List of Programs

		Date	Page
1	Stack Operations		1
2	Evaluation of Postfix Expression		6
3	Queue Operations		8
4	Circular Queue Operations		13
5	Single Linked List		18
6	Polynomial Addition Using Linked List		27
7	Binary Search Tree		31
8	Linear Search		39
9	Binary Search		41
10	Insertion Sort		43
11	Bubble Sort		45
12	Quick Sort		47
13	Merge Sort		49
14	Hashing		51
15	BFS and DFS		55

Exercise 1

Stack Operations

Aim

Write a program to implement stack operations.

Stack Operations Menu

```
#include<stdio.h>
#include "stackds.c"
int main()
{
  int choice,d;
  do
  {
    printf("\n\n*****************\n");
    printf("STACK OPERATIONS\n");
    printf("*****************\n");
    printf("\n1. Push");
    printf("\n2. Pop");
    printf("\n3. Display");
    printf("\n4. Exit");
    printf("\n\nEnter your choice: ");
    scanf("%d",&choice);
    switch(choice)
    {
      case 1:
      printf("Enter the number: ");
      scanf("%d",&d);
      push(d);
      break;
      case 2:
      d = pop();
      printf("\nPopped item: %d", d);
      break;
      case 3:
      display();
      break;
      case 4:
      printf("\nThank you..\n\n");
      break;
      default:
      printf("Invalid choice\n");
    }
  }while(choice != 4);
}
```

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
  int data;
  struct node *next;
};
struct node *top = NULL;

void push(int n)
{
  struct node *new;
  new = (struct node *)malloc(sizeof(struct node));
  if(new == NULL)
  {
    printf("\nStack Overflow");
    exit(1);
  }
  new->data = n;
  new->next = top;
  top = new;
}

int pop()
{
  int data;
  struct node *temp;
  temp = top;
  if(top == NULL)
  {
    printf("Stack Empty\n");
  }
  else
  {
    data = temp->data;
    top = top->next;
    free(temp);
    return data;
  }
}

void display()
{
  struct node *temp;
  temp = top;
  if(top == NULL)
  {
    printf("Stack Empty\n");
  }
  else
  {
    printf("The stack is\n");
    while(temp != NULL)
    {
      printf("%d\n",temp->data);
      temp = temp->next;
    }
  }
  printf("\n");
}
```

Output

```
*****************
STACK OPERATIONS
*****************

1. Push
2. Pop
3. Display
4. Exit

Enter your choice: 1
Enter the number: 11


*****************
STACK OPERATIONS
*****************

1. Push
2. Pop
3. Display
4. Exit

Enter your choice: 1
Enter the number: 22


*****************
STACK OPERATIONS
*****************

1. Push
2. Pop
3. Display
4. Exit

Enter your choice: 1
Enter the number: 33
```

```
*****************
STACK OPERATIONS
*****************

1. Push
2. Pop
3. Display
4. Exit

Enter your choice: 3
The stack is
33
22
11



*****************
STACK OPERATIONS
*****************

1. Push
2. Pop
3. Display
4. Exit

Enter your choice: 2

Popped item: 33

*****************
STACK OPERATIONS
*****************

1. Push
2. Pop
3. Display
4. Exit

Enter your choice: 3
The stack is
22
11



*****************
STACK OPERATIONS
*****************

1. Push
```

```
2. Pop
3. Display
4. Exit

Enter your choice: 4

Thank you..
```

Exercise 2

Evaluation of Postfix Expression

Aim

Write a program to evaluate postfix expression using stack.

Evaluation of Postfix Expression

```
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
#include<ctype.h>
#include<string.h>
struct node
{
  int data;
  struct node *next;
};
struct node *top = NULL;

void push(int n)
{
  struct node *new;
  new = (struct node *)malloc(sizeof(struct node));
  new->data = n;
  new->next = top;
  top = new;
}

int pop()
{
  int data;
  struct node *temp;
  temp = top;
  data = temp->data;
  top = top->next;
  free(temp);
  return data;
}
int main()
{
  char postfix[100],e;
  int i=0,a,b,r;
  printf("Enter the postfix expression: ");
  fgets(postfix,100,stdin);
  for(i=0;i<strlen(postfix) - 1; i++)
  {
```

```
  e = postfix[i];
  if(isdigit(e))
  {
   push(e - '0');
  }
  else
  {
   a = pop();
   b = pop();
   switch(e)
   {
    case '+':
     r = a+b;
     break;
    case '-':
     r = b-a;
     break;
    case '*':
     r = a*b;
     break;
    case '/':
     r = b/a;
     break;
    case '^':
     r = pow(b,a);
     break;
   }
   push(r);
  }
 }
 printf("\nResult=%d\n\n",r);
}
```

Output

```
Enter the postfix expression: 231*+9-

Result=-4
```

Exercise 3

Queue Operations

Aim

Write a program to implement Queue Operations.

Queue Operations Menu

```
#include<stdio.h>
#include "queueds.c"
int main()
{
  int choice,d;
  do
  {
    printf("\n\n*****************\n");
    printf("QUEUE OPERATIONS\n");
    printf("*****************\n");
    printf("\n1. Enqueue");
    printf("\n2. Dequeue");
    printf("\n3. Display");
    printf("\n4. Exit");
    printf("\n\nEnter your choice: ");
    scanf("%d",&choice);
    switch(choice)
    {
      case 1:
        printf("Enter the number: ");
        scanf("%d",&d);
        enqueue(d);
        break;
      case 2:
        d = dequeue();
        break;
      case 3:
        display();
        break;
      case 4:
        printf("\nThank you..\n\n");
        break;
      default:
        printf("Invalid choice\n");
    }
  }while(choice != 4);
}
```

Queue Data Structure

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

struct node
{
  int data;
  struct node *next;
};
struct node *front = NULL;
struct node *rear = NULL;

void enqueue(int x)
{
  struct node *new;
  new = (struct node *)malloc(sizeof(struct node));
  new->data = x;
  new->next = NULL;
  if(front == NULL && rear == NULL)
  {
    front = rear = new;
  }
  else
  {
    rear->next = new;
    rear = new;
  }
}

int dequeue()
{
  struct node *temp;
  int d;
  temp = front;

  if(front == NULL)
  {
    printf("\nThe queue is empty\n\n");
  }
  else
  {
    d = temp->data;
    front = front->next;
    free(temp);
    return d;
  }
}

void display()
{
  struct node *temp;
```

```
 if(front == NULL)
 {
  printf("\nThe queue is empty\n\n");
 }
 else
 {
  temp = front;
  printf("\nThe queue is: ");
  while(temp != NULL)
  {
   printf("%d ",temp->data);
   temp = temp ->next;
  }
 }
 printf("\n");
}
```

Output

```
*****************
QUEUE OPERATIONS
*****************

1. Enqueue
2. Dequeue
3. Display
4. Exit

Enter your choice: 1
Enter the number: 11


*****************
QUEUE OPERATIONS
*****************

1. Enqueue
2. Dequeue
3. Display
4. Exit

Enter your choice: 1
Enter the number: 22


*****************
QUEUE OPERATIONS
*****************

1. Enqueue
2. Dequeue
3. Display
```

```
4. Exit

Enter your choice: 1
Enter the number: 33


*****************
QUEUE OPERATIONS
*****************

1. Enqueue
2. Dequeue
3. Display
4. Exit

Enter your choice: 3

The queue is: 11 22 33


*****************
QUEUE OPERATIONS
*****************

1. Enqueue
2. Dequeue
3. Display
4. Exit

Enter your choice: 2


*****************
QUEUE OPERATIONS
*****************

1. Enqueue
2. Dequeue
3. Display
4. Exit

Enter your choice: 3

The queue is: 22 33


*****************
QUEUE OPERATIONS
*****************

1. Enqueue
```

```
2. Dequeue
3. Display
4. Exit

Enter your choice: 4

Thank you..
```

Exercise 4

Circular Queue Operations

Aim

Write a program to implement Circular Queue Operations.

Circular Queue Operations Menu

```
#include<stdio.h>
#include "cqueueds.c"
int main()
{
  int choice,d;
  do
  {
    printf("\n\n**************************\n");
    printf("CIRCULAR QUEUE OPERATIONS\n");
    printf("**************************\n");
    printf("\n1. Enqueue");
    printf("\n2. Dequeue");
    printf("\n3. Display");
    printf("\n4. Exit");
    printf("\n\nEnter your choice: ");
    scanf("%d",&choice);
    switch(choice)
    {
      case 1:
        printf("Enter the number: ");
        scanf("%d",&d);
        enqueue(d);
        break;
      case 2:
        d = dequeue();
        break;
      case 3:
        display();
        break;
      case 4:
        printf("\nThank you..\n\n");
        break;
      default:
        printf("Invalid choice\n");
    }
  }while(choice != 4);
}
```

Circular Queue Data Structure

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

struct node
{
  int data;
  struct node *next;
};
struct node *front = NULL;
struct node *rear = NULL;

void enqueue(int x)
{
  struct node *new;
  new = (struct node *)malloc(sizeof(struct node));
  new->data = x;
  new->next = NULL;
  if(rear == NULL)
  {
    front = rear = new;
    rear->next = front;
  }
  else
  {
    rear->next = new;
    rear = new;
    rear->next = front;
  }
}

int dequeue()
{
  struct node *temp;
  int d;
  temp = front;

  if(rear == NULL)
  {
    printf("\nThe queue is empty\n\n");
  }
  else if(front == rear)
  {
    front = rear = NULL;
    free(temp);
  }
  else
  {
    d = temp->data;
    front = front->next;
    rear->next = front;
    free(temp);
    return d;
  }
}

void display()
{
  struct node *temp;
  temp = front;
  if(front == NULL)
  {
    printf("\nThe queue is empty\n\n");
  }
  else
  {
    printf("\nThe queue is: ");
    do
    {
      printf("%d ",temp->data);
      temp = temp->next;
    }while(temp != front);
  }
  printf("\n");
}
```

Output

```
**************************
CIRCULAR QUEUE OPERATIONS
**************************

1. Enqueue
2. Dequeue
3. Display
4. Exit

Enter your choice: 1
Enter the number: 11


**************************
CIRCULAR QUEUE OPERATIONS
**************************

1. Enqueue
2. Dequeue
3. Display
4. Exit

Enter your choice: 1
Enter the number: 22
```

```
**************************
CIRCULAR QUEUE OPERATIONS
**************************

1. Enqueue
2. Dequeue
3. Display
4. Exit

Enter your choice: 1
Enter the number: 33


**************************
CIRCULAR QUEUE OPERATIONS
**************************

1. Enqueue
2. Dequeue
3. Display
4. Exit

Enter your choice: 3

The queue is: 11 22 33


**************************
CIRCULAR QUEUE OPERATIONS
**************************

1. Enqueue
2. Dequeue
3. Display
4. Exit

Enter your choice: 2


**************************
CIRCULAR QUEUE OPERATIONS
**************************

1. Enqueue
2. Dequeue
3. Display
4. Exit

Enter your choice: 3
```

```
The queue is: 22 33


**************************
CIRCULAR QUEUE OPERATIONS
**************************

1. Enqueue
2. Dequeue
3. Display
4. Exit

Enter your choice: 4

Thank you..
```

Exercise 5

Single Linked List

Aim

Write a program to implement various linked list operations.

Linked List Operations Menu

```
#include<stdio.h>
#include "linklistds.c"
int main()
{
  int choice,d,k;
  do
  {
    printf("\n***LINKED LIST OPERATIONS***\n");
    printf("\n1. Insert at Front");
    printf("\n2. Insert at End");
    printf("\n3. Insert After");
    printf("\n4. Delete from Front");
    printf("\n5. Delete from End");
    printf("\n6. Delete");
    printf("\n7. Search");
    printf("\n8. Traverse");
    printf("\n9. Exit");
    printf("\n\nEnter your choice: ");
    scanf("%d",&choice);
    switch(choice)
    {
      case 1:
        printf("Enter the number to be inserted: ");
        scanf("%d",&d);
        insertatfront(d);
        break;
      case 2:
        printf("Enter the number to be inserted: ");
        scanf("%d",&d);
        insertatend(d);
        break;
      case 3:
        printf("Enter the number to be inserted: ");
        scanf("%d",&d);
        printf("Enter the number after which the new number is to be
           inserted: ");
        scanf("%d",&k);
        insertafter(k,d);
        break;
      case 4:
        deletefromfront();
        break;
      case 5:
        deletefromend();
        break;
      case 6:
        printf("Enter the number to be deleted: ");
        scanf("%d",&d);
        delete(d);
        break;
      case 7:
        printf("Enter the number to be searched: ");
        scanf("%d",&k);
        if (search(k) == 1)
        {
          printf("%d is found in the list\n",k);
        }
        else
        {
          printf("%d is not found in the list\n",k);
        }
        break;
      case 8:
        traverse(k);
        break;
      case 9:
        printf("\nThank you..\n\n");
        break;
      default:
        printf("Invalid choice\n");
    }
  }while(choice != 9);
}
```

Linked List Data Structure

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
  int data;
  struct node *next;
};

struct node *header;

void insertatfront(int k)
{
  struct node *new;
  new = (struct node *)malloc(sizeof(struct node));
  new->data = k;
  new->next = header;
  header = new;
}

void insertatend(int k)
{
  struct node *new, *temp;
  temp = header;

  new = (struct node *) malloc(sizeof(struct node));
  new->data = k;
  new->next = NULL;

  if(temp == NULL)
  {
    header = new;
  }
  else
  {
    while(temp->next != NULL)
    {
      temp = temp->next;
    }
    temp->next = new;
  }
}

void insertafter(int k, int d)
{
  struct node *new, *temp;
  temp = header;
  while(temp != NULL)
  {
    if(temp->data == k)
    {
      new = (struct node *) malloc(sizeof(struct node));
      new->data = d;
      new->next = temp->next;
      temp->next = new;
      return;
    }
    temp=temp->next;
  }
  printf("\nNode with data %d does not exist in the list\n",k);
}

void deletefromfront()
{
  struct node *temp;
  temp = header;
  if(temp == NULL)
```

```
    {
      printf("List is empty\n");
      return;
    }
    header = temp->next;
    free(temp);
  }

  void deletefromend()
  {
    struct node *temp,*prev;
    temp = header;
    if(temp == NULL)
    {
      printf("List is empty\n");
      return;
    }
    if(temp->next == NULL)
    {
      header = NULL;
      free(temp);
      return;
    }
    while(temp->next != NULL)
    {
      prev = temp;
      temp = temp->next;
    }
    prev->next = NULL;
    free(temp);
  }

  void delete(int k)
  {
    struct node *temp, *prev;
    if(header == NULL)
    {
      printf("List is empty\n");
      return;
    }
    temp = header;
    if(temp->data == k)
    {
      header = temp->next;
      free(temp);
      return;
    }

    while(temp != NULL)
    {
      if(temp->data == k)
      {
        prev->next = temp->next;
        free(temp);
        return;
      }
      else
      {
        prev = temp;
        temp = temp->next;
      }
    }
    printf("\nNode with data %d does not exist in the list\n",k);
  }

  int search(int k)
  {
    struct node *temp;
    int flag = 0;
    temp = header;
    while(temp != NULL)
    {
      if(temp->data == k)
      {
        flag = 1;
        break;
      }
      temp = temp->next;
    }
    return flag;
  }


  void traverse()
  {
    struct node *temp;
    temp = header;
    if(temp == NULL)
    {
      printf("\nThe list is empty...");
      return;
    }
    printf("\nThe list: ");
    while(temp != NULL)
    {
      printf("%d ",temp->data);
      temp = temp->next;
    }
    printf("\n");
  }
```

Output

```
***********************
LINKED LIST OPERATIONS
***********************

1. Insert at Front
2. Insert at End
3. Insert After
4. Delete from Front
5. Delete from End
6. Delete
7. Search
8. Traverse
9. Exit

Enter your choice: 1
Enter the number to be inserted: 11


***********************
LINKED LIST OPERATIONS
***********************

1. Insert at Front
2. Insert at End
3. Insert After
4. Delete from Front
5. Delete from End
6. Delete
7. Search
8. Traverse
9. Exit

Enter your choice: 2
Enter the number to be inserted: 22


***********************
LINKED LIST OPERATIONS
***********************

1. Insert at Front
2. Insert at End
3. Insert After
4. Delete from Front
5. Delete from End
6. Delete
7. Search
8. Traverse
9. Exit
```

```
Enter your choice: 8

The list: 11 22


***********************
LINKED LIST OPERATIONS
***********************

1. Insert at Front
2. Insert at End
3. Insert After
4. Delete from Front
5. Delete from End
6. Delete
7. Search
8. Traverse
9. Exit

Enter your choice: 2
Enter the number to be inserted: 33


***********************
LINKED LIST OPERATIONS
***********************

1. Insert at Front
2. Insert at End
3. Insert After
4. Delete from Front
5. Delete from End
6. Delete
7. Search
8. Traverse
9. Exit

Enter your choice: 8

The list: 11 22 33


***********************
LINKED LIST OPERATIONS
***********************

1. Insert at Front
2. Insert at End
3. Insert After
4. Delete from Front
5. Delete from End
```

```
6. Delete
7. Search
8. Traverse
9. Exit

Enter your choice: 3
Enter the number to be inserted: 22
Enter the number after which the new number is to be inserted: 22


***********************
LINKED LIST OPERATIONS
***********************

1. Insert at Front
2. Insert at End
3. Insert After
4. Delete from Front
5. Delete from End
6. Delete
7. Search
8. Traverse
9. Exit

Enter your choice: 8

The list: 11 22 22 33


***********************
LINKED LIST OPERATIONS
***********************

1. Insert at Front
2. Insert at End
3. Insert After
4. Delete from Front
5. Delete from End
6. Delete
7. Search
8. Traverse
9. Exit

Enter your choice: 6
Enter the number to be deleted: 22


***********************
LINKED LIST OPERATIONS
***********************
```

```
1. Insert at Front
2. Insert at End
3. Insert After
4. Delete from Front
5. Delete from End
6. Delete
7. Search
8. Traverse
9. Exit

Enter your choice: 8

The list: 11 22 33


***********************
LINKED LIST OPERATIONS
***********************

1. Insert at Front
2. Insert at End
3. Insert After
4. Delete from Front
5. Delete from End
6. Delete
7. Search
8. Traverse
9. Exit

Enter your choice: 7
Enter the number to be searched: 33
33 is found in the list


***********************
LINKED LIST OPERATIONS
***********************

1. Insert at Front
2. Insert at End
3. Insert After
4. Delete from Front
5. Delete from End
6. Delete
7. Search
8. Traverse
9. Exit

Enter your choice: 9

Thank you..
```

Exercise 6

Polynomial Addition Using Linked List

Aim

Write a program to represent polynomials using linked list and add polynomials.

Polynoimal Operations Menu

```
#include<stdio.h>
#include "polynomial.c"
void createPolynomial(struct node **p);
int main()
{
  struct node *p1=NULL,*p2=NULL,*p3=NULL;
  createPolynomial(&p1);
  createPolynomial(&p2);
  printf("Addition\n");
  display(p1);
  display(p2);
  p3=add(p1,p2);
  display(p3);
}

void createPolynomial(struct node **p)
{
  int n,e,i;
  float c;
  printf("\nEnter the number of terms in the polynomial: ");
  scanf("%d",&n);
  for(i=0;i<n;i++)
  {
    printf("Enter the coefficient for term %d: ",i+1);
    scanf("%f",&c);
    printf("Enter the exponent for term %d: ",i+1);
    scanf("%d",&e);
    addterm(p,c,e);
  }
}
```

Polynoimal Data Structure

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
  float coeff;
  int exp;
  struct node *next;
};

void addterm(struct node **h,float c, int e)
{
  struct node *new, *temp;
  temp=*h;
  new = malloc(sizeof(struct node));
  new->coeff=c;
  new->exp=e;
  new->next=NULL;
  if(*h==NULL || e > (*h)->exp)
  {
    new->next=*h;
    *h=new;
  }
  else
  {
    temp=*h;
    while(temp->next!=NULL && temp->next->exp > e)
    {
      temp=temp->next;
    }
    new->next=temp->next;
    temp->next=new;
  }
}

struct node* add(struct node *h1,struct node *h2)
{
  struct node *p1=h1,*p2=h2,*h3=NULL;
  while(p1!=NULL && p2!=NULL)
  {
    if(p1->exp == p2->exp)
    {
      addterm(&h3,p1->coeff+p2->coeff,p1->exp);
      p1=p1->next;
      p2=p2->next;
    }
    else if(p1->exp > p2->exp)
    {
      break;
      addterm(&h3,p1->coeff,p1->exp);
      p1=p1->next;
    }
    else
    {
      addterm(&h3,p2->coeff,p2->exp);
      p2=p2->next;
    }
  }
```

```c
  while(p1!=NULL)
  {
    addterm(&h3,p1->coeff,p1->exp);
    p1=p1->next;
  }
  while(p2!=NULL)
  {
    addterm(&h3,p2->coeff,p2->exp);
    p2=p2->next;
  }
  return h3;
}

void display(struct node *h)
{
  struct node *temp;
  temp=h;
  while(temp!=NULL)
  {
    if(temp->exp == 0)
    {
      printf("%0.1f",temp->coeff);
    }
    else
    {
      printf("%0.1fx^%d",temp->coeff,temp->exp);
    }
    if(temp->next != NULL)
    {
      printf(" + ");
    }
    temp=temp->next;
  }
  printf("\n");
}
```

Output

```
Enter the number of terms in the polynomial: 3
Enter the coefficient for term 1: 3
Enter the exponent for term 1: 2
Enter the coefficient for term 2: 2
Enter the exponent for term 2: 1
Enter the coefficient for term 3: 5
Enter the exponent for term 3: 0

Enter the number of terms in the polynomial: 4
Enter the coefficient for term 1: 4
Enter the exponent for term 1: 3
Enter the coefficient for term 2: 3
Enter the exponent for term 2: 2
Enter the coefficient for term 3: 2
Enter the exponent for term 3: 1
Enter the coefficient for term 4: 10
Enter the exponent for term 4: 0
Addition
3.0x^2 + 2.0x^1 + 5.0
4.0x^3 + 3.0x^2 + 2.0x^1 + 10.0
4.0x^3 + 6.0x^2 + 4.0x^1 + 15.0
```

Exercise 7

Binary Search Tree

Aim

Write a program to implement binary search trees - creation, insertion, deletion, search.

BST Operations Menu

```
#include<stdio.h>
#include "bstds.c"
int main()
{
  int choice,d;
  do
  {
    printf("\n\n*****************\n");
    printf("BST OPERATIONS\n");
    printf("*****************\n");
    printf("\n1. Insert");
    printf("\n2. Delete");
    printf("\n3. Traverse (Inorder)");
    printf("\n4. Search");
    printf("\n5. Exit");
    printf("\n\nEnter your choice: ");
    scanf("%d",&choice);
    switch(choice)
    {
      case 1:
        printf("Enter the number: ");
        scanf("%d",&d);
        insert(d);
        break;
      case 2:
        printf("Enter the number to be deleted: ");
        scanf("%d",&d);
        delete(d);
        break;
      case 3:
        inorder(root);
        break;
      case 4:
        printf("Enter the number to search: ");
        scanf("%d",&d);
        search(d);
        break;
      break;
        case 5:
```

```
        printf("\nThank you..\n\n");
        break;
      default:
        printf("Invalid choice\n");
    }
  }while(choice != 5);
}
```

BST Operations Menu

```
#include<stdio.h>
#include<stdlib.h>
struct node* getSuccessor(struct node *p);
struct node
{
  int data;
  struct node *left, *right;
};
struct node *root =  NULL;

void insert(int k)
{
  int flag = 0;
  struct node *ptr,*pre,*new;
  ptr = root;
  while(ptr != NULL && flag == 0)
  {
    if(k < ptr -> data)
    {
      pre = ptr;
      ptr = ptr->left;
    }
    else if(k > ptr -> data)
    {
      pre = ptr;
      ptr = ptr->right;
    }
    else
    {
      flag = 1;
      printf("Key already exists");
      return;
    }
  }

  if(ptr == NULL)
  {
    new = (struct node *)malloc(sizeof(struct node));
    new->data = k;
    new->left = new->right = NULL;

    if(root == NULL)
    {
      root = new;
      return;
    }
    if(pre->data < k)
    {
      pre->right = new;
    }
    else
    {
      pre->left = new;
    }
  }
}

void inorder(struct node *root)
{
  if(root == NULL)
  {
    return;
  }
  inorder(root->left);
  printf("%d ",root->data);
  inorder(root->right);
}

void search(int k)
{
  struct node *ptr;
  ptr = root;
  int flag = 0;
  while(ptr != NULL && flag == 0)
  {
    if(k < ptr -> data)
    {
      ptr = ptr->left;
    }
    else if(k > ptr -> data)
    {
      ptr = ptr->right;
    }
    else
    {
      flag = 1;
      break;
    }
  }

  if(flag == 1)
  {
    printf("\n%d is found\n",k);
  }
  else
  {
    printf("\n%d is not found\n",k);
  }
}
void delete(int k)
{
  struct node *ptr,*parent,*successor;
  int flag = 0,data_successor;
  ptr = root;
  if(ptr == NULL)
  {
    printf("The tree is empty");
    return;
  }
  while(ptr != NULL && flag == 0)
  {
    if(k < ptr -> data)
    {
      parent = ptr;
      ptr = ptr->left;
    }
    else if(k > ptr -> data)
    {
      parent = ptr;
      ptr = ptr->right;
    }
    else
    {
      flag = 1;
    }
  }

  if(flag == 0)
  {
    printf("Key not found in the list\n");
    return;
  }

  if(ptr -> left == NULL && ptr ->right == NULL)
  {
    if(parent->left = ptr)
    {
      parent->left = NULL;
    }
    else
    {
      parent->right = NULL;
    }
  }
```

```
    else if(ptr->right !=NULL && ptr->left!=NULL)
    {
      successor = getSuccessor(ptr);
      data_successor = successor->data;
      delete(data_successor);
      ptr->data = data_successor;
    }
    else
    {
      if(parent->left = ptr)
      {
        if(ptr->left == NULL)
        {
          parent->left = ptr->right;
        }
        else
        {
          parent->left = ptr->left;
        }
      }
      else
      {
        if(ptr->left == NULL)
        {
          parent->right = ptr->right;
        }
        else
        {
          parent->right = ptr->left;
        }
      }
    }
  }

  struct node* getSuccessor(struct node *p)
  {
    struct node *succ;
    succ = p->right;
    if(succ != NULL)
    {
      while(succ->left != NULL)
      {
        succ = succ->left;
      }
    }
    return succ;
  }
```

Output

```
*****************
BST OPERATIONS
```

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

1. Insert
2. Delete
3. Traverse (Inorder)
4. Search
5. Exit

Enter your choice: 1
Enter the number: 11


*****************
BST OPERATIONS
*****************

1. Insert
2. Delete
3. Traverse (Inorder)
4. Search
5. Exit

Enter your choice: 1
Enter the number: 5


*****************
BST OPERATIONS
*****************

1. Insert
2. Delete
3. Traverse (Inorder)
4. Search
5. Exit

Enter your choice: 1
Enter the number: 6


*****************
BST OPERATIONS
*****************

1. Insert
2. Delete
3. Traverse (Inorder)
4. Search
5. Exit

Enter your choice: 1
Enter the number: 22


*****************
BST OPERATIONS
*****************

1. Insert
2. Delete
3. Traverse (Inorder)
4. Search
5. Exit

Enter your choice: 1
Enter the number: 15


*****************
BST OPERATIONS
*****************

1. Insert
2. Delete
3. Traverse (Inorder)
4. Search
5. Exit

Enter your choice: 1
Enter the number: 33


*****************
BST OPERATIONS
*****************

1. Insert
2. Delete
3. Traverse (Inorder)
4. Search
5. Exit

Enter your choice: 3
5 6 11 15 22 33

*****************
BST OPERATIONS
*****************

1. Insert
2. Delete
3. Traverse (Inorder)
```

```
4. Search
5. Exit

Enter your choice: 4
Enter the number to search: 15

15 is found


*****************
BST OPERATIONS
*****************

1. Insert
2. Delete
3. Traverse (Inorder)
4. Search
5. Exit

Enter your choice: 4
Enter the number to search: 99

99 is not found


*****************
BST OPERATIONS
*****************

1. Insert
2. Delete
3. Traverse (Inorder)
4. Search
5. Exit

Enter your choice: 5

Thank you..
```

Exercise 8

Linear Search

Aim

Write a program to implement linear search algorithm and print number of comparisons.

Linear Search

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
  int n,*a,k,i;
  printf("\nEnter how many numbers: ");
  scanf("%d",&n);
  printf("Enter %d numbers: ",n);
  a = (int *)malloc(n*sizeof(int));
  for(i=0;i<n;i++)
  {
    scanf("%d",a+i);
  }
  printf("\nEnter the key to be searched: ");
  scanf("%d",&k);
  for(i=0;i<n;i++)
  {
    if(a[i] == k)
    {
      printf("%d is found at location %d\n",k,i+1);
      printf("Number of comparisons done: %d\n\n",i+1);
      break;
    }
  }

  if(i==n)
  {
    printf("%d is not found in the array.\n\n",k);
    printf("Number of comparisons done: %d\n\n",n);
  }
}
```

Output

```
Enter how many numbers: 6
Enter 6 numbers: 1 2 3 4 5 6

Enter the key to be searched: 6
6 is found at location 6
```

```
Number of comparisons done: 6

Enter how many numbers: 5
Enter 5 numbers: 1 2 3 4 5

Enter the key to be searched: 6
6 is not found in the array.

Number of comparisons done: 5
```

Exercise 9

Binary Search

Aim

Write a program to implement binary search algorithm and print number of comparisons.

Binary Search

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
  int n,*a,k,l,u,flag = 0,mid,i,c=0;
  printf("\nEnter how many numbers: ");
  scanf("%d",&n);
  printf("Enter %d numbers in ascending order: ",n);
  a = (int *)malloc(n*sizeof(int));
  for(i = 0;i < n;i++)
  {
    scanf("%d",a + i);
  }
  printf("\nEnter the key to be searched: ");
  scanf("%d",&k);
  l = 0;
  u = n - 1;
  while(flag == 0 && l <= u)
  {
    mid = (l + u)/2;
    c++;
    if(k == a[mid])
    {
      printf("\n%d is found at location %d", k, mid + 1);
      printf("\nNumber of comparisions done: %d", c);
      flag = 1;
    }
    else if(k < a[mid])
    {
      u = mid - 1;
    }
    else
    {
      l = mid + 1;
    }
  }
  if(flag == 0)
  {
    printf("\n%d is not found in the array.",k);
    printf("\nNumber of comparisions done: %d", c);
  }
  printf("\n\n");
}
```

Output

```
Enter how many numbers: 16
Enter 16 numbers in ascending order: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

Enter the key to be searched: 16

16 is found at location 16
Number of comparisions done: 5
```

Exercise 10

Insertion Sort

Aim

Write a program to implement Insertion sort algorithm and print number of comparisons.

Insertion Sort

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
  int *a,i,j,n,temp,c=0;
  printf("Enter how many numbers: ");
  scanf("%d",&n);
  a = (int *)malloc(n*sizeof(int));
  printf("\nEnter %d numbers:",n);
  for(i=0;i<n;i++)
  {
    scanf("%d",a+i);
  }

  for(i=1;i<n;i++)
  {
    temp = a[i];
    j = i-1;
    while(j>=0 && a[j]>temp)
    {
      c++;
      a[j+1] = a[j];
      j--;
    }
    c++;
    a[j+1] = temp;
  }
  printf("Numbers after sorting: ");
  for(i=0;i<n;i++)
  {
    printf("%d ",*(a+i));
  }
  printf("\nNumbers of comparisons: %d",c);
  printf("\n");
}
```

Output

```
Enter how many numbers: 6
Enter 6 numbers:33 22 11 66 55 44
Numbers after sorting: 11 22 33 44 55 66
Numbers of comparisons: 11
```

Exercise 11

Bubble Sort

Aim

Write a program to implement Bubble sort algorithm and print number of comparisons.

Bubble Sort

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
  int n,*a,i,j,c=0,temp;
  printf("\nEnter how many numbers: ");
  scanf("%d",&n);
  printf("Enter %d numbers: ",n);
  a = (int *)malloc(n*sizeof(int));
  for(i=0;i<n;i++)
  {
    scanf("%d",a+i);
  }
  for(i=0;i<=n-2;i++)
  {
    for(j=0;j<=n-i-2;j++)
    {
      c++;
      if(a[j] > a[j+1])
      {
        temp = a[j];
        a[j] = a[j+1];
        a[j+1] = temp;
      }
    }
  }

  printf("\nThe sorted array: ");
  for(i=0;i<n;i++)
  {
    printf("%d ",a[i]);
  }
  printf("\nNumber of comparisons done: %d\n\n",c);
}
```

Output

```
Enter how many numbers: 6
Enter 6 numbers: 33 22 11 66 55 44
```

```
The sorted array: 11 22 33 44 55 66
Number of comparisons done: 15
```

Exercise 12

Quick Sort

Aim

Write a program to implement Bubble sort algorithm and print number of comparisons.

Bubble Sort

```
#include<stdio.h>
#include<stdlib.h>
int c = 0;
void swap(int *a, int *b)
{
  int temp;
  temp = *a;
  *a = *b;
  *b = temp;
}
void printArray(int *a, int n)
{
  int i;
  printf("\nThe array: ");
  for(i = 0;i < n;i++)
  {
    printf("%d ",a[i]);
  }
  printf("\n\n");
}

int partition(int *a, int lb, int ub)
{
  int pivot, start, end;
  pivot = a[lb];
  start = lb;
  end = ub;
  while(start < end)
  {
    c++;
    while(a[start] <= pivot && start < ub)
    {
      start++;
    }
    c++;
    while(a[end] > pivot && end > lb)
    {
      end--;
    }
    if(start < end)
    {
      swap(&a[start], &a[end]);
    }
    swap(&a[lb],&a[end]);
    return end;
  }
}

void quicksort(int *a, int lb, int ub)
{
  int loc;
  if(lb < ub)
  {
    loc = partition(a,lb,ub);
    quicksort(a, lb, loc - 1);
    quicksort(a, loc + 1, ub);
  }
}
int main()
{
  int *a,i,n;
  printf("\nEnter how many numbers: ");
  scanf("%d",&n);
  printf("Enter %d numbers: ",n);
  a = (int *)malloc(n*sizeof(int));
  for(i=0;i<n;i++)
  {
    scanf("%d",a+i);
  }
  printf("Array before sorting:");
  printArray(a,n);
  quicksort(a, 0, n - 1);
  printf("Array after sorting:");
  printArray(a,n);
  printf("Number of comparisons done: %d\n\n",c);
}
```

Output

```
Enter how many numbers: 5
Enter 5 numbers: 3 2 1 4 5
Array before sorting:
The array: 3 2 1 4 5

Array after sorting:
The array: 1 2 3 4 5

Number of comparisons done: 6
```

Exercise 13

Merge Sort

Aim

Write a program to implement Merge sort algorithm and print number of comparisons.

Merge Sort

```
#include<stdio.h>
#include<stdlib.h>
void mergesort(int *,int,int);
void merge(int *a,int lb,int mid,int ub);
int c=0,n;
int main()
{
  int *a,i,j,temp;
  printf("\nEnter how many numbers: ");
  scanf("%d",&n);
  printf("Enter %d numbers: ",n);
  a = (int *)malloc(n*sizeof(int));
  for(i=0;i<n;i++)
  {
    scanf("%d",a+i);
  }
  mergesort(a,0,n-1);
  printf("\nThe sorted array: ");
  for(i=0;i<n;i++)
  {
    printf("%d ",a[i]);
  }
  printf("\nNumber of comparisons done: %d\n\n",c);
}

void mergesort(int *a,int lb,int ub)
{
  int mid;
  if(lb < ub)
  {
    mid = (lb + ub)/2;
    mergesort(a,lb,mid);
    mergesort(a,mid+1,ub);
    merge(a,lb,mid,ub);
  }
}
```

```
void merge(int *a,int lb,int mid,int ub)
{
  int i,j,k;
  int *b;
  b = (int *)malloc(n*(sizeof(int)));
  i=lb;
  j=mid+1;
  k=lb;
  while(i<=mid && j<=ub)
  {
    c++;
    if(a[i] <= a[j])
    {
      b[k] = a[i];
      i++;
    }
    else
    {
      b[k] = a[j];
      j++;
    }
    k++;
  }
  while(j<=ub)
  {
    b[k] = a[j];
    j++;
    k++;
  }
  while(i<=mid)
  {
    b[k] = a[i];
    i++;
    k++;
  }
  for(k=lb;k<=ub;k++)
  {
    a[k] = b[k];
  }
}
```

Output

```
Enter how many numbers: 4
Enter 4 numbers: 4 3 2 1

The sorted array: 1 2 3 4
Number of comparisons done: 4
```

Exercise 14

Hashing

Aim

Write a program to implement of hash tables using various mapping functions, various collision and overflow resolving schemes.

Hashing

```
#include<stdio.h>
#include<stdlib.h>
#define SIZE 10

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

struct node *head[SIZE] = {NULL}, *ptr;

int divisionHash(int k)
{
  return k%SIZE;
}
void insert(int k)
{
  int i;
  i = divisionHash(k);
  struct node *new = (struct node *)malloc(sizeof(struct node));
  new->data = k;
  new->next=NULL;
  if(head[i] == NULL)
  {
    head[i]=new;
  }
  else
  {
    ptr = head[i];
    while(ptr->next != NULL)
    {
      ptr = ptr->next;
    }
    ptr->next=new;
  }

}
```

```
void display()
{
  int i;
  for(i=0;i<SIZE;i++)
  {
    ptr = head[i];
    while(ptr!=NULL)
    {
      if(ptr->next !=NULL)
      printf("%d --> ",ptr->data);
      else
      printf("%d",ptr->data);
      ptr=ptr->next;
    }
    printf("\n");
  }
}

void search(int k)
{
  int i;
  i = divisionHash(k);
  if(head[i] == NULL)
  {
    printf("Search element is not in the list");
  }
  else
  {
    for(ptr = head[i];ptr!=NULL;ptr=ptr->next)
    {
      if(ptr->data == k)
      {
        printf("Search element is in the list at index %d", i);
        break;
      }
    }
    if(ptr==NULL)
    {
      printf("Search element is not in the list");
    }
  }
}
int main()
{
  int choice, k;
  do
  {
    printf("\n1.Insert\n2.Display\n3.Search\n4.Exit");
    printf("\nEnter your choice: ");
    scanf("%d",&choice);
    switch(choice)
    {
      case 1:
        printf("Enter the key value to be inserted: ");
        scanf("%d",&k);
        insert(k);
        break;
      case 2:
        display();
        break;
      case 3:
        printf("Enter the key value to be searched: ");
        scanf("%d",&k);
        search(k);
        break;
      case 4:
        printf("Thank you\n\n");
        break;
      default:
        printf("Invalid choice");
    }
  }while(choice!=4);
}
```

Output

```
1.Insert
2.Display
3.Search
4.Exit
Enter your choice: 1
Enter the key value to be inserted: 11

1.Insert
2.Display
3.Search
4.Exit
Enter your choice: 1
Enter the key value to be inserted: 21

1.Insert
2.Display
3.Search
4.Exit
Enter your choice: 1
Enter the key value to be inserted: 32

1.Insert
2.Display
3.Search
4.Exit
Enter your choice: 1
Enter the key value to be inserted: 45

1.Insert
2.Display
3.Search
4.Exit
Enter your choice: 2

11 --> 21
32


45





1.Insert
2.Display
3.Search
4.Exit
Enter your choice: 3
Enter the key value to be searched: 32
Search element is in the list at index 2
1.Insert
2.Display
3.Search
4.Exit
Enter your choice: 3
Enter the key value to be searched: 55
Search element is not in the list
1.Insert
2.Display
3.Search
4.Exit
Enter your choice: 4
Thank you
```

Exercise 15

BFS and DFS

Aim

Write a program to implement BFS and DFS.

BFS and DFS

```
#include<stdio.h>
#include<stdlib.h>
int g[10][10],n;

struct node
{
  int data;
  struct node *next;
};
struct node *front = NULL;
struct node *rear = NULL;
int dfs_visited[10];

void enqueue(int x)
{
  struct node *new;
  new = (struct node *)malloc(sizeof(struct node));
  new->data = x;
  new->next = NULL;

  if(front == NULL)
  {
    front = rear = new;
  }
  else
  {
    rear->next = new;
    rear = new;
  }

}

int dequeue()
{
  struct node *temp;
  int d;
  temp = front;
  d = temp->data;
  front = front->next;
```

```
    free(temp);
    return d;
  }

  int isEmpty()
  {
    return front==NULL;
  }

  void bfs()
  {
    int i=0,j,visited[10],node;
    for(j=0;j<n;j++)
    {
      visited[j] = 0;
    }
    printf("%d ",i);
    visited[i] = 1;
    enqueue(i);
    while(!isEmpty())
    {
      node = dequeue();

      for(j=0;j<n;j++)
      {
        if(g[node][j] == 1 && visited[j] == 0)
        {
          printf("%d ",j);
          visited[j] = 1;
          enqueue(j);
        }
      }
    }
  }

  void dfs(int i)
  {
    int j;
    printf("%d ",i);
    dfs_visited[i] = 1;
    for(j=0;j<n;j++)
    {
      if(g[i][j] == 1 && dfs_visited[j] == 0)
      {
        dfs(j);
      }
    }
  }
  int main()
  {
    int i,j,invalid=0;
```

```
 printf("Enter the number of nodes in the graph: ");
 scanf("%d",&n);
 printf("\nEnter the adjancency matrix of the graph\n");
 for(i=0;i<n;i++)
 {
  for(j=0;j<n;j++)
  {
   scanf("%d",&g[i][j]);
   if(g[i][j] != 0 && g[i][j] != 1)
   {
    invalid = 1;
   }
  }
 }
 if(invalid == 1)
 {
  printf("Invalid adjancency matrix. Enter only 1 or 0.\n\n");
  exit(1);
 }
 printf("\n***************\n");
 printf("BFS Traversal\n");
 printf("***************\n");
 bfs();
 printf("\n\n***************\n");
 printf("DFS Traversal\n");
 printf("***************\n");
 dfs(4);
 printf("\n\n");
}
```

Output

```
Enter the number of nodes in the graph: 7

Enter the adjancency matrix of the graph
0 1 1 1 0 0 0
1 0 1 0 0 0 0
1 1 0 1 1 0 0
1 0 1 0 1 0 0
0 0 1 1 0 1 1
0 0 0 0 1 0 0
0 0 0 0 1 0 0

***************
BFS Traversal
***************
0 1 2 3 4 5 6

***************
DFS Traversal
***************
4 2 0 1 3 5 6
```