DATA STRUCTURES AND ALGORITHMS LAB

Record Book

MARCH 2018

PRAKASH KUMAR
1747246

Department of Computer Science,
CHRIST (Deemed to be University)
Bangalore – 560029,
Karnataka, India

Program Number. ***1***

Program Name: ***Implement sequential search and binary search techniques***

Date of Implementation: ***27.11.2017***

Description: ***A menu driven program to implement linear and binary search with necessary validations.***

```
/*Program to implement Sequential Search and Binary Search
*/
#include<stdio.h>
#include<stdlib.h>
#include<ctype.h>

#define MAX 50                    /*Maximum size of the array*/

int seq_search(int [],int);
void sort_arr(int [],int);
int binarySearch(int [],int);
void layout_func();

int main()
{
int choice,arr[MAX],i,length,flag,temp,foundOrNot;
char choice_y_n,check[MAX];
layout_func();
printf("Enter the number of elements in array\n");
scanf("%d",&length);
while(length < 2)
{
printf("\nMinimum elements required should be 2\nRe-Enter Length:\t");
scanf("%d",&length);
}
printf("Enter %d elements\n",length);
```

```
for(i = 0;i < length;i += 1)
{
gets(check);
temp = atoi(check);
if(isdigit(check[0]))
{
arr[i] = temp;
}
else
{
printf("\nOnly Integers as Input\n");
i -= 1;
}

}
for(i = 0;i < length-1; i += 1)
{
if(arr[i] < arr[i+1])
{
//printf("if mein huuu");
flag = 0;
}
else
{
//printf("else mein huuu");
flag = 1;
break;
}
}
do
{
system("cls");
```

```
layout_func();
printf("Please enter your choice\n1. Sequential Search\n2. Binary Search\nInput:\t");
scanf("%d",&choice);
switch(choice)
{
case 1:
{
foundOrNot = seq_search(arr,length);
if(foundOrNot == -1)
{
printf("\nElement Not Found");
}
else
{
printf("\nElement found at position %d",foundOrNot);
}
break;
}
case 2:
{
if(flag == 1)
{
printf("\nArray is not sorted, PERFORMING SORTING");
sort_arr(arr,length);
}
else
{
printf("\nArray is sorted, performing Binary Search");
foundOrNot = binarySearch(arr,length);

if(foundOrNot == -1)
{
```

```
printf("\nElement Not found");
}
else
{
printf("\nElement found at position %d",foundOrNot);
}

}
break;
}
default:
{
printf("\nWrong Choice, Please try again\n");
break;
}
}
printf("\nDo you wish to continue?\nInput(y/n):");
getchar();
scanf("%c",&choice_y_n);
}while(choice_y_n == 'Y' || choice_y_n == 'y');
return 0;
}



void sort_arr(int rcv_arr[MAX],int length)
{
int i,j,temp,foundOrNot;
for(i = 0; i < length; i += 1)
{
/*
* Place the currently selected element array[i]
```

```
* to its correct place.
*/
for(j = i+1; j < length; j += 1)
{
/*
* Swap if currently selected array element
* is not at its correct position.
*/
if(rcv_arr[i] > rcv_arr[j])
{
temp = rcv_arr[i];
rcv_arr[i] = rcv_arr[j];
rcv_arr[j] = temp;
}
}
}

/* Print the sorted array */
printf("\nElements of array in sorted ascending order: ");
for(i = 0; i < length; i += 1)
{
printf("%d\t", rcv_arr[i]);
}

foundOrNot = binarySearch(rcv_arr,length);
if(foundOrNot == -1)
{
printf("\nElement Not found");
}

}
```

```
int seq_search(int rcv_arr[MAX],int size)
{
int no_search,pos = 0,i;
printf("\nEnter a number to be search\n");
scanf("%d",&no_search);
for(i = 0;i < size;i += 1)
{
if(rcv_arr[i] == no_search)
{
pos = i+1;
break;
}
}
if(pos!=0)
{
return pos;
}
else
{
return - 1;
}
}


int binarySearch(int arr[], int size)
{
int i, first, last, middle, search;
printf("\nEnter element to find\n");
```

```
scanf("%d", &search);

first = 0;
last = size - 1;
middle = (first+last)/2;

while (first <= last) {
if (arr[middle] < search)
first = middle + 1;
else if (arr[middle] == search) {
return middle+1;
break;
}
else
last = middle - 1;

middle = (first + last)/2;
}
if (first > last)
return -1;

}

void layout_func()
{
printf("\t\t\t ____________________________________ \n");
printf("\n\t\t\t ***** LINEAR AND BINARY SEARCH ***** \n");
printf("\t\t\t ____________________________________ \n\n\n");
printf("\t_____________________________________________________________________\n");
}
```

Output:-

```
D:\c\binary_linear.exe

                    ***********************************
                    ***** LINEAR AND BINARY SEARCH *****
                    ***********************************

Please enter your choice
1. Sequential Search
2. Binary Search
Input:  1

Enter a number to be search
2

Element found at position 4
Do you wish to continue?
Input(y/n):
```

```
D:\c\binary_linear.exe

                    ***********************************
                    ***** LINEAR AND BINARY SEARCH *****
                    ***********************************

Please enter your choice
1. Sequential Search
2. Binary Search
Input:  2

Array is not sorted, PERFORMING SORTING
Elements of array in sorted ascending order: 1  2      3      4      5
Enter element to find
2

Do you wish to continue?
Input(y/n):
```

Program Number. 2

Program Name: Implement selection and Insertion Sort Techniques

Date of Implementation: 27.11.2017

Description: A menu driven program to implement selection and insertion sort using structures.

```
#include <stdio.h>
#include <string.h>
//#define MAX_STRINGS 5
#define MAX_STRING_LEN 200
#include<stdlib.h>
#define MAX_LEN 200

char MAX_STRINGS;
void InsertionSort(char list[][MAX_STRING_LEN]);
void selectionSort(char arr[][MAX_LEN], int n);
void layout();
void thank_you();
int main()
{
int index,choice;
char strings[15][MAX_STRING_LEN];
char choice_y_n;
int flag;



/* Get input */
layout();


do{
printf("\n1.Press 1 to Enter Data:\n2.Press 2 for Insertion Sort\n3.Press 3 for Selection sort\n4.Press 4 to exit");
```

```
a:printf("\n\nEnter your choice :\t");
scanf("%d",&choice);





switch(choice)
{
case 1:
{
printf("Enter the size of Array");
scanf("%d",&MAX_STRINGS);
/*int flag=1;



while(flag==1){



if ( scanf("%d", &MAX_STRINGS) == 1 ){ // also you were missing & specifier
flag = 0;
//return 0;
}
else{
printf("-> Wrong format, try again! <-\n");
getchar(); // to catch the enter from the input -- make sure you include stdlib.h

}
}*/
printf("Enter %d strings.\n", MAX_STRINGS);
for (index = 0; index < MAX_STRINGS; index++)
{
```

```
printf("Input string %d : ", index);
scanf("%199s", strings[index]);     // limit the width so we don't go past the buffer
strings[index][sizeof(strings[index]) - 1] = '\0';
}
break;
}

case 2:
{
InsertionSort(strings);

printf("\nThe input set, in alphabetical order:\n");
for (index = 0; index < MAX_STRINGS; index++)
{
printf("%s\n", strings[index]);
}
break;
}

case 3:
{

selectionSort(strings,MAX_STRINGS);
printf("\nThe input set, in alphabetical order:\n");
for (index = 0; index < MAX_STRINGS; index++)
{
printf("%s\n", strings[index]);
}




break;
```

```
}

case 4:
{
return 0;
}




}




printf("\nDo you wish to continue?\nInput(y/n):");
getchar();
scanf("%c",&choice_y_n);
}while(choice_y_n == 'Y' || choice_y_n == 'y');




system("cls");
thank_you();
return 0;

}




void InsertionSort(char list[][MAX_STRING_LEN])
{
```

```
int i;
for ( i = 1; i < MAX_STRINGS; i++)
{
int j = i;

while (j > 0 && strcmp(list[j - 1], list[j]) > 0)
{
char tmp[MAX_STRING_LEN];
strncpy(tmp, list[j - 1], sizeof(tmp) - 1);
tmp[sizeof(tmp) - 1] = '\0';

strncpy(list[j - 1], list[j], sizeof(list[j - 1]) - 1);
list[j - 1][sizeof(list[j - 1]) - 1] = '\0';

strncpy(list[j], tmp, sizeof(list[j]));
list[j][sizeof(list[j]) - 1] = '\0';

--j;
}
}
}


void selectionSort(char arr[][MAX_LEN], int n)
{
int i, j, min_idx;

// One by one move boundary of unsorted subarray
char minStr[MAX_LEN];
for (i = 0; i < n-1; i++)
{
// Find the minimum element in unsorted array
```

```
int min_idx = i;
strcpy(minStr, arr[i]);
for (j = i+1; j < n; j++)
{
// If min is greater than arr[j]
if (strcmp(minStr, arr[j]) > 0)
{
// Make arr[j] as minStr and update min_idx
strcpy(minStr, arr[j]);
min_idx = j;
}
}

// Swap the found minimum element with the first element
if (min_idx != i)
{
char temp[MAX_LEN];
strcpy(temp, arr[i]); //swap item[pos] and item[i]
strcpy(arr[i], arr[min_idx]);
strcpy(arr[min_idx], temp);
}
}
}

void thank_you()
{
printf("\n\n_____\n\n");
printf("\n\t\tTHANK YOU FOR USING MY PROGRAM\n");
printf("\n\t Press ENter to exit");

}
void layout()
```

```
{
printf("\n_____________________________________________________________________________\n");
printf("\n \t\tSELECTION & INSERTION SORT\n");
printf("\n_____________________________________________________________________________\n\n");
}
```

Output :-

```
_____________________________________________________________________

                SELECTION & INSERTION SORT

_____________________________________________________________________


1.Press 1 to Enter Data:
2.Press 2 for Insertion Sort
3.Press 3 for Selection sort
4.Press 4 to exit

Enter your choice :     1
Enter the size of Array3
Enter 3 strings.
Input string 0 : Prakash
Input string 1 : Amal
Input string 2 : binod

Do you wish to continue?
Input(y/n):y

1.Press 1 to Enter Data:
2.Press 2 for Insertion Sort
3.Press 3 for Selection sort
4.Press 4 to exit

Enter your choice :     2

The input set, in alphabetical order:
Amal
Prakash
binod

Do you wish to continue?
Input(y/n):y
```

Program Number. 3

Program Name: Implementation of Stacks.

Date of Implementation: 06.12.2017

Description: A menu driven program to Stack operations in array and link list with necessary validations.

```
#include <stdio.h>
#include <conio.h>
#include<stdlib.h>
#include<string.h>
# define max 5
#define MAXSIZE 5


struct node
{
int data1;
struct node *link;
};

int i;

struct stack
{
   int stk[MAXSIZE];
   int top;
};
typedef struct stack STACK;
STACK s;

/******************/
char stack[max][80];
int top2;
```

```
/*************************/
void push(void);
int  pop(void);
void display(void);

/*****************************************************************************************
*****************************************************/

                                        /*Push function for linked_list operation using structure*/

/*****************************************************************************************
*****************************************************/

struct node *push1(struct node *p , int value)
{
        struct node *temp;
                                /* Allocating the dynamic size for structure */

        temp=(struct node *)malloc(sizeof(struct node));
        if(temp==NULL)
        {
                printf("No Memory available\n");
                exit(0);
        }
                temp->data1 = value;
                temp->link = p;
                p = temp;
                return(p);
        }

                        /* Pop operation using linked list */
struct node *pop1(struct node *p , int *value)
{
```

```
        struct node *temp;
        if(p==NULL)
        {
        printf(" The stack is empty and cannot Delete (POP) element\n");
    exit(0);
        }
         *value = p->data1;
        temp = p;
        p = p->link;
        free(temp);
        return(p);
        }


/*****************************************************************************************
*****************************************************/


                                                    /* PUSH OPERATION IN STACK WITH STRINGS
USING ARRAYS */


/*****************************************************************************************
*****************************************************/


int push2(char stack[max][80], int *top2, char data[80])        //taking array of the size max[5] rows for the
string
{
        if(*top2 == max -1)
```

```
                return(-1);
        else
        {
                *top2 = *top2 + 1;
                strcpy(stack[*top2], data);
                return(1);
        } // else
} // push2



/* Pop operation to extract strings from stack */

int pop2(char stack[max][80], int *top2, char data[80])
{
        if(*top2 == -1)
                return(-1);
        else
        {
                strcpy(data, stack[*top2]);
                *top2 = *top2 - 1;
                return(1);
        } // else
} // pop2



void show_str()         // functions to display all the strings in the stack
{

        if(top2>=0)
        {
    printf("\n The elements in STACK \n");
    for(i=top2; i>=0; i--)
        printf("\n%s",stack[i]);
```

```
        printf("\n Press Next Choice");
  }
  else
  {
    printf("\n The STACK is empty");
  }
}



/*****************************************************************************************
*****************************************************/

                                                        //Functions for integer push
and pop opeRATIONS
                                                        /*  Function to add an element
to the stack */

/*****************************************************************************************
*****************************************************/


void push ()
{
  int num;
  if (s.top == (MAXSIZE - 1))          //basic push operation comparing the top element with the size of stack
  {
    printf ("Stack is Full\n");
    return;
  }
  else
  {
```

```
      printf ("Enter the element to be pushed\n");
      scanf ("%d", &num);
      s.top = s.top + 1;
      s.stk[s.top] = num;
   }
   return;

}
/*  Function to delete an element from the stack */

int pop ()
{
   int num;        // num = numbers of elements in the stack
   if (s.top == - 1)                    // if top is -1 means no element is there
   {
      printf ("Stack is Empty\nElement can't be Popped ");
      return (s.top);
   }
   else
   {
      num = s.stk[s.top];
      printf ("poped element is = %dn", s.stk[s.top]);       // the element which is deleted is that which is last inserted
      s.top = s.top - 1;
   }
   return(num);
}
/*  Function to display the status of the stack */

void display ()
{
   int i;
   if (s.top == -1)           // checking the numbers of element in the stack
   {
```

```
    printf ("Stack is empty\n");
    return;      // if stack is empty then return
  }
  else
  {
    printf ("\n The status of the stack is \n");
    for (i = s.top; i >= 0; i--)
    {
      printf ("%d\n", s.stk[i]);
    }
  }
  printf ("\n");
}



/*   Layout function no logic */


void thank_you()
{
        printf("\n\n_____________________________________________________________________________ \n\n");
        printf("\n\t\tTHANK YOU FOR USING MY PROGRAM\n");
        printf("\n\t Press ENter to exit");

}
void layout()
{
        printf("\n\t_____________________________________________________________________________ \n");
        printf("\n \t\t\tIMPLEMENTATION OF STACK OPERATIONS\n");
        printf("\n\t_____________________________________________________________________________ \n\n");
}
```

```
/* Layout function ends here */



/*_____________________________________________________________________________________
_____________________________________________________________*/



/*                                                                                    MAIN
FUNCTION STARTS HERE
                                                                    */


/*_____________________________________________________________________________________
_____________________________________________________________*/



int main()
{

        int choose;                 //variable for the main switch operation

        /*****/
        char stack[max][80], data[80];  // array of characters
        int top2, option, reply;

        // Initialise Stack
        top2 = -1;
        system("cls");
        /******/

        /*integer operatiom*/
        int choice;
  int opt = 1;
```

```
s.top = -1;
//integer operation
        a: system("cls");
        layout();

        printf("\n\n\t\t1.Press 1 to Use stack Operation using Linked list\n\n\t\t2.Press 2 to use Stack Operation for
Strings.\n\n\t\t3.Press 3 to use Stack operation for Integers Using Structure\n\n\t\tPress 4 to exit :\t");
        scanf("%d",&choose);
        getchar();

        switch(choose)
        {
                case 1:
                        {
                                system("cls");
                                struct node *top1 = NULL;
                                int n,value;
                                        do
                                        {
                                                do
                                                {
                                                        printf("Enter the element to be pushed in stack\n");
                                                        scanf("%d",&value);
                                                        top1 = push1(top1,value);        //calling lined list push
function
                                                        printf("Enter 1 to continue\n");
                                                        scanf("%d",&n);
                                                } while(n == 1);

                                                printf("Enter 1 to pop an element from stack\n");
                                                scanf("%d",&n);
                                                while( n == 1)
                                                {
```

```
                                        top1 = pop1(top1,&value);          //calling pop
function
                                        printf("The value poped is %d\n",value);
                                        printf("Enter 1 to pop an element and other to
push1\n");
                                        scanf("%d",&n);
                                }
                                printf("Enter 1 to continue\n");
                                scanf("%d",&n);
                                } while(n == 1);

                        break;



                }

        case 2:
                {
                        do
                        {
                                //printf("\n C Language to implement basic stack operations for String
based Stack \n");
                                //system("cls");
                                printf("\n 1. Press 1 to PUSH String ");
                                printf("\n 2. Press 2 to POP String ");
                                printf("\n 3. Press 3 to Display Stack Elements ");
                                printf("\n 3. Press 4 For Main Menu");
                                printf("\n Press 4 to EXIT the program");
                                printf("\n Select proper option ( 1 / 2 / 3 ) : ");
                                scanf("%d", &option);
                                switch(option)
                                {
                                        case 1 : // push2
                                                printf("\n Enter a String : ");
```

```
                    getchar();
                    gets(data);

                    reply = push2(stack,  &top2, data);
                    if(reply == -1)
                        printf("\nStack is Full \nThe String %s can't  be entered
into Stack \n",data);
                    else
                        printf("\n Entered String %s Pushed in Stack \n",data);

                    break;

                case 2 : // pop2
                    reply = pop2(stack,  &top2, data);
                    if(reply == -1)
                        printf("\n Stack is Empty \nNo elements Remaining in
the stack");
                    else
                        printf("\n Popped String is : %s", data);
                        printf("\n");
                        break;

                case 3 :
                    {
                        if(top2>=0)
                        {
                        printf("\n The elements in STACK \n");
                    for(i=top2; i>=0; i--)
                    {

                    printf("\n\t\t%s",stack[i]);
                    }
                    printf("\n\n\n");
                    }
```

```
                    else
                    {
                    printf("\n The STACK is empty");
                    }

                        break;

                    }
                case 4 :
                    {
                        goto a;
                    }

                case 5:
                    {
                        thank_you();
                        return 0;
                    }
            } // switch
        }while(1);



    }

    /* case 3 calls the function for the stack opeartions of integers */
case 3:
    {
        system("cls");
        printf ("STACK OPERATION\n");

        while (opt)
    {
    printf ("----------------------------------------\n");
```

```
printf ("     1   -->   PUSH            \n");
printf ("     2   -->   POP             \n");
printf ("     3   -->   DISPLAY            \n");
printf ("     4   -->   EXIT         \n");
printf ("----------------------------------------\n");

printf ("Enter your choice\n");
scanf    ("%d", &choice);
            switch (choice)
{
            case 1:
            push();
            break;

                        case 2:
            pop();
            break;

                        case 3:
            display();
            break;

                        case 4:
            return;
}
fflush (stdin);
printf ("Do you want to continue(Type 0 or 1)?\n");
             scanf    ("%d", &opt);
            }



}
```

```
            default:
            {
                return 0;
            }
        }

        return 0;
}
```

Output :-

```
D:\c\stack_all.exe

        ____________________________________________________________

                        IMPLEMENTATION OF STACK OPERATIONS

        ____________________________________________________________


                1.Press 1 to Use stack Operation using Linked list

                2.Press 2 to use Stack Operation for Strings.

                3.Press 3 to use Stack operation for Integers Using Structure

                Press 4 to exit :       2

1. Press 1 to PUSH String
2. Press 2 to POP String
3. Press 3 to Display Stack Elements
3. Press 4 For Main Menu
Press 4 to EXIT the program
Select proper option ( 1 / 2 / 3 ) : 1

Enter a String : pk

Entered String pk Pushed in Stack
```

Program Number. 4

Program Name: Implementation of various queue operations.

Date of Implementation: 08.01.2018

Description: A menu driven program to Queue operations in array and link list with necessary validations.

```
/*
 * C Program to Implement Queue Data Structure using Linked List
 */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include<conio.h>
#define max 5
#define MODE_INT 1
#define MODE_STR 2
#include<ctype.h>

struct node
{
   int info;
   struct node *ptr;
}*front,*rear,*temp,*front1;





int i;



int choice;
```

```
int frontelement();
void enq(int data);
void deq();
void empty();
void display();
void create();
void queuesize();

int validate(char*, int);

int count = 0;

void main()
{
        a:
    system("cls");

    printf("\n_____________________________________________________________________
__________\n");
    printf("\n\t\tQUEUE");
    printf("\n_____________________________________________________________________
_____________\n");



    printf("\n1.Press 1 For Link-List Implementation");
    printf("\n2.Press 2 for Array Implementation for Strings");
    printf("\nEnter Your Choice :\t");
    scanf("%d",&choice);

    switch(choice)
    {
        case 1:
            {
```

```
        int no, ch, e;
        b:
        system("cls");
printf("\n 1 - Enter data into queue ");
printf("\n 2 - Delete element from queue");
printf("\n 3 - Show the Front element");
printf("\n 4 - Show the Last Element ");
printf("\n 5 - Empty the queue");
        printf("\n 6 - Exit Program");
printf("\n 7 - Display Queue Elements");
printf("\n 8 - Display Queue size");
printf("\n 9 - Back to Previous Menu");
create();
while (1)
{
printf("\n Enter choice :");
scanf("%d", &ch);
switch (ch)
{
        case 1:
                printf("\nEnter data \n(ENTER INTEGERS ONLY) :\t ");
                scanf("%d", &no);




                enq(no);
                break;
        case 2:
                deq();
                break;
                case 3:
                e = frontelement();
```

```
            if (e != 0)
            printf("Front element : %d", e);
            else
            printf("\n No front element in Queue as queue is empty");
                break;

    case 4:
                {
                        e = rearelement();
            if (e != 0)
            printf("Front element : %d", e);
            else
            printf("\n No front element in Queue as queue is empty");
                break;
                                }

    case 5:
            empty();
            break;
    case 6:
            exit(0);
    case 7:
            display();
            break;
    case 8:
            queuesize();
            break;

    case 9:
            goto a;
            break;
    default:
```

```
                    printf("Wrong choice, Please enter correct choice  ");
                    getchar();
                    goto b;
                    break;
            }
            }
            }

            case 2:

            {
                    char queue[max][80], data[80];
int frontq, rearq , reply, option,i;
int status,temp;
//... Initialise a Queue
frontq = rearq = -1;
do
{
        printf("\n Queue using array of strings \n");
        printf("\n 1. Press 1 to Enter String in a Queue");
        printf("\n 2. Press 2 to Delete String from a Queue");
        printf("\n 3. Display the queue Elements");
        printf("\n 4. Press 4 to back to previous menu");
        printf("\n 5. Exit ");
        printf("\nEnter your choice: ");
        //getchar();
        status = scanf("%d", &option);
        while(status != 1)
        {
                while((temp=getchar()) != EOF && temp != '\n');
                printf("Invalid input...\nPlease enter a number: ");
                status = scanf("%d", &option);
```

```
        }
        switch(option)
        {
                case 1 : // insert
                //char input[10];
                INPUT:
                        printf("\n Enter the String to be insert in a Queue : ");
                        getchar();
                        //scanf("%s", data);
                        //printf("Enter number\n> ");
                        fgets(data, 10, stdin);
                        data[strcspn(data,"\n")] = 0;

                        if (validate(data, MODE_STR)) {
                                //printf("Valid.\n");

                                reply = insq(queue, &rearq, data);
                                if( reply == -1 )
                                printf("\n Queue is Full \n");
                                else
                                        printf("\n Entered String : %s is Inserted in a Queue \n",data);
                                        break;
                        }
                        else {
                        printf("Invalid input.\n");
                        goto INPUT;
                        }


                case 2 : // delete
                        reply = delq(queue, &frontq, &rearq, data);
                        if( reply == -1 )
```

```
                    printf("\n Queue is Empty \n");
            else
                    printf("\n Deleted String from Queue is : %s", data);
                    printf("\n");
            break;
        case 3 : if (reply == - 1)
            printf("Queue is empty \n");
            else
              {
                printf("Queue is : \n");
                for (i = frontq; i < rearq; i++)
                    printf("\n %s ", queue[i]);
                printf("\n");
              }
                    break;

        case 4:
            goto a;
    } // switch
}while(option != 0);

        }


        case 3:
            {
                    exit(0);
            }

        default :
            {
                    printf("\nWrong Choice Please Enter Again");
```

```
                    getchar();
                    goto a;
                    break;
                }
            }
}
/* Create an empty queue */
void create()
{
   front = rear = NULL;
}

/* Returns queue size */
void queuesize()
{
   printf("\n Queue size : %d", count);
}

/* Enqueing the queue */
void enq(int data)
{
   if (rear == NULL)
   {
      rear = (struct node *)malloc(1*sizeof(struct node));
      rear->ptr = NULL;
      rear->info = data;
      front = rear;
   }
   else
   {
      temp=(struct node *)malloc(1*sizeof(struct node));
      rear->ptr = temp;
```

```
        temp->info = data;
        temp->ptr = NULL;

        rear = temp;
    }
    count++;
}

/* Displaying the queue elements */
void display()
{
    front1 = front;

    if ((front1 == NULL) && (rear == NULL))
    {
        printf("Queue is empty");
        return;
    }
    while (front1 != rear)
    {
        printf("%d ", front1->info);
        front1 = front1->ptr;
    }
    if (front1 == rear)
        printf("%d", front1->info);
}

/* Dequeing the queue */
void deq()
{
    front1 = front;
```

```
    if (front1 == NULL)
    {
        printf("\n Error: Trying to display elements from empty queue");
        return;
    }
    else
        if (front1->ptr != NULL)
        {
            front1 = front1->ptr;
            printf("\n Dequed value : %d", front->info);
            free(front);
            front = front1;
        }
        else
        {
            printf("\n Dequed value : %d", front->info);
            free(front);
            front = NULL;
            rear = NULL;
        }
        count--;
}

/* Returns the front element of queue */
int frontelement()
{
    if ((front != NULL) && (rear != NULL))
        return(front->info);
    else
        return 0;
}
```

```
int rearelement()
{

        if ((front != NULL) && (rear != NULL))
      return(rear->info);
   else
      return 0;

}

/* Display if queue is empty or not */
void empty()
{
    if ((front == NULL) && (rear == NULL))
      printf("\n Queue empty");
   else
     printf("Queue not empty");
}


int insq(char queue[max][80], int *rearq, char data[80])
{
        if(*rearq == max -1)
                return(-1);
        else
        {
                *rearq = *rearq + 1;
                strcpy(queue[*rearq], data);
```

```
                return(1);
        } // else
} // insq
int delq(char queue[max][80], int *frontq, int *rearq, char data[80])
{
        if(*frontq == *rearq)
                return(-1);
        else
        {
                (*frontq)++;
                strcpy(data, queue[*frontq]);
                return(1);
        } // else
} // delq

/*void queueshow()
{
        int reply;
                if (reply == - 1)
                                        printf("Queue is empty \n");
                                        else
                                        {
                                        printf("Queue is : \n");
                                        for (i = frontq; i < rearq; i++)
                                                printf("%s ", queue[i]);
                                        printf("\n");
                                        }



}*/
```

```
int validate(char *cptr, int mode) {
        int len = strlen(cptr);
        int i;
        for (i = 0; i < len; i++) {
                if (mode == MODE_INT) {
                        if (!(isdigit(cptr[i]) || cptr[i] == '-')) {
                                return 0;
                        }
                } else {
                        if (!isalpha(cptr[i])) {
                                return 0;
                        }
                }
        }
        return 1;
}
```

Output :-

```
D:\c\queue.exe

                    QUEUE

1.Press 1 For Link-List Implementation
2.Press 2 for Array Implementation for Strings
Enter Your Choice :     2

 Queue using array of strings

 1. Press 1 to Enter String in a Queue
 2. Press 2 to Delete String from a Queue
 3. Display the queue Elements
 4. Press 4 to back to previous menu
 5. Exit
Enter your choice: 1

 Enter the String to be insert in a Queue : prakash

 Entered String : prakash is Inserted in a Queue
```

Program Number. 5

Program Name: Implementation of link list and some operations on link list.

Date of Implementation: 08.01.2018

Description: A menu driven program for link list operations with necessary validations.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define MAX 30

struct pass_data
{
   int  passno;
   char passName[MAX];
   char destination[MAX];
   struct pass_data *next;
};

/* ******************************************************************/
/*  Function to insert a node at the front of the linked list.       */
/*  front: front pointer, id: passenger I444444D, name: passenger name       */
/*  desg: passenger destination                                      */
/*  Returns the new front pointer.                                   */
/* ******************************************************************/
struct pass_data *insert(struct pass_data *front, int id, char name[],
char desg[])
{
   struct pass_data *newnode;

   newnode = (struct pass_data*)malloc(sizeof(struct pass_data));
```

```
    if (newnode == NULL)
    {
        printf("\n Allocation failed \n");
        exit(2);
    }
    newnode->passno = id;
    strcpy(newnode->passName, name);
    strcpy(newnode->destination, desg);
    newnode->next = front;
    front = newnode;
    return(front);
}
/*  End of insert() */

/*  Function to display a node in a linked list */
void printNode(struct pass_data *p)
{
    printf("\n passenger Details...\n");
    printf("\n Emp No       : %d", p->passno);
    printf("\n Name         : %s", p->passName);
    printf("\n destination    : %s\n", p->destination);
    printf("-------------------------------------\n");
}
/*  End of printNode() */

/*  ********************************************************/
/*  Function to deletpnode a node based on passenger number */
/*  front: front pointer, id: Key value                    */
/*  Returns: the modified list.                            */
/*  ********************************************************/
struct pass_data* deletpnode(struct pass_data *front, int id)
{
```

```
   struct pass_data *ptr;
   struct pass_data *bptr;

   if (front->passno == id)
   {
      ptr = front;
      printf("\n Node deleted:");
      printNode(front);
      front = front->next;
      free(ptr);
      return(front);
   }
   for (ptr = front->next, bptr = front; ptr != NULL; ptr = ptr->next,
bptr = bptr->next)
   {
      if (ptr->passno == id)
      {
         printf("\n Node deleted:");
         printNode(ptr);
         bptr->next = ptr->next;
         free(ptr);
         return(front);
      }
   }
   printf("\n passenger Number %d not found ", id);
   return(front);
}
/*  End of deletpnode() */

/* ***************************************************************/
/*  Function to search the nodes in a linear fashion based emp ID */
/*  front: front pointer, key: key ID.                          */
```

```
/* ************************************************************/
void search(struct pass_data *front, int key)
{
   struct pass_data *ptr;

   for (ptr = front; ptr != NULL; ptr = ptr -> next)
   {
      if (ptr->passno == key)
      {
         printf("\n Key found:");
         printNode(ptr);
         return;
      }
   }
   printf("\n passenger Number %d not found ", key);
}
/*  End of search() */

/*  Function to display the linked list */
void display(struct pass_data  *front)
{
   struct pass_data *ptr;

   for (ptr = front; ptr != NULL; ptr = ptr->next)
   {
      printNode(ptr);
   }
}
/*  End of display() */

/*  Function to display the menu of operations on a linked list */
void menu()
```

```
{
   printf("---------------------------------------------\n");
   printf("Press 1 to CREATE a reservation in the list      \n");
   printf("Press 2 to DELETE a reservation from the list      \n");
   printf("Press 3 to DISPLAY the reservation list              \n");
   printf("Press 4 to SEARCH the reservation list                \n");
   printf("Press 5 to EXIT                          \n");
   printf("---------------------------------------------\n");
}
/*  End of menu() */

/*  Function to select the option */
char option()
{
   int choice;

   printf("\n\n>> Enter your choice:  ");
   switch(choice=getch())
   {
      case '1':
      case '2':
      case '3':
      case '4':
      case '5':   return(choice);
      default :   printf("\n Invalid choice.");
   }
   return choice;
}
/*  End of option() */

/*  The main() program begins */
void main()
```

```
{
    struct pass_data *linkList;
    char name[21], dest[51];
    char choice;
    int pno;

    linkList = NULL;
    printf("\n Welcome to BMSRT Reservation System \n");
    menu();
    do
    {
        /*  choose oeration to be performed */
        choice = option();
        switch(choice)
        {
        case '1':
            printf("\n Enter the passenger Number      : ");
            scanf("%d", &pno);
            printf("Enter the passenger name        : ");
            fflush(stdin);
            gets(name);
            printf("Enter the passenger destination : ");
            gets(dest);
            linkList = insert(linkList, pno, name, dest);
            break;
        case '2':
            printf("\n\n Enter the passenger number to be cancel the reservation: ");
            scanf("%d", &pno);
            linkList = deletpnode(linkList, pno);
            break;
        case '3':
            if (linkList == NULL)
```

```
        {
            printf("\n There are no resrvations currently....");
            break;
        }
        display(linkList);
        break;
    case '4':
        printf("\n\n Enter the passenger number to be searched: ");
        scanf("%d", &pno);
        search(linkList, pno);
        break;
    case '5': break;
    }
  } while (choice != '5');
}
-
```

Output :-

```
D:\c\link_str.exe

 Welcome to BMSRT Reservation System
---------------------------------------------
Press 1 to CREATE a reservation in the list
Press 2 to DELETE a reservation from the list
Press 3 to DISPLAY the reservation list
Press 4 to SEARCH the reservation list
Press 5 to EXIT
---------------------------------------------


>> Enter your choice:
 Enter the passenger Number      :
1745
Enter the passenger name        : prakash
Enter the passenger destination : banglore


>> Enter your choice:
 passenger Details...

 Emp No       : 1745
 Name           : prakash
 destination    : banglore
-------------------------------------


>> Enter your choice:
```

Program Number. 6

Program Name: Implementation of Quick sort

Date of Implementation: 24.01.2018

Description: A menu driven program for Quick sort using link list and array with necessary validations.

```
#define MAX 20
#define SIZE 10

#include<stdio.h>
#include<conio.h>
#include<string.h>
#include <iostream>
#include <cstdio>




using namespace std;

                    /* a node of the singly linked list */
                    struct Node
                    {
                    int data;
                    struct Node *next;
                    };

/* A utility function to insert a node at the beginning of linked list */
                    void push(struct Node** head_ref, int new_data)
                    {
  /* allocate node */
                              struct Node* new_node = new Node;

  /* put in the data  */
```

```
                    new_node->data  = new_data;

    /* link the old list off the new node */
                    new_node->next = (*head_ref);

    /* move the head to point to the new node */
                    (*head_ref)    = new_node;
                    }

/* A utility function to print linked list */
                    void printList(struct Node *node)
                    {
                while (node != NULL)
                    {
                    printf("%d  ", node->data);
                    node = node->next;
                        }
                        printf("\n");
                    }

// Returns the last node of the list
struct Node *getTail(struct Node *cur)
{
    while (cur != NULL && cur->next != NULL)
        cur = cur->next;
    return cur;
}

// Partitions the list taking the last element as the pivot
struct Node *partition(struct Node *head, struct Node *end,
                struct Node **newHead, struct Node **newEnd)
{
```

```
struct Node *pivot = end;
struct Node *prev = NULL, *cur = head, *tail = pivot;

// During partition, both the head and end of the list might change
// which is updated in the newHead and newEnd variables
while (cur != pivot)
{
    if (cur->data < pivot->data)
    {
        // First node that has a value less than the pivot - becomes
        // the new head
        if ((*newHead) == NULL)
            (*newHead) = cur;

        prev = cur;
        cur = cur->next;
    }
    else // If cur node is greater than pivot
    {
        // Move cur node to next of tail, and change tail
        if (prev)
            prev->next = cur->next;
        struct Node *tmp = cur->next;
        cur->next = NULL;
        tail->next = cur;
        tail = cur;
        cur = tmp;
    }
}

// If the pivot data is the smallest element in the current list,
// pivot becomes the head
```

```
    if ((*newHead) == NULL)
        (*newHead) = pivot;

    // Update newEnd to the current last node
    (*newEnd) = tail;

    // Return the pivot node
    return pivot;
}



//here the sorting happens exclusive of the end node
struct Node *quickSortRecur(struct Node *head, struct Node *end)
{
    // base condition
    if (!head || head == end)
        return head;

    Node *newHead = NULL, *newEnd = NULL;

    // Partition the list, newHead and newEnd will be updated
    // by the partition function
    struct Node *pivot = partition(head, end, &newHead, &newEnd);

    // If pivot is the smallest element - no need to recur for
    // the left part.
    if (newHead != pivot)
    {
        // Set the node before the pivot node as NULL
        struct Node *tmp = newHead;
        while (tmp->next != pivot)
            tmp = tmp->next;
```

```
        tmp->next = NULL;

        // Recur for the list before pivot
        newHead = quickSortRecur(newHead, tmp);

        // Change next of last node of the left half to pivot
        tmp = getTail(newHead);
        tmp->next =  pivot;
    }

    // Recur for the list after the pivot element
    pivot->next = quickSortRecur(pivot->next, newEnd);

    return newHead;
}0

// The main function for quick sort. This is a wrapper over recursive
// function quickSortRecur()
void quickSort(struct Node **headRef)
{
    (*headRef) = quickSortRecur(*headRef, getTail(*headRef));
    return;
}

// Driver program to test above functions



    struct Node *a = NULL;

    int data;
    char choice_y_n;
```

```
/*****************************************************************************************
********/

typedef struct
{
        char student_name[MAX];
        int yearofadmsn;
        char fname[MAX];
   //int std_count;
        //int student[SIZE];
        int marks;
} student;


void sel_name_sort(student [],int );
void sel_price_sort(student [],int );
void ins_name_sort(student [],int );
void ins_price_sort(student [],int );
void studentfn();
void layout();
void thank_you();


int main()
{
        printf("=============================================Implementation of
structures in Quick Sort=============================================");
        layout();
        printf("\n");

        student std_array[SIZE];
```

```
int index,ch,choice;

int std_count;

int opt;

printf("\n\n1.Press 1 for Array Implementation\n2.Press 2 for Link List Implementation\n\tEnter Choice :\t");
scanf("%d",&opt);

switch(opt)
{

    case 1:{

                printf("\nHow Many Records You want to Enter :\t");
                scanf("%d",&std_count);
                getchar();

                system("cls");
                studentfn();
                printf("Enter Some Details : \t");
                for(index=0;index<std_count;index+=1)
                    {
                    printf("\n Student's name: ");
                    scanf("%s",std_array[index].student_name);

        //printf("\nFather's Name : ");
```

```
//scanf("%s",std_array[index].fname);

                                        printf("\nMarks Obtained : ");
                                        scanf("%d",&(std_array[index].marks));

//printf("\n Year of dmission : ");
//scanf("%d",&(std_array[index].yearofadmsn));

                                        printf("\n");
                                        }

                                        printf("\n The array entered is");
                                        for(index=0;index<std_count;index+=1)
                                        {
                                                printf("\n Student's name : ");
                                                printf("%s",std_array[index].student_name);
                                                printf("\t");



        //printf("Father's Name': ");
        //printf("%s",std_array[index].fname);
        //printf("\t");

                                                printf("Marks Obtained: ");
                                                printf("%d",std_array[index].marks);
                                                printf("\t");

        //printf("Year of admission: ");
        //printf("%d",std_array[index].yearofadmsn);
        //printf("\n");
                                        }
```

```
do
{       system("cls");
        printf("\n MAIN MENU");
        printf("\n 1. QUICK  SORT");
        printf("\n 2. MERGE SORT");
        printf("\n 3.exit");
        printf("\n Enter your choice:");
        scanf("%d",&choice);

        switch(choice)
        {
                case 1:
                        system("cls");

                printf("\n Selection sort chosen");
            do
        {
            printf("\n QUICK SORT MENU");
            printf("\n 1.sort by Student's name");
            printf("\n 2.sort by Student's Marks");
            printf("\n 3.exit");
            printf("\n Enter your choice:");
            scanf("%d",&ch);

            switch(ch)
            {
                case 1: printf("\n Sorting by
Student's name ");

    sel_name_sort(std_array,std_count);
                printf("\n\n");
                break;
```

```
                                                    case 2: printf("\n Sorting by
Student's Marks ");
                                                    sel_price_sort(std_array,std_count);
                                                    printf("\n\n");
                                                        break;

                                            default: printf("\n Wrong choice ");
                                                    break;
                                            }
                                    }
                                    while(ch!=3);

                                    break;

                                    case 2:{
                                            printf("\n\nWork in Progress-
.......................");
                                            break;
                                        }

                                    default :{
                                                    printf("\n !!!!Wrong choice!!!!!");
                                                    break;
                                                        }

                                    }
                            }
                            while(choice != 3);
                            break;

                    }
```

```
            case 2:{



do{

            printf("\nEnter Element in to the LInk List :\t");
            scanf("%d",&data);
            push(&a, data);



            printf("\nDo you wish to continue?\nInput(y/n):");
            getchar();
            scanf("%c",&choice_y_n);
}while(choice_y_n == 'y' || choice_y_n == 'Y');

      /* push(&a, 20);
push(&a, 4);
push(&a, 3);
push(&a, 30);*/

printf( "Linked List before sorting \n");
printList(a);

quickSort(&a);

cout << "Linked List after sorting \n";
printList(a);
```

```
    return 0;
                    break;
            }
        }
}

//void quick_name_sort()

void sel_name_sort(student b_array[],int std_count)
{
        int i , j , min , index ;
        student temp;
        for( i = 0 ; i <  std_count - 1 ; i++ )
        {
                min = i;
                for( j = i + 1 ; j < std_count ; j++ )
                {
                        if( strcmp( b_array[j].student_name , b_array[min].student_name ) < 0 )
                        {
                                min = j;
                        }
                }
```

```
            if( min != i )
            {
                temp = b_array[i];
                b_array[i] = b_array[min];
                b_array[min] = temp;
            }
        }

        printf("\n The array sorted is");
        printf("\n Student's name\t\t Marks Obtained\t");
            for(index=0;index<std_count;index+=1)
        {

            printf("\n\n%s",b_array[index].student_name);
            printf("\t\t\t");

            //printf("Marks Obtained ");
            printf("%d",b_array[index].marks);
            printf("\t");

            //printf("Father's Name' ");
            //printf("%s",b_array[index].fname);
            //printf("\t");

            //printf("Year of admission ");
            //printf("\t%d",b_array[index].yearofadmsn);
            //printf("\n");
        }
}

void sel_price_sort(student b_array[],int std_count)
{
```

```
int i , j , min , index ;
student temp;
for( i = 0 ; i <  std_count - 1 ; i++ )
{
        min = i;
        for( j = i + 1 ; j < std_count ; j++ )
        {
                if( b_array[j].marks < b_array[min].marks )
                {
                        min = j;
                }
        }
        if( min != i )
        {
                temp = b_array[i];
                b_array[i] = b_array[min];
                b_array[min] = temp;
        }
}

printf("\n The array sorted is ");
printf("\n Student's name\t\tMarks Obtained");
        for(index=0;index<std_count;index+=1)
{

        printf("\n\n%s",b_array[index].student_name);
        printf("\t\t\t");




        //printf("Father's Name' " );
        //printf("\t%s",b_array[index].fname);
```

```
                //printf("\t");

                //printf("Marks Obtained ");
                printf("\t%d",b_array[index].marks);
                printf("\t");

                //printf("Year of Admission :  ");
                //printf("\t%d",b_array[index].yearofadmsn);
                //nbmnbvprintf("\n");
        }
}

void ins_name_sort(student b_arrayins[],int std_count)
{
        int i , j ;
        student key;

        for(i = 1; i < std_count; i+=1)
        {
                key = b_arrayins[i];
                j = i - 1;

                while( (strcmp(b_arrayins[j].student_name , key.student_name) > 0) && j >= 0)
                {
                        b_arrayins[j+1] = b_arrayins[j];
                        j = j-1 ;
                }

                b_arrayins[j+1] = key;

        }
```

```
        printf("\n The array sorted is");
                for(i=0;i<std_count;i+=1)
        {
                printf("\n Student's name : ");
                printf("%s",b_arrayins[i].student_name);
                printf("\t");

                printf("Student's price : ");
                printf("%d",b_arrayins[i].marks);
                printf("\t");

                printf("Student's fname : ");
                printf("%s",b_arrayins[i].fname);
                printf("\t");

                printf("Year published : ");
                printf("%d",b_arrayins[i].yearofadmsn);
                printf("\n");
        }
}

void ins_price_sort(student b_arrayins[],int std_count)
{
        int i , j ;
        student key;

        for(i = 1; i < std_count; i+=1)
        {
                key = b_arrayins[i];
                j = i - 1;

                while(j >= 0 && b_arrayins[j].marks > key.marks)
```

```
                {
                        b_arrayins[j+1] = b_arrayins[j];
                        j = j-1 ;
                }

                b_arrayins[j+1] = key;
    }

        printf("\n The array sorted is");
                for(i=0;i<std_count;i+=1)
        {
                printf("\n Student's name : ");
                printf("%s",b_arrayins[i].student_name);
                printf("\t");

                printf("Student's price : ");
                printf("%d",b_arrayins[i].marks);
                printf("\t");

                printf("Father's Name : '");
                printf("%s",b_arrayins[i].fname);
                printf("\t");

                printf("Year of Admission : ");
                printf("%d",b_arrayins[i].yearofadmsn);
                printf("\n");
        }
}


void thank_you()
{
```

```
        printf("\n\n_____________________________________________________________________\n\
n");

        printf("\n\t\tTHANK YOU FOR USING MY PROGRAM\n");

        printf("\n\t Press ENter to exit");


}

void layout()

{

        printf("\n_____________________________________________________________________\n")
;

        printf("\n \t\tQUICK & MERGE SORT IMPLEMENTATION\n");

        printf("\n_____________________________________________________________________\n\
n");

}

void studentfn()

{

        printf("\n\n*********************************************************************
**************\n");

        printf("\n\n");

        printf("\t\tSTUDENT MANAGEMENT SYSTEM");

        printf("\n\n");

        printf("\n\n");

        printf("\n\n*********************************************************************
**************\n");


}
```

Output :-

```
==============================================Implementation of structures in Quick Sort===
==============
_____________________________________________________________________

                QUICK & MERGE SORT IMPLEMENTATION

_____________________________________________________________________



1.Press 1 for Array Implementation
2.Press 2 for Link List Implementation
        Enter Choice :  2

Enter Element in to the LInk List :     54545

Do you wish to continue?
Input(y/n):y

Enter Element in to the LInk List :     564654

Do you wish to continue?
Input(y/n):y

Enter Element in to the LInk List :     656565

Do you wish to continue?
Input(y/n):n
```

Program Number. 7

Program Name: Implementation of Merge Sort

Date of Implementation: 14.02.2018

Description: - A menu driven program for implementation of Merge sort using link list and array with necessary validations.

```
#define MAX 20
#define SIZE 10

#include<stdio.h>
#include<conio.h>
#include<string.h>
#include <iostream>
#include <cstdio>
#include<stdlib.h>





int a[50];
int b[10];

void merging(int low, int mid, int high)
{
int l1, l2, i;

for(l1 = low, l2 = mid + 1, i = low; l1 <= mid && l2 <= high; i++)
{
if(a[l1] <= a[l2])
b[i] = a[l1++];
else
b[i] = a[l2++];
```

```
}

while(l1 <= mid)
b[i++] = a[l1++];

while(l2 <= high)
b[i++] = a[l2++];

for(i = low; i <= high; i++)
a[i] = b[i];
}

void sort(int low, int high)
{
int mid;

if(low < high)
{
mid = (low + high) / 2;
sort(low, mid);
sort(mid+1, high);
merging(low, mid, high);
}
else
{
return;
}
}
```

```
struct node
```

```
{
int data;
struct node* next;
};

struct node* sortedmerge(struct node* a, struct node* b);
void frontbacksplit(struct node* source, struct node** frontRef, struct node** backRef);



void mergesort(struct node** headRef)
{
struct node* head = *headRef;
struct node* a;
struct node* b;
if ((head == NULL) || (head -> next == NULL))
{
return;
}
frontbacksplit(head, &a, &b);
mergesort(&a);
mergesort(&b);
*headRef = sortedmerge(a, b);
}

struct node* sortedmerge(struct node* a, struct node* b)
{
struct node* result = NULL;

if (a == NULL)
return(b);
else if (b == NULL)
return(a);
```

```
if ( a-> data <= b -> data)
{
result = a;
result -> next = sortedmerge(a -> next, b);
}
else
{
result = b;
result -> next = sortedmerge(a, b -> next);
}
return(result);
}

void frontbacksplit(struct node* source, struct node** frontRef, struct node** backRef)
{
struct node* fast;
struct node* slow;
if (source==NULL || source->next==NULL)
{
*frontRef = source;
*backRef = NULL;
}
else
{
slow = source;
fast = source -> next;
while (fast != NULL)
{
fast = fast -> next;
if (fast != NULL)
{
slow = slow -> next;
fast = fast -> next;
```

```
}
}

*frontRef = source;
*backRef = slow -> next;
slow -> next = NULL;
}
}

void printlist(struct node *node)
{
while(node != NULL)
{
printf("%d\t", node -> data);
node = node -> next;
}
}

void push(struct node** head_ref, int new_data)
{
struct node* new_node = (struct node*) malloc(sizeof(struct node));
new_node -> data  = new_data;
new_node->next = (*head_ref);
(*head_ref) = new_node;
}





void studentfn();
void layout();
void thank_you();
```

```
int main()
{

a:

layout();
printf("\n");

//student std_array[SIZE];

int index,ch,choice;
int choice_y_n;
int std_count;

int opt;



system("cls");
printf("\n\n1.Press 1 for Array Implementation\n2.Press 2 for Link List Implementation\n\tEnter Choice :\t");
scanf("%d",&opt);

switch(opt)
{



case 1:{
do{
system("cls");
int i,count;
printf("\nHow many elements You want to enter \t:");
scanf("%d", &count);
```

```
printf("\nEnter %d Elements: ",count);
for(i=0;i<count;i++)
scanf("%d", &a[i]);
printf("List before sorting\n");

for(i = 0; i <count; i++)
printf("%d ", a[i]);
sort(0, count-1);

printf("\nList after sorting\n");

for(i = 0; i <count; i++)
printf("%d ", a[i]);

printf("\nDo you wish to continue ?\nInput(y/n):");
getchar();
scanf("%c",&choice_y_n);
}while(choice_y_n == 'y' || choice_y_n == 'Y');

goto a;
break;

}



case 2:{



do{
```

```
system("cls");

struct node* a = NULL;
int i,n,arr[20];

printf("\n\t\t\tImplementation of Mergesort with LInked List \n\n ");
printf("\n How many Elements you want to Enter :\t\n");
scanf(" %d",&n);

printf("\n Enter %d Elements in the List \n",n);
for(i=0;i<n;i++)
{
scanf(" %d",&arr[i]);
}

for(i=0;i<n;i++)
{
push(&a,arr[i]);
}


printf("\n\t Linked List before sorting\n\n");

for(i=0;i<n;i++)
{
printf("%d\t",arr[i]);
}
printf("\n");

mergesort(&a);
```

```
printf("\n\n\t Linked List after sorting\n\n");
printlist(a);



printf("\nDo you wish to continue ?\nInput(y/n):");
getchar();
scanf("%c",&choice_y_n);
}while(choice_y_n == 'y' || choice_y_n == 'Y');

goto a;
break;
}
}
return 0;
}



void thank_you()
{
printf("\n\n_____ \n\n");
printf("\n\t\tTHANK YOU FOR USING MY PROGRAM\n");
printf("\n\t Press ENter to exit");

}
void layout()
{
printf("\n_____ \n");
printf("\n \t\tMERGE SORT IMPLEMENTATION WITH ARRAY AND LINK LIST \n");
printf("\n_____ \n\n");
}
void studentfn()
{
printf("\n\n*********************************************\n");
```

```
printf("\n\n");

printf("\t\tSTUDENT MANAGEMENT SYSTEM");

printf("\n\n");

printf("\n\n");

printf("\n\n*****************************************************************************************\
n");


}
```

Output :-

```
                    Implementation of Mergesort with LInked List

 How many Elements you want to Enter :
5

 Enter 5 Elements in the List
9
7
3
1
2

         Linked List before sorting

9       7       3       1       2


         Linked List after sorting

1       2       3       7       9
Do you wish to continue ?
Input(y/n):
```

Program Number. 8

Program Name: Implementation of String Matching Algorithm.

Date of Implementation: 14.02.2018

Description: - A menu driven program for implementation of KMP String matching Algorithm.

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#include<string.h>
#include<bits/stdc++.h>

void computeLPSArray(char *pat, int M, int *lps);

// Prints occurrences of txt[] in pat[]
void KMPSearch(char *pat, char *txt)
{
int M = strlen(pat);
int N = strlen(txt);

// create lps[] that will hold the longest prefix suffix
// values for pattern
int lps[M];

// Preprocess the pattern (calculate lps[] array)
computeLPSArray(pat, M, lps);

int i = 0;  // index for txt[]
int j  = 0;  // index for pat[]
while (i < N)
{
if (pat[j] == txt[i])
{
```

```
j++;
i++;
}

if (j == M)
{
printf("\n\n\t\t\tFound - ( %s ) - pattern at index %d ",pat, (i-j));
j = lps[j-1];



}



else if (i < N && pat[j] != txt[i])
{

if (j != 0)
j = lps[j-1];
else
i = i+1;
}



}
if(j != M)
{
printf("\n\n\t\t\t********\tThere is no More Match for the Given String\t*********");
//break;
}
```

```
}



void computeLPSArray(char *pat, int M, int *lps)
{

int len = 0;

lps[0] = 0; // lps[0] is always 0



int i = 1;
while (i < M)
{
if (pat[i] == pat[len])
{
len++;
lps[i] = len;
i++;
}
else // (pat[i] != pat[len])
{

if (len != 0)
{
len = lps[len-1];

// Also, note that we do not increment
// i here
}
else // if (len == 0)
```

```
{
lps[i] = 0;
i++;
}
}
}
}

void layout()
{
printf("\n\t\t9th Data Structure Lab");
printf("\n\t\t_____\n"
);
printf("\t\t\t\tKMP PATTERN MATCHING PROGRAM\n");
printf("\t\t_____")
;
}

/* ------------------------------------  Main Function Starts Here ------------------------------------------------------------
---------------------*/



int main()
{
char txt[100];
char pat[50];
char option;
int choice;



do{
system("cls");
layout();
```

```
printf("\n\nEnter any  text or sentence :\n\t");
fflush(stdin);
gets(txt);

do{

printf("\n\nEnter the pattern You Wnat to search in the String : \t");
//getchar();
fflush(stdin);
gets(pat);
KMPSearch(pat, txt);



printf("\n\nDO you want to search more in same String Press 1 for Yes Otherwise Press 0 :\t");
scanf("%d",&choice);
}while(choice == 1);
/*if(choice == 1)
{
goto a;
}*/

printf("\n\n\nDo you want to Check Pattern with new Sring again\nPress Y or y for Continue..\n\tEnter your
option here :\t ");
getchar();
scanf("%c",&option);
//getchar();
}while(option == 'y' || option == 'Y');
return 0;
}
```

Output :-

```
D:\c\string_matching.exe

                9th Data Structure Lab
                ____________________________________________________________
                                KMP PATTERN MATCHING PROGRAM
                ____________________________________________________________

Enter any  text or sentence :
        My name is prakash kumar. and i am a good boy.


Enter the pattern You Wnat to search in the String :    am


                        Found - ( am ) - pattern at index 4

                        Found - ( am ) - pattern at index 32

                                ********        There is no More Match for the Given String     *********

DO you want to search more in same String Press 1 for Yes Otherwise Press 0 :
```

Program Number. 9

Program Name: Implementation of AVL Tree (Insertion, Deletion).

Date of Implementation: 03.03.2018

Description: - A menu driven program for implementation of AVL tree with Insert and Delete operations in AVL Tree.

```
// C program to insert a node in AVL tree
#include<stdio.h>
#include<stdlib.h>
#define MAX 50

// An AVL tree node
struct Node
{
int key;
struct Node *left;
struct Node *right;
int height;
};

// A utility function to get maximum of two integers
int max(int a, int b);

// A utility function to get height of the tree
int height(struct Node *N)
{
if (N == NULL)
return 0;
return N->height;
}
```

```
// A utility function to get maximum of two integers
int max(int a, int b)
{
return (a > b)? a : b;
}

/* Helper function that allocates a new node with the given key and
NULL left and right pointers. */
struct Node* newNode(int key)
{
struct Node* node = (struct Node*)
malloc(sizeof(struct Node));
node->key   = key;
node->left   = NULL;
node->right  = NULL;
node->height = 1;  // new node is initially added at leaf
return(node);
}

// A utility function to right rotate subtree rooted with y
// See the diagram given above.
struct Node *rightRotate(struct Node *y)
{
struct Node *x = y->left;
struct Node *T2 = x->right;

// Perform rotation
x->right = y;
y->left = T2;

// Update heights
y->height = max(height(y->left), height(y->right))+1;
```

```
x->height = max(height(x->left), height(x->right))+1;

// Return new root
return x;
}

// A utility function to left rotate subtree rooted with x
// See the diagram given above.
struct Node *leftRotate(struct Node *x)
{
struct Node *y = x->right;
struct Node *T2 = y->left;

// Perform rotation
y->left = x;
x->right = T2;

//  Update heights
x->height = max(height(x->left), height(x->right))+1;
y->height = max(height(y->left), height(y->right))+1;

// Return new root
return y;
}

// Get Balance factor of node N
int getBalance(struct Node *N)
{
if (N == NULL)
return 0;
return height(N->left) - height(N->right);
}
```

```
// Recursive function to insert key in subtree rooted
// with node and returns new root of subtree.
struct Node* insert(struct Node* node, int key)
{
/* 1.  Perform the normal BST insertion */
if (node == NULL)
return(newNode(key));

if (key < node->key)
node->left  = insert(node->left, key);
else if (key > node->key)
node->right = insert(node->right, key);
else // Equal keys are not allowed in BST
return node;

/* 2. Update height of this ancestor node */
node->height = 1 + max(height(node->left),
height(node->right));

/* 3. Get the balance factor of this ancestor
node to check whether this node became
unbalanced */
int balance = getBalance(node);

// If this node becomes unbalanced, then
// there are 4 cases

// Left Left Case
if (balance > 1 && key < node->left->key)
return rightRotate(node);
```

```
// Right Right Case
if (balance < -1 && key > node->right->key)
return leftRotate(node);

// Left Right Case
if (balance > 1 && key > node->left->key)
{
node->left =  leftRotate(node->left);
return rightRotate(node);
}

// Right Left Case
if (balance < -1 && key < node->right->key)
{
node->right = rightRotate(node->right);
return leftRotate(node);
}

/* return the (unchanged) node pointer */
return node;
}



struct Node * minValueNode(struct Node* node)
{
struct Node* current = node;

/* loop down to find the leftmost leaf */
while (current->left != NULL)
current = current->left;

return current;
```

```
}



// Recursive function to delete a node with given key
// from subtree with given root. It returns root of
// the modified subtree.
struct Node* deleteNode(struct Node* root, int key)
{
// STEP 1: PERFORM STANDARD BST DELETE
if (root == NULL)
return root;

// If the key to be deleted is smaller than the
// root's key, then it lies in left subtree
if ( key < root->key )
root->left = deleteNode(root->left, key);

// If the key to be deleted is greater than the
// root's key, then it lies in right subtree
else if( key > root->key )
root->right = deleteNode(root->right, key);

// if key is same as root's key, then This is
// the node to be deleted
else
{
// node with only one child or no child
if( (root->left == NULL) || (root->right == NULL) )
{
struct Node *temp = root->left ? root->left :
root->right;
```

```
// No child case
if (temp == NULL)
{
temp = root;
root = NULL;
}
else // One child case
*root = *temp; // Copy the contents of
// the non-empty child
free(temp);
}
else
{
// node with two children: Get the inorder
// successor (smallest in the right subtree)
struct Node* temp = minValueNode(root->right);

// Copy the inorder successor's data to this node
root->key = temp->key;

// Delete the inorder successor
root->right = deleteNode(root->right, temp->key);
}
}




// If the tree had only one node then return
if (root == NULL)
return root;

// STEP 2: UPDATE HEIGHT OF THE CURRENT NODE
```

```
root->height = 1 + max(height(root->left),
height(root->right));

// STEP 3: GET THE BALANCE FACTOR OF THIS NODE (to
// check whether this node became unbalanced)
int balance = getBalance(root);

// If this node becomes unbalanced, then there are 4 cases

// Left Left Case
if (balance > 1 && getBalance(root->left) >= 0)
return rightRotate(root);

// Left Right Case
if (balance > 1 && getBalance(root->left) < 0)
{
root->left =  leftRotate(root->left);
return rightRotate(root);
}

// Right Right Case
if (balance < -1 && getBalance(root->right) <= 0)
return leftRotate(root);

// Right Left Case
if (balance < -1 && getBalance(root->right) > 0)
{
root->right = rightRotate(root->right);
return leftRotate(root);
}

return root;
```

```
}

/*void search(struct Node* root, int key)
{
if(root != NULL)
{
if(ser == root->key)
{
printf("\nElement found .");
return ;

}
preOrder(root->left);
preOrder(root->right);
}

}*/
// A utility function to print preorder traversal
// of the tree.
// The function also prints height of every node
void preOrder(struct Node *root)
{
if(root != NULL)
{
printf("%d ", root->key);
preOrder(root->left);
preOrder(root->right);
}
}

void inOrder(struct Node *root)
{
```

```
if (root != NULL)
{
inOrder(root->left);
printf("%d ",root->key);
inOrder(root->right);

}
}

void postOrder(struct Node *root)
{
if(root != NULL)
{
postOrder(root->left);
postOrder(root->right);
printf("%d ", root->key);
}
}

void layout()
{
printf("\n\t\t10th Data Structure Lab");
printf("\n\t\t_____\n"
);
printf("\t\t\t\tAVL TREE :- INSERTION, DELETION\n");
printf("\t\t_____")
;
}



int main()
{
```

```
int no_of_nodes, i, element, opt;
int arr[MAX];
struct Node *root = NULL;
char choice;
int options;



main:
system("cls");
layout();



printf("\n\nPress 1 for Insert Nodes in AVL Tree.\nPress 2 For Display the Elements\nPress 3 for Delete Elements form Tree\nPress 4 to Exit the Program");
printf("\n\nEnter your choice here :\t");
fflush(stdin);
scanf("%d", &options);

switch(options)
{
case 1:
{
do{

printf("\n\nHow many Nodes you Want to Enter :- ");
scanf("%d", &no_of_nodes);

printf("\nEnter %d Elements  :-\t ",no_of_nodes);
for(i = 0; i < no_of_nodes; i += 1)
{

scanf("%d", &arr[i]);
root = insert(root, arr[i]);
```

```
}

printf("\n\nDo you want To Insert More Element :\nPress Y For Yes :\t");
getchar();
scanf("%c", &choice);
}while(choice == 'Y' || choice == 'y');



printf("\n\nDo You want to Go to Main Menu. Press Y or y for yes :\t");
getchar();
scanf("%c",&choice);
if(choice == 'y' || choice == 'Y')
{
goto main;
}

break;
}

case 2:
{
if(root != NULL)
{

printf("\nPreorder traversal of the constructed AVL tree is \n");

printf("\n\t\t\t_____________________________________________________\n");
printf("\t\t\t\t\t");
preOrder(root);
printf("\n\t\t\t_____________________________________________________ ");
```

```
printf("\n\nInorder traversal of the constructed AVL tree is \n");
printf("\n\t\t\t_____________________________________________\n");
printf("\t\t\t\t\t");
inOrder(root);
printf("\n\t\t\t_____________________________________________");

printf("\n\nPostOrder traversal of the constructed AVL tree is \n");
printf("\n\t\t\t_____________________________________________\n");
printf("\t\t\t\t\t");
postOrder(root);
printf("\n\t\t\t_____________________________________________");

}
else
{
system("cls");
printf("\n\n\t\tDear SIR there is NO elements in Tree ..... \n********\t\tPlease goto first menu and insert
Some elements ");
}

printf("\n\nDo You want to Go to Main Menu. Press Y or y for yes :\t");
getchar();
scanf("%c",&choice);
if(choice == 'y' || choice == 'Y')
{
goto main;
}

break;

}
```

```
case 3:
{
if(root!= NULL)
{



system("cls");
printf("\t\t\tAVL TREE DELETION");
printf("\n_____________________________________________________________________
_______\n");
printf("\nTree Elements Are :-\t");
preOrder(root);



do{


printf("\nWhich element you want to delete :\t");
scanf("%d", &element);
root = deleteNode(root, element);


printf("\nPreorder traversal of the AVL tree AFTER DELETION is \n");


printf("\n\t\t\t_____________________________________________\n");
printf("\t\t\t\t\t");
preOrder(root);
printf("\n\t\t\t_____________________________________________ ");



printf("\n\nInorder traversal of the constructed AVL tree AFTER DELETION  is \n");
printf("\n\t\t\t_____________________________________________\n");
printf("\t\t\t\t\t");
inOrder(root);
printf("\n\t\t\t_____________________________________________ ");
```

```
printf("\n\nPostOrder traversal of the constructed AVL tree AFTER DELETION  is \n");
printf("\n\t\t\t_____________________________________________\n");
printf("\t\t\t\t\t");
postOrder(root);
printf("\n\t\t\t_____________________________________________ ");

printf("\n\nWant to delete more. Press Y :\t");
getchar();
scanf("%c",&choice);
}while(choice == 'y' || choice == 'Y');
}

else
{
system("cls");
printf("\n\n\t\tDear SIR there is NO elements in Tree ..... \n********\t\tPlease goto first menu and insert
Some elements ");
}
printf("\n\nDo You want to Go to Main Menu. Press Y or y for yes :\t");
getchar();
scanf("%c",&choice);
if(choice == 'y' || choice == 'Y')
{
goto main;
}

break;

}

case 4:
{
```

```
exit(0);
break;

}

default:
{
goto main;
}

}

return 0;
}
```

Output :-

```
D:\c\avl_tree.exe

                10th Data Structure Lab
                ________________________________________________________________
                                AVL TREE :- INSERTION, DELETION
                ________________________________________________________________

Press 1 for Insert Nodes in AVL Tree.
Press 2 For Display the Elements
Press 3 for Delete Elements form Tree
Press 4 to Exit the Program

Enter your choice here :        2

Preorder traversal of the constructed AVL tree is

                        ____________________________________
                                        3 1 6 8
                        ____________________________________

Inorder traversal of the constructed AVL tree is

                        ____________________________________
                                        1 3 6 8
                        ____________________________________

PostOrder traversal of the constructed AVL tree is

                        ____________________________________
                                        1 8 6 3
```

Program Number. 10.

Program Name: Dijkstra Algorithm

Date of Implementation: 12.03.2018

Description: - A menu driven program for implementation of Dijkstra algortitm to find the shortest path between the vertices.

```
/* Dijkstra's Algorithm in C */
#include<stdio.h>
#include<conio.h>
#include<process.h>
#include<string.h>
#include<math.h>
#define IN 99
#define N 5

int dijsktra(int cost[][N],int source,int target)
{
int dist[N],prev[N],selected[N]={0},i,m,min,start,d,j;
char path[N];
for(i=1;i< N;i++)
{
dist[i] = IN;
prev[i] = -1;
}
start = source;
selected[start]=1;
dist[start] = 0;
while(selected[target] ==0)
{
min = IN;
m = 0;
for(i=1;i< N;i++)
```

```
{
d = dist[start] +cost[start][i];
if(d< dist[i]&&selected[i]==0)
{

dist[i] = d;
prev[i] = start;
}
if(min>dist[i] && selected[i]==0)
{
min = dist[i];
m = i;
}
}
start = m;
selected[start] = 1;
}
start = target;
j = 0;
while(start != -1)
{
path[j++] = start+65;
start = prev[start];
}
path[j]='\0';
strrev(path);
printf("%s->", path);
getchar();
return dist[target];
}


int dijkstra(int cost[][N], int source, int target);
```

```
int main()
{
int cost[N][N],i,j,w,ch,co;
int source, target,x,y;
char cho;
int ext;
printf("\t The Shortest Path Algorithm ( DIJKSTRA'S ALGORITHM in C \n\n");
for(i=1;i< N;i++)
for(j=1;j< N;j++)
cost[i][j] = IN;

do{

for(x=1;x< N;x++)
{
for(y=x+1;y< N;y++)
{
printf("Enter the weight of the path between nodes %d and %d: ",x,y);
scanf("%d",&w);
cost [x][y] = cost[y][x] = w;
}
printf("\n");
}

do{

printf("\nEnter the source:");
scanf("%d", &source);
printf("\nEnter the target");
scanf("%d", &target);
co = dijsktra(cost,source,target);
printf("\nThe Shortest Path: %d",co);
```

```
printf("\nDo you want to know path between another node .....\nPress Y for Yes.\t");
fflush(stdin);
scanf("%c",&cho);

}while(cho == 'y' || cho =='Y');



printf("\nDo you want to exit. Press 0");
getchar();
scanf("%d",&ext);
}while(ext != 0);

printf("\n\n.....");



return 0;
}
```

Output :-

```
D:\c\dijkstra_algo.exe
         The Shortest Path Algorithm ( DIJKSTRA'S ALGORITHM in C

Enter the weight of the path between nodes 1 and 2: 2
Enter the weight of the path between nodes 1 and 3: 6
Enter the weight of the path between nodes 1 and 4: 4

Enter the weight of the path between nodes 2 and 3: 1
Enter the weight of the path between nodes 2 and 4: 3

Enter the weight of the path between nodes 3 and 4: 9



Enter the source:1

Enter the target4
BE->
The Shortest Path: 4
Do you want to know path between another node .....
Press Y for Yes.
```

Program Number. 11.

Program Name: Breadth First & Depth Search.

Date of Implementation: 14.03.2018

Description: - A menu driven program for implementation of Breadth first search and Depth first Search algorithm.

```
#define MAX 1000

#include<stdio.h>
#include<stdlib.h>


typedef struct np
{
int data;
struct np *left, *right;
}node;

node *root = NULL;



node **createQueue(int *front,int *rear)
{
node **queue = (node **)malloc(sizeof(node *)*MAX);

*front = *rear = 0;
return queue;
}

void enqueue(node **queue,int *rear,node *new_node)
{
```

```
queue[*rear] = new_node;
(*rear) += 1;
}

node *dequeue(node **queue,int *front)
{
(*front) += 1;
return queue[(*front) - 1];
}

void levelOrder(node *root)
{
int front,rear;
node **queue = createQueue(&front,&rear);
node *temp_node = root;

while(temp_node)
{
printf("\n %d",temp_node->data);

if(temp_node->left)
enqueue(queue,&rear,temp_node->left);

if(temp_node->right)
enqueue(queue,&rear,temp_node->right);

temp_node = dequeue(queue,&front);
}
}

node *newnode(int x)
{
node *temp = (node *)malloc(sizeof(node));
```

```
if(NULL == temp)
{
printf("\n Malloc failed");
}

temp->data = x;
temp->left = NULL;
temp->right = NULL;

return temp;
}

node *insert(node *node,int data)
{
if(node == NULL)
return newnode(data);

if(data < node->data)
node->left = insert(node->left,data);

else if(data > node->data)
node->right = insert(node->right,data);

return node;
}


/*****************************************************************************************
**********************/


typedef struct np1
{
```

```
int key;
struct np1 *left, *right;
}node1;

node1 *root1 = NULL;

node1 *newnode1(int x)
{
node1 *temp = (node1 *)malloc(sizeof(node1));

temp->key = x;
temp->left = temp->right = NULL;

return temp;
}


void inorder(node1 *root1)
{
if(root1 != NULL)
{
inorder(root1->left);
printf("%d \n",root1->key);
inorder(root1->right);
}
}

void preorder(node1 *root1)
{
if(root1 != NULL)
{
printf("%d \n",root1->key);
preorder(root1->left);
```

```
preorder(root1->right);
}
}

void postorder(node1 *root1)
{
if(root1 != NULL)
{
postorder(root1->left);
postorder(root1->right);
printf("%d \n",root1->key);
}
}


node1 *insert(node1 *node1,int key)
{
if(node1 == NULL)
return newnode1(key);

if(key < node1->key)
node1->left = insert(node1->left,key);

else if(key > node1->key)
node1->right = insert(node1->right,key);

return node1;
}

node1 *search(node1 *root1,int key)
{
if(root1 == NULL || root1->key == key)
return root1;
```

```
if(root1->key < key)
return search(root1->right,key);

else if(root1->key > key)
return search(root1->left,key);
}




int main()
{
int choose;

main_menu:
system("cls");
printf("\n\n_____");
printf("\nBREADTH & DEPTH FRIST SEARCH");
printf("\n-----------------------------------------------------------------");



printf("\nPress 1 For Breadth First Search");
printf("\nPress 2 for Depth First Search");
printf("\nPress 3 for Exit");
printf("\nEnter Choice :\t");
scanf("%d",&choose);

switch(choose)
{
case 1:

system("cls");
{
```

```
int value,ele;
int choice;
char ch;

root = newnode(1);

do
{
do
{
printf("\n BFS MENU");
printf("\n 1.insert");
printf("\n 2.levelOrder traversal");
printf("\n 3.Goto Main Menu");
printf("\n Enter your choice:");
scanf("%d",&choice);

switch(choice)
{
case 1:printf("\n Enter the value to be inserted into tree:");
scanf("%d",&value);
insert(root,value);
break;

case 2:printf("\n levelOrder traversal:");
levelOrder(root);
break;

case 3:
{
goto main_menu;
}
default:printf("\n !!!!WRONG CHOICE!!!!");
```

```
break;
}
}
while(choice != 3);
printf("\n Do you wish to continue?");
getchar();
scanf("%c",&ch);
}
while(ch == 'y' || ch == 'Y');

break;



}



case 2:
{
system("cls");
int value,ele;
int choice;
char ch;
root = newnode(1)                                                    ;
do
{
do
{
printf("\n BST MENU");
printf("\n 1.insert");
printf("\n 2.inorder traversal");
printf("\n 3.preorder traversal");
printf("\n 4.postorder traversal");
printf("\n 5.search");
```

```
printf("\n 6.Goto Main Menu");
printf("\n Enter your choice:");
scanf("%d",&choice);

switch(choice)
{
case 1:printf("\n Enter the value to be iserted into tree:");
scanf("%d",&value);
insert(root,value);
break;

case 2:printf("\n Inorder traversal:");
inorder(root1);
break;

case 3:printf("\n Preorder traversal:");
preorder(root1);
break;

case 4:printf("\n Postorder traversal:");
postorder(root1);
break;

case 5:printf("\n Enter the value to be searched:");
scanf("%d",&ele);
printf("%d",search(root1,ele));
break;


case 6:
{
goto main_menu;
break;
```

```
}
default:printf("\n !!!!WRONG CHOICE!!!!");
break;
}
}
while(choice != 6);
printf("\n Do you wish to continue?");
getchar();
scanf("%c",&ch);
}
while(ch == 'y' || ch == 'Y');



}

case 3:
{
exit(0);

}
}
}
```

Output :-

```
BREADTH & DEPTH FRIST SEARCH
-------------------------------------------
Press 1 For Breadth First Search
Press 2 for Depth First Search
Press 3 for Exit
Enter Choice :
```

```
BFS MENU
1.insert
2.levelOrder traversal
3.Goto Main Menu
Enter your choice:1

Enter the value to be inserted into tree:36

BFS MENU
1.insert
2.levelOrder traversal
3.Goto Main Menu
Enter your choice:1

Enter the value to be inserted into tree:11

BFS MENU
1.insert
2.levelOrder traversal
3.Goto Main Menu
Enter your choice:2

levelOrder traversal:
1
25
11
36
```

Thank You!