objectives:

In this lab, student will be able to:

- Formulate a recursive solution to a given problem
- Familiarize with recursion concept in C programs

I. SOLVED EXERCISE:

1) Write a C program to implement binary search

Title: IMPLEMENT BINARY SEARCH

1. Program to perform binary search on a set of keys.

Aim: To understand the working recursive function call and also binary search technique.

Description Binary search method employs the process of searching for a record only in half of the list, depending on the comparison between the element to be searched and the central element in the list. It requires the list to be sorted to perform such a comparison. It reduces the size of the portion to be searched by half after each iteration.

Algorithm: Binary Search

Assumption: The input array is in sorted order.

Step1: Given array A[low, high], find the value of mid location as mid =(low+high)/2

Step2: if Low > high return a Failure status and terminate the search; go to step 4.

Step3: Else Compare the key (element to be searched) with the mid element.

If key matches with middle element, we return the mid index; go to step 4.

Else If key is greater than the mid element, then key can only lie in right half sub-array after the mid element. So we recur for right half.

Else (x is smaller) recur for the left half until there are no more elements left in the array.

Step4: stop

Program:

File Name : binary_search_function.h

```
int bin_search(int low,int high,int item,int a[])
{
    int mid;
    if(low>high)
        return(-1);
    else
    {
        mid=(low+high)/2;
        if(item==a[mid])
            return(mid);
        else if(item<a[mid])
            return(bin_search(low,mid-1,item,a));
        else
            return(bin_search(mid+1,high,item,a));
    }
}
```

File Name : binary_search.c

```
#include <stdio.h>
#include "binary_search_function.h"
void main()
{
    int i, pos, a[30],n, item;
    printf("Enter number of items:");
    scanf("%d",&n);
    printf("Enter the elements in ascending order:\n");
    for(i=0;i<n;i++)
        scanf("%d",&a[i]);
    printf("Enter element to be searched:");
    scanf("%d",&item);
    pos=bin_search(0,n-1,item,a);
    if(pos!=-1)
        printf("Item found at location %d",pos+1);
    else
        printf("Item not found");
}
```

Sample Input and Output:

Enter number of items: 6

Enter the elements in ascending order:

```
12 23 54 65 88 99

Enter element to be searched:99

Item found at location 6
```

Questions for LAB1

1. Write a function Smallest to find the smallest element in an array using pointer. Create a dynamically allocated array and read the values from keyboard in main. Display the result in the main function.
2. Implement a C program to read, display and to find the product of two matrices using functions with suitable parameters. Note that the matrices should be created using dynamic memory allocation functions and the elements are accessed using array dereferencing.
3. Samuel wants to store the data of his employees, which includes the following fields: (i) Name of the employee (ii) Date of birth which is a collection of {day, month, year} (iii) Address which is a collection of {house number, zip code and state}. Write a 'C' program to read and display the data of N employees using pointers to array of structures.

Note : You may use the following structure .

```
struct DOB {
   int day, month, year;
}
struct ADRS {
   int house_no;
   long zipcode;
   char state[20];
}
struct EMPLOYEE {
   char name[20];
   struct DOB dob;
   struct ADRS address;
}
struct EMPLOYEE emp[10];
EMPLOYEE* ptr = emp;
```

Questions for LAB2

1. Create a structure STUDENT consisting of variables of structures:

i. DOB {day, month (use pointer), year},

ii. STU_INFO {reg_no, name(use pointer), address},

iii. COLLEGE {college_name (use pointer), university_name}

where structure types from i to iii are declared outside the STUDENT independently. Show how to read and display member variables of DOB type if pointer variable is created for DOB inside STUDENT and STUDENT variable is also a pointer variable. The program should read and display the values of all members of STUDENT structure.

Note: You may use the following structure.

```
struct DOB{
   int day;
   char* mth;
   int year;
}
struct STU_INFO{
   int reg_no;
   char* name;
   char[20] adrs;
}
struct COLLEGE{
   char* clg_name;
   char[20] univ_name;
}
struct STUDENT{
   struct DOB dob;
   struct STU_INFO stu_info;
   struct COLLEGE clg;
}

struct STUDENT student;
char[10] month;
```

scanf(“%s”, month);

student.dob.mth = (char*) malloc (sizeof (month);

strcpy(student.dob.mth, month);

2. Write C programs using recursion to copy one string to another using Recursion.
3. Write C programs using recursion to check whether a given String is Palindrome or not, using Recursion

4. Write C programs using recursion to simulate the working of Tower of Hanoi for n disks. Print the number of moves.

***** Refer DSA Lab5 and Lab6 instructions videos *****

STACK CONCEPTS

Objectives:

In this lab, student will be able to:

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

I. SOLVED EXERCISE:

1) Write a c program to check if the given parenthesized expression has properly matching open and closing parenthesis.

Description: We use a stack to check the expression for matching opening and closing parenthesis. Here, the expression is scanned from start to end if an opening brace is encountered then it is pushed on to the stack. When a closing parenthesis is encountered a pop operation is performed. Ideally, if the number of opening and closing braces matches, then the stack will be empty after checking the entire expression.

Algorithm:

```
Step1: Set balanced to true
Step2: Set symbol to the first character in current expression
Step3: while (there are still more characters AND expression is still balanced)
            if (symbol is an opening symbol)
                Push symbol onto the stack
            else if (symbol is a closing symbol)
                if the stack is empty
                    Set balanced to false
                Else
                    Set openSymbol to the character at the top of the stack
                    Pop the stack
                    Set balanced to (symbol matches openSymbol)
```

```
            Set symbol to next character in current expression
Step4:  if (balanced)
            Write "Expression is well formed."
        else
            Write "Expression is not well formed."
  Step5: stop
```

Program:

File name: stack_operations.h

```
# define MAX 10
# define true 1
# define false 0
/* Structure definition */
typedef struct
{
        char item[MAX];
        int top;
}stack;
void push(stack *ps,char x);
char pop(stack *ps);
int empty(stack *ps);
/* Push operation */
void push(stack *ps,char x)
{
        if (ps->top!=MAX-1)
        {
                ps->top++;
                ps->item[ps->top]=x;
        }
}
```

```
 /* Pop operation */
char pop(stack *ps)
{
        if(!empty(ps))
          return(ps->item[ps->top--]);
}

/* Stack empty operation */
int empty(stack *ps)
{
        if (ps->top==-1)
                return(true);
        else
                return(false);
}
```

File name: check_expr.c

```
#include <stdio.h>
#include <stdlib.h>
#include "stack_operations.h"
void main()
{
        char expn[25],c,d;
        int i=0;
        stack s;
        s.top=-1;
        printf("\n Enter the expression: ");
        gets(expn);
        while((c=expn[i++])!='\0')
        {
                if(c=='(')
                        push(&s,c);
                else
                        if(c==')')
```

```
                        {
                                d=pop(&s);
                                if(d!='(')
                                {
                                printf("\n Invalid Expression");
                                break;
                                }
                        }
        }
        if(empty(&s))
                printf("\n Balanced Expression");
        else
                printf("\n Not a Balanced  Expression");
}
```

Sample Input and Output

Run 1:

Enter the expression: (a+b)+(c*d*(a-b)

Not a Balanced Expression

Run 2:

Enter the expression: (a+b)+(c*d*(a-b))

Balanced Expression

STACK APPLICATIONS

Objectives:

In this lab, student will be able to:

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

I. SOLVED EXERCISE:

1) Program for evaluation of postfix expression in C

Algorithm for Evaluation of Postfix Expression

Create an empty stack and start scanning the postfix expression from left to right.

- If the element is an operand, push it into the stack.
- If the element is an operator **O**, pop twice and get A and B respectively. Calculate B**O**A and push it back to the stack.
- When the expression is ended, the value in the stack is the final answer.

Evaluation of a postfix expression using a stack is explained in below example (fig. 2):

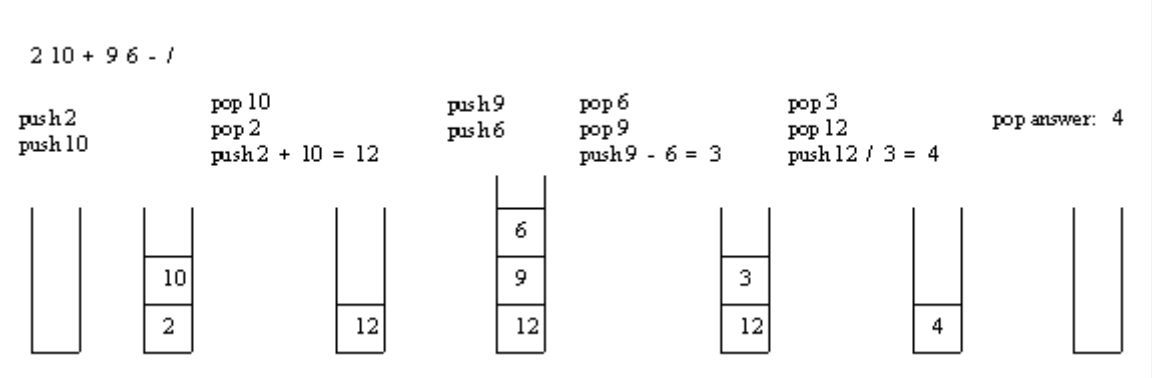


Figure 2: Illustrates the evaluation of a postfix expression using a stack

File name: eval_postfix_fun.h

```
#define MAX 20

typedef struct stack
{
   int data[MAX];
   int top;
}stack;

void init(stack *);
int empty(stack *);
```

```
int full(stack *);
int pop(stack *);
void push(stack *,int);
int evaluate(char x,int op1,int op2);

int evaluate(char x,int op1,int op2)
{
   if(x=='+')
      return(op1+op2);
   if(x=='-')
      return(op1-op2);
   if(x=='*')
      return(op1*op2);
   if(x=='/')
      return(op1/op2);
   if(x=='%')
      return(op1%op2);
}

void init(stack *s)
{
   s->top=-1;
}

int empty(stack *s)
{
   if(s->top==-1)
      return(1);

   return(0);
}

int full(stack *s)
{
   if(s->top==MAX-1)
      return(1);

   return(0);
}

void push(stack *s,int x)
{
   s->top=s->top+1;
   s->data[s->top]=x;
}

int pop(stack *s)
{
   int x;
   x=s->data[s->top];
```

```
    s->top=s->top-1;
    return(x);
}
```

File name:eval_postfix_expr.c

```
#include<stdio.h>
#include "eval_postfix_fun.h"
int main()
{
    stack s;
    char x;
    int op1,op2,val;
    init(&s);
    printf("Enter the expression(eg: 59+3*)\nsingle digit operand and operators
            only:");

    while((x=getchar())!='\n')
    {
        if(isdigit(x))
            push(&s,x-'0');        /*x-'0' for removing the effect of ascii  */
        else
        {
            op2=pop(&s);
            op1=pop(&s);
            val=evaluate(x,op1,op2);
            push(&s,val);
        }
    }
    val=pop(&s);
    printf("\nvalue of expression=%d",val);
    return 0;
}
```

Sample Input and Output:

Enter the expression(eg: 59+3*)
single digit operand and operators only: 12+3*
value of expression= 9

Questions for Lab3

Write a 'C' program to:

1) Implement a menu driven program to define a stack of characters. Include push, pop and display functions. Also include functions for checking error conditions such as underflow and overflow (ref. figure 1) by defining isEmpty and isFull functions. Use these function in push, pop and display functions appropriately. Use type defined structure to define a STACK containing a character array and an integer top. Do not use global variables.

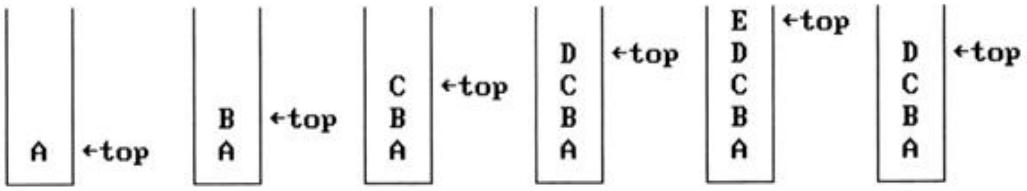


Figure 1: Inserting and deleting elements in a stack

2) Convert a given decimal number to binary using stack.
3) Determine whether a given string is palindrome or not using stack.

Questions for Lab4

Write a C program to:

1) Evaluate a given prefix expression using stack.
2) Convert an infix expression to prefix.
3) Implement two stacks in an array.

(Refer video DSA Lab 7 instructions and DSA Lab 8 instructions)

QUEUE CONCEPTS

Objectives:

In this lab, student will be able to:

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

I. SOLVED EXERCISE:

1) Implement a queue of integers. Include functions insertq, deleteq and displayq.

Description:

Whenever we perform an insertion the rear pointer is incremented by 1 and front remains the same. Whenever a deletion is performed the front is incremented by one. But in real life problem the front pointer will be in the same position the object which is in the queue shifts to the front. But when we implement a queue in computer the front moves, this is because if we shift the elements to the front the time complexity increases. To implement a linear queue we consider two pointers or markers front and rear. Initial value of front and rear is assumed to be -1

Algorithm: Queue Insert

Step 1. Check for overflow

```
If R== N-1
        Then write ('OVERFLOW')
    Return
```

Step2. else Increment rear pointer

```
R= R +1
```

Step3: Insert element

```
Q[R ] = val
```

Step4: If F = -1

```
then F = 0
```

Return.

Algorithm: Queue Delete

Step1: Check for underflow

If F = = -1

Then Write ('UNDERFLOW')

Return

Step2: Delete an element

val = Q[F]

Step3: Queue empty?

if F > R

then F=R =-1

else F= F +1

Step4. Return an element

Return(val)

Step5: Stop

Program:

File name: queue_fun.h

```
#define MAX 20
typedef struct {
   int x[MAX];
   int front;
   int rear;
} queue;

void insertq(queue *, int);
void displayq(queue);
int deleteq(queue *);
```

Formatted Table

```
void insertq(queue * q,int x)
 {
   if(q->rear==MAX)
   {
      printf("\nOverflow\n");
   }
   else
   {
      q->x[++q->rear]=x;
      if(q->front==-1)
      {
         q->front=0;
      }
   }
 }

 int deleteq(queue * q)
 {
   int x;
   if(q->front==-1)
   {
      printf("\nUnderflow!!!\n");
   }
   else if(q->front==q->rear)
   {
      x=q->x[q->front];
      q->front=q->rear=-1;
      return x;
   }
   else
   {
      return q->x[q->front++];
```

```
    }
  }

  void displayq(queue q)
 {
    int i;
    if(q.front==-1&&q.rear==-1)
    {
       printf("\nQueue is Empty!!!");
    }
    else
    {
       printf("\nQueue is:\n");
       for(i=q.front;i<=q.rear;i++)
       {
          printf("%d\n",q.x[i]);
       }
    }
}
```

File name: queue.c

```
#include <stdio.h>
#include "queue_fun.h"

int main()
{
   queue q;
   q.front=-1;
   q.rear=-1;
   int ch,x,flag=1;
   while(flag)
```

```
    {
      printf("\n\n1. Insert Queue\n2. Delete Queue\n3. Display Queue\n4. Exit\n\n");
      printf("Enter your choice: ");
      scanf("%d",&ch);
      switch(ch)
      {
         case 1:
           printf("\nEnter the Element:");
           scanf("%d",&x);
           insertq(&q,x);
           break;
        case 2:
           x=deleteq(&q);
           printf("\nRemoved %d from the Queue\n",x);
           break;
        case 3:
           displayq(q);
           break;
        case 4:
           flag=0;
           break;
        default:
           printf("\nWrong choice!!! Try Again.\n");
      }
    }
  return 0;
}
```

QUEUE APPLICATIONS

Objectives:

In this lab, student will be able to:

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

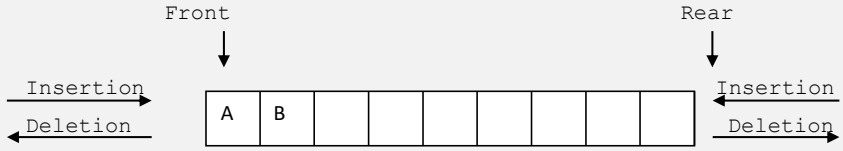
I. SOLVED EXERCISE:

1) Implement a dequeue of integers with following functions.

a) deleteLeft b) addLeft c) deleteRight d) addRight e) display

Description:

A queue that supports insertion and deletion at both the front and rear is called **double-ended queue or Dequeue.** A Dequeue is a linear list in which elements can be added or removed at either end but not in the middle.



The operations that can be performed on Dequeue are

1. Insert to the beginning
2. Insert at the end
3. Delete from the beginning
4. Delete from end

There are two variation of Dequeue namely, an input-restricted Dequeue and an Output restricted Dequeue – which are intermediate between a Dequeue and a queue. Specifically, an input-restricted Dequeue is a Dequeue which allows insertions at only one end of the list but allows deletions at both ends of the list; and an output-restricted Dequeue is a Dequeue which allows deletions at only one end of the list allows insertion at both ends of the list.

Algorithm: Insert End (rear)

Step1: if(front==MAX/2)

Write("Queue Full: Can't enter more elements at the end of queue");

return;

Step2: Else Insert the element and update the front pointer i.e queue[front]=token;

front=front+1;

Algorithm: Insert Start (front)

Step1: If (front==MAX/2)

Write("Queue Full: can't enter more elements at the start of queue");

return;

Step2: Insert the element by updating the rear pointer i.e

rear=rear-1;

queue[rear]=token;

Algorithm: Delete End (rear)

Step1: If(front==rear)

Write("Queue empty");

return 0;

Step2: Otherwise update the front and return the element i.e

front=front-1;

t=queue[front+1];

return t;

Algorithm: Delete Start (front)

Step1: If(front==rear)

Write("\nQueue empty");

return 0;

Step2: Update the rear pointer and return the element i.e

rear=rear+1;

t=queue[rear-1];

return t;

Program:

File name: deque_fun.h

```
#define MAX 30
typedef struct dequeue
{
   int data[MAX];
   int rear,front;
}dequeue;

void initialize(dequeue *p);
int empty(dequeue *p);
int full(dequeue *p);
void enqueueR(dequeue *p,int x);
void enqueueF(dequeue *p,int x);
int dequeueF(dequeue *p);
int dequeueR(dequeue *p);
void print(dequeue *p);

void initialize(dequeue *P)
{
   P->rear=-1;
   P->front=-1;
}

int empty(dequeue *P)
{
   if(P->rear==-1)
      return(1);
   return(0);
```

```
}

int full(dequeue *P)
{
   if((P->rear+1)%MAX==P->front)
      return(1);

   return(0);
}
void enqueueR(dequeue *P,int x)
{
   if(empty(P))
   {
      P->rear=0;
      P->front=0;
      P->data[0]=x;
   }
   else
   {
      P->rear=(P->rear+1)%MAX;
      P->data[P->rear]=x;
   }
}

void enqueueF(dequeue *P,int x)
{
   if(empty(P))
   {
      P->rear=0;
      P->front=0;
      P->data[0]=x;
   }
   else
```

```
    {
        P->front=(P->front-1+MAX)%MAX;
        P->data[P->front]=x;
    }
}

int dequeueF(dequeue *P)
{
    int x;
    x=P->data[P->front];
    if(P->rear==P->front)    /*delete the last element */
        initialize(P);
    else
        P->front=(P->front+1)%MAX;
    return(x);
}

int dequeueR(dequeue *P)
{
    int x;
    x=P->data[P->rear];
    if(P->rear==P->front)
        initialize(P);
    else
        P->rear=(P->rear-1+MAX)%MAX;
    return(x);
}

void print(dequeue *P)
{
    if(empty(P))
    {
        printf("\nQueue is empty!!");
```

```
        exit(0);
    }
    int i;
    i=P->front;
    while(i!=P->rear)
    {
        printf("\n%d",P->data[i]);
        i=(i+1)%MAX;
    }
    printf("\n%d\n",P->data[P->rear]);
}
```

File name: dequeuer.c

```
#include<stdio.h>
#include<process.h>
#include "dequeue_fun.h"
int main()
{
    int i,x,op,n;
    dequeue q;
    initialize(&q);
    do
    {

printf("\n1.Create\n2.Insert(rear)\n3.Insert(front)\n4.Delete(rear)\n5.Delete(front)");
        printf("\n6.Print\n7.Exit\n\nEnter your choice:");
        scanf("%d",&op);
        switch(op)
        {
            case 1: printf("\nEnter number of elements:");
                    scanf("%d",&n);
                    initialize(&q);
```

```
        printf("\nEnter the data:");

        for(i=0;i<n;i++)
        {
           scanf("%d",&x);
           if(full(&q))
           {
              printf("\nQueue is full!!");
              exit(0);
           }
           enqueueR(&q,x);
        }
        break;

case 2: printf("\nEnter element to be inserted:");
        scanf("%d",&x);

        if(full(&q))

        {
           printf("\nQueue is full!!");
           exit(0);
        }
        enqueueR(&q,x);
        break;

case 3: printf("\nEnter the element to be inserted:");
        scanf("%d",&x);
        if(full(&q))
        {
           printf("\nQueue is full!!");
           exit(0);
        }
```

```
                enqueueF(&q,x);
                break;

        case 4: if(empty(&q))
                {
                   printf("\nQueue is empty!!");
                   exit(0);
                }
                x=dequeueR(&q);
                printf("\nElement deleted is %d\n",x);
                break;

        case 5: if(empty(&q))
                {
                   printf("\nQueue is empty!!");
                   exit(0);
                }
                x=dequeueF(&q);
                printf("\nElement deleted is %d\n",x);
                break;

        case 6: print(&q);
                break;
        default: break;
      }
   }while(op!=7);
   return 0;
}
```

Questions for Lab 5

1) Implement a circular queue of integers. Include functions insertcq, deletecq and displaycq.

2) Implement two circular queues of integers in a single array where first queue will run from 0 to N/2 and second queue will run from N/2+1 to N-1 where N is the size of the array.

3) Implement a queue with two stacks without transferring the elements of the second stack back to stack one. (use stack1 as an input stack and stack2 as an output stack).

Questions for Lab6

1) Implement an ascending priority queue.

   Note: An ascending priority queue is a collection of items into which items can be inserted arbitrarily and from which only the smallest item can be removed. If apq is an ascending priority queue, the operation pqinsert(apq,x) inserts element x into apq and pqmindelete(apq) removes the minimum element from apq and returns its value.

2) Implement a queue of strings using an output restricted dequeue (no deleteRight).
   Note: An output-restricted deque is one where insertion can be made at both ends, but deletion can be made from one end only, where as An input-restricted deque is one where deletion can be made from both ends, but insertion can be made at one end only.

3) Write a program to check whether given string is a palindrome using a dequeue.

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.