

**department of Computer Engineering & Applications**

**Institute of Engineering & Technology**

Lab Manual

**Subject Name & Code: Data Structures & Algorithm Lab (CSE3081)**

**Course: B.Tech.**

**Year: II Semester: III**

**INDEX**

|  |  |  |
| --- | --- | --- |
| **Ex.No.** | **Program** | **Page No.** |
| 1 | Program to implement various operations in a singly linked list. |  |
| 2 | Program to implement insertion, deletion and traversal in a doubly linked List. |  |
| 3 | Program to implement polynomial addition using linked list. |  |
| 4 | Program to demonstrate the various operations on stack. |  |
| 5 | Program to convert an infix expression into postfix expression. |  |
| 6 | Program to evaluate a given postfix expression. |  |
| 7 | Program to implement Tower of Hanoi problem using Recursion. |  |
| 8 | Program to demonstrate the implementation of various operations on linear and circular queue. |  |
| 9 | Program to demonstrate the implementation of insertion and traversals on a binary search tree. |  |
| 10 | Program to implement Dijkstra’s Algorithm to find the shortest path between source and destination. |  |
| 11 | Program to search a given element as entered by the user using sequential and binary search to search a given element as entered by the user. |  |
| 12 | Implementation of various sorting algorithms like Selection Sort, Bubble Sort, Insertion Sort, Merge Sort, Quick Sort and Heap Sort. |  |

**EXPERIMENT NO: 1**

**Environment:** Linux (Ubunto 14)

**Tools/ Language:**  C

**Objective:**  Program to implement various operations in a singly linked list.

**Algorithms:**

**1. INSERT\_BEG**

1. Create a node and name it TEMP.
2. if (TEMP = Null), then

Print: ”Overflow” and exit.

1. Set TEMP -> data = VLAUE
2. Set TEMP -> next = START
3. Set START = TEMP
4. Exit

**2. INSERT\_END**

1. Create a node and name it TEMP.
2. if (TEMP = Null), then

Print: ”Overflow” and exit.

1. Set TEMP -> data =VALUE
2. Set TEMP -> next = Null
3. If (START = Null), then

Set START = TEMP

1. Else

Set PTR=START

while PTR->next != NULL

Set PTR= PTR->next

Set PTR->next = TEMP

1. Exit

**3. DELETE\_BEG**

1. If (START = Null)

Print: “Underflow” and exit.

1. Set PTR = START
2. Set START = PTR->next
3. Print: “Deleted node is PTR->data”
4. Free PTR
5. Exit

**4. DELETE\_END**

1. If (START = Null)

Print: “Underflow” and exit.

1. Set PTR = PTR1 = START
2. if PTR->next = Null

Set START = Null

1. Else

While PTR->next != Null

Set PTR1=PTR and PTR = PTR->next

Set PTR1->next = Null

1. Print: “Deleted node is PTR->data”
2. Free PTR
3. Exit

**Code:**

#include<stdio.h>

#include<stdlib.h>

struct node

{

int data;

struct node \*next;

}\*first=NULL;

void insert()

{

struct node \*temp;

struct node \*nn=(struct node\*)malloc(sizeof(struct node));

printf("enter the data\n");

scanf("%d",&nn->data);

temp=first;

while(temp->next!=first)

temp=temp->next;

temp->next=nn;

nn->next=NULL;

}

void display()

{

struct node \*temp;

temp=first;

if(temp==NULL)

{

printf("no elements\n");

return;

}

printf("elements in linked list are\n");

while(temp!=NULL)

{

printf("%d\n",temp->data);

temp=temp->next;

}

}

void deletion()

{

struct node \*temp;

temp=first;

first=first->next;

temp->next=NULL;

free(temp);

}

int main()

{

int op;

do

{

printf("1.insertion\n2.deletion\n3.display\n4.exi\n");

printf("enter option\n");

scanf("%d",&op);

switch(op)

{

case 1:insert();

break;

case 2:deletion();

break;

case 3:display();

break;

}

}while(op!=6);

}

**Post Experiment Questions:**

1. A Linked list can grow and shrink in size dynamically at \_\_\_\_\_\_\_.
2. Write an algorithm to detect loop in a linked list.
3. Reverse a linked list.
4. Implement an algorithm to reverse a singly linked list. (with and without recursion)

**EXPERIMENT NO: 2**

**Environment:** Linux (Ubunto 14)

**Tools/ Language:**  C

**Objective:**  Program to implement insertion, deletion and traversal in a doubly linked List.

**Algorithm:**

1. **insertAfter(List list, Node node, Node newNode)**
   1. newNode.prev := node
   2. newNode.next := node.next
   3. if node.next == null
   4. list.lastNode := newNode
   5. else
   6. node.next.prev := newNode
   7. node.next := newNode
2. **insertBefore(List list, Node node, Node newNode)**
   1. newNode.prev := node.prev
   2. newNode.next := node
   3. if node.prev == null
   4. list.firstNode := newNode
   5. else
   6. node.prev.next := newNode
   7. node.prev := newNode
3. **insertBeginning(List list, Node newNode)**
   1. if list.firstNode == null
   2. list.firstNode := newNode
   3. list.lastNode := newNode
   4. newNode.prev := null
   5. newNode.next := null
   6. else
   7. insertBefore(list, list.firstNode, newNode)
4. **insertEnd(List list, Node newNode)**
   1. if list.lastNode == null
   2. insertBeginning(list, newNode)
   3. else
   4. insertAfter(list, list.lastNode, newNode)
5. **remove(List list, Node node)**
   1. if node.prev == null
   2. list.firstNode := node.next
   3. else
   4. node.prev.next := node.next
   5. if node.next == null
   6. list.lastNode := node.prev
   7. else
   8. node.next.prev := node.prev
   9. destroy node
6. **Traverse\_list()**
   1. node := list.firstNode
   2. while node ≠ null
   3. <do something with node.data>
   4. node := node.next

**Code:**

#include<stdio.h>

#include<conio.h>

#include<stdlib.h>

struct node

{

struct node \*previous;

int data;

struct node \*next;

}\*head, \*last;

void insert\_begning(int value)

{

struct node \*var,\*temp;

var=(struct node \*)malloc(sizeof(struct node));

var->data=value;

if(head==NULL)

{

head=var;

head->previous=NULL;

head->next=NULL;

last=head;

}

else

{

temp=var;

temp->previous=NULL;

temp->next=head;

head->previous=temp;

head=temp;

}

}

void insert\_end(int value)

{

struct node \*var,\*temp;

var=(struct node \*)malloc(sizeof(struct node));

var->data=value;

if(head==NULL)

{

head=var;

head->previous=NULL;

head->next=NULL;

last=head;

}

else

{

last=head;

while(last!=NULL)

{

temp=last;

last=last->next;

}

last=var;

temp->next=last;

last->previous=temp;

last->next=NULL;

}

}

int insert\_after(int value, int loc)

{

struct node \*temp,\*var,\*temp1;

var=(struct node \*)malloc(sizeof(struct node));

var->data=value;

if(head==NULL)

{

head=var;

head->previous=NULL;

head->next=NULL;

}

else

{

temp=head;

while(temp!=NULL && temp->data!=loc)

{

temp=temp->next;

}

if(temp==NULL)

{

printf("\n%d is not present in list ",loc);

}

else

{

temp1=temp->next;

temp->next=var;

var->previous=temp;

var->next=temp1;

temp1->previous=var;

}

}

last=head;

while(last->next!=NULL)

{

last=last->next;

}

}

int delete\_from\_end()

{

struct node \*temp;

temp=last;

if(temp->previous==NULL)

{

free(temp);

head=NULL;

last=NULL;

return 0;

}

printf("\nData deleted from list is %d \n",last->data);

last=temp->previous;

last->next=NULL;

free(temp);

return 0;

}

int delete\_from\_middle(int value)

{

struct node \*temp,\*var,\*t, \*temp1;

temp=head;

while(temp!=NULL)

{

if(temp->data == value)

{

if(temp->previous==NULL)

{

free(temp);

head=NULL;

last=NULL;

return 0;

}

else

{

var->next=temp1;

temp1->previous=var;

free(temp);

return 0;

}

}

else

{

var=temp;

temp=temp->next;

temp1=temp->next;

}

}

printf("data deleted from list is %d",value);

}

void display()

{

struct node \*temp;

temp=head;

if(temp==NULL)

{

printf("List is Empty");

}

while(temp!=NULL)

{

printf("-> %d ",temp->data);

temp=temp->next;

}

}

int main()

{

int value, i, loc;

head=NULL;

printf("Select the choice of operation on link list");

printf("\n1.) insert at begning\n2.) insert at end\n3.) insert at middle");

printf("\n4.) delete from end\n5.) reverse the link list\n6.) display list\n7.)exit");

while(1)

{

printf("\n\nenter the choice of operation you want to do ");

scanf("%d",&i);

switch(i)

{

case 1:

{

printf("enter the value you want to insert in node ");

scanf("%d",&value);

insert\_begning(value);

display();

break;

}

case 2:

{

printf("enter the value you want to insert in node at last");

scanf("%d",&value);

insert\_end(value);

display();

break;

}

case 3:

{

printf("after which data you want to insert data ");

scanf("%d",&loc);

printf("enter the data you want to insert in list ");

scanf("%d",&value);

insert\_after(value,loc);

display();

break;

}

case 4:

{

delete\_from\_end();

display();

break;

}

case 5:

{

printf("enter the value you want to delete");

scanf("%d",value);

delete\_from\_middle(value);

display();

break;

}

case 6 :

{

display();

break;

}

case 7 :

{

exit(0);

break;

}

}

}

printf("\n\n%d",last->data);

display();

getch();

}

**Post Experiment Questions:**

1. Write down the algorithms for various operations in circular linked lists.
2. Delete an element from a doubly linked list.
3. Implement an algorithm to reverse a doubly linked list
4. Write down the algorithm for implementing DEQUE using doubly linked lists.

**EXPERIMENT NO: 3**

**Environment:** Linux (Ubunto 14)

**Tools/ Language:**  C

**Objective:**  Program to implement polynomial addition using linked list.

**Algorithm:** The algorithm adds to polynomials, represented by linked lists with start as P and Q respectively and stores the result in another linked list R.

ADD\_POLYNOMIAL (P, Q, R)

1. Set PPTR=P, QPRT=Q,
2. while ( PPTR!=NULL and QPTR!=NULL)
3. if (PPTR->expo = QPTR->expo)
4. Insert a node at end of R with coff=PPTR->coff + QPTR->coff and expo=PPTR->expo
5. Set PPTR = PPTR->next and QPTR = QPTR->next
6. if (PPTR->expo > QPTR->expo)
7. Insert a copy of PPTR at end of R
8. Set PPTR = PPTR->next
9. if (PPTR->expo < QPTR->expo)
10. Insert a copy of QPTR at end of R
11. Set QPTR = QPTR->next
12. while (PPTR != NULL)
13. Insert a copy of PPTR at end of R
14. Set PPTR = PPTR->next
15. while (QPTR != NULL)
16. Insert a copy of QPTR at end of R
17. Set QPTR = QPTR->next
18. Exit.

**Code:**

#include<stdio.h>

#include<conio.h>

void create();

void display();

void polyadd();

struct list{

int coeff;

int pow;

struct list \*next;

};

struct list \*poly1,\*poly2,\*poly3;

void main()

{

clrscr();

poly1=poly2=poly3=NULL;

poly1=(struct list\*)malloc(sizeof(struct list));

poly2=(struct list\*)malloc(sizeof(struct list));

poly3=(struct list\*)malloc(sizeof(struct list));

printf("Enter the first polynomial:\n");

create(poly1);

printf("Enter the second polynomial:\n");

create(poly2);

printf("Addition of two polynomial:\n");

polyadd(poly1,poly2,poly3);

printf("first polynomial\n");

display(poly1);

printf("\nsecond polynomial\n");

display(poly2);

printf("\nprintf the polynomial:\n");

display(poly3);

getch();

}

void create(struct list \*n\_node)

{

char ch;

int c,p;

do{

printf("Enter the coeffiecient and power:\t");

scanf("%d%d",&c,&p);

n\_node->coeff=c;

n\_node->pow=p;

n\_node->next=(struct list\*)malloc(sizeof(struct list));

n\_node=n\_node->next;

n\_node->next=NULL;

printf("Continue\n");

ch=getch();

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

}

void display(struct list \*node)

{

while(node->next!=NULL)

{

printf("%dx^%d",node->coeff,node->pow);</em>

if(node->coeff>0)

printf("+");

else

printf("-");

node=node->next;

}

}

void polyadd(struct list \*poly1,struct list \*poly2,struct list \*poly3)

{

while(poly1->next && poly2->next)

{

if(poly1->pow>poly2->pow)

{

poly3->coeff=poly1->coeff;

poly3->pow=poly1->pow;

poly1=poly1->next;

}

else if(poly2->pow>poly1->pow)

{

poly3->coeff=poly2->coeff;

poly3->pow=poly2->pow;

poly2=poly2->next;

}

else

{

poly3->coeff=poly1->coeff+poly2->coeff;

poly3->pow=poly1->pow;

poly1=poly1->next;

poly2=poly2->next;

}

poly3->next=(struct list\*)malloc(sizeof(struct list));

poly3=poly3->next;

poly3->next=NULL;

}

while(poly1->next ||poly2->next)

{

if(poly1->next)

{

poly3->coeff=poly1->coeff;

poly3->pow=poly1->pow;

poly1=poly1->next;

}

if(poly2->next)

{

poly3->coeff=poly2->coeff;

poly3->pow=poly2->pow;

poly2=poly2->next;

}

poly3->next=(struct list\*)malloc(sizeof(struct list));

poly3=poly3->next;

poly3->next=NULL;

}

}

**Post Experiment Questions:**

1. Write the various applications of linked list.
2. how linked list can be used in sparse matrix .

**EXPERIMENT NO: 4**

**Environment:** Linux (Ubunto 14)

**Tools/ Language:**  C

**Objective:**  Program to demonstrate the various operations on stack.

**Algorithm:**

1. **Push( ):**

1. If (TOP == MAX) Then

2. Print: Overflow

3. Else

4. Set TOP = TOP + 1

5. Set STACK[TOP] = ITEM

6. Print: ITEM inserted

7. Exit

1. **Pop ():**
2. If (TOP == 0) Then

2. Print: Underflow

3. Else

4. Set ITEM = STACK[TOP]

5. Set TOP =TOP-1

6. Print: ITEM deleted

7. Exit

**Code:**

#include<stdio.h>

#include<string.h>

#include<ctype.h>

#define size 100

int top=-1;

int flag=0;

int stack[size];

void push(int \*,int);

int pop(int \*);

void display(int \*);

void push(int s[],int d)

{

if(top==(size-1))

flag=0;

else

{

flag=1;

++top;

s[top]=d;

}}

int pop(int s[])

{

int popped\_element;

if(top==-1)

{

popped\_element=0;

flag=0;

}

else

{

flag=1;

popped\_element=s[top];

--top;

}

return(popped\_element);

}

void display(int s[])

{

int i;

if(top==-1)

{

printf("\n stack is empty");

}

else

{

for(i=top;i>=0;--i)

printf("\n %d",s[i]);

}}

/\* this is the main function \*/

void main()

{

int data;

char choice;

int q=0;

int top=-1;

clrscr();

do

{

printf("\n push->i pop->p quit->q:");

printf("enter your choice");

do

{

choice=getchar();

choice=tolower(choice);

}

while(strchr("ipq",choice)==NULL);

printf("your choice is %c",choice);

switch(choice)

{

case'i':printf("\n input element to push");

scanf("%d",&data);

push(stack,data);

if(flag)

{

printf("\n after inserting ");

display(stack);

if(top==(size-1))

printf("\n stack is full");

}

else

printf("\n stack is overflown after pushing");

break;

case 'p':data=pop(stack);

if(flag)

{

printf("\n data is popped:%d",data);

printf("\n now the stack is as follows :\n");

display(stack);

}

else

printf("\n stack is underflown");

break;

case'q':q=1;

} }

while(!q);

}

**Post Experiment Questions:**

1. How would you implement a queue from a stack?
2. What is stack?
3. What are the applications of a stack?
4. List the difference between queue and stack.

**EXPERIMENT NO: 5**

**Environment:** Linux (Ubunto 14)

**Tools/ Language:**  C

**Objective:**  Program to convert an infix expression into postfix expression.

**Algorithm:** Let Q be any infix expression and we have to convert it to postfix expression P. For this the following procedure will be followed.

1. Push left parenthesis onto STACK and add right parenthesis at the end of Q.

2. Scan Q from left to right and repeat step 3 to 6 for each element of Q until the STACK is empty.

3. If an operand is encountered add it to P.

4. If a left parenthesis is encountered push it onto the STACK.

5. If an operator is encountered, then

Repeatedly pop from STACK and add to P each operator which has same precedence as or higher precedence than the operator encountered.

Push the encountered operator onto the STACK.

6. If a right parenthesis is encountered, then

Repeatedly pop from the STACK and add to P each operator until a left parenthesis is encountered.

Remove the left parenthesis; do not add it to P.

7. Exit

**Code:**

#define SIZE 50 /\* Size of Stack \*/

#include <ctype.h>

char s[SIZE];

int top = -1; /\* Global declarations \*/

push(char elem) { /\* Function for PUSH operation \*/

s[++top] = elem;

}

char pop() { /\* Function for POP operation \*/

return (s[top--]);

}

int pr(char elem) { /\* Function for precedence \*/

switch (elem) {

case '#':

return 0;

case '(':

return 1;

case '+':

case '-':

return 2;

case '\*':

case '/':

return 3;

}

}

main() { /\* Main Program \*/

char infx[50], pofx[50], ch, elem;

int i = 0, k = 0;

printf("\n\nRead the Infix Expression ? ");

scanf("%s", infx);

push('#');

while ((ch = infx[i++]) != '\0') {

if (ch == '(')

push(ch);

else if (isalnum(ch))

pofx[k++] = ch;

else if (ch == ')') {

while (s[top] != '(')

pofx[k++] = pop();

elem = pop(); /\* Remove ( \*/

} else { /\* Operator \*/

while (pr(s[top]) >= pr(ch))

pofx[k++] = pop();

push(ch);

}

}

while (s[top] != '#') /\* Pop from stack till empty \*/

pofx[k++] = pop();

pofx[k] = '\0'; /\* Make pofx as valid string \*/

printf("\n\nGiven Infix Expn: %s Postfix Expn: %s\n", infx, pofx);

}

**Post Experiment Questions:**

1. What are the polish notaions ?
2. What is an expression tree?

**EXPERIMENT NO: 6**

**Environment:** Linux (Ubunto 14)

**Tools/ Language:** C

**Objective:**  Program to evaluate a given postfix expression.

**Algorithm:**

1. Create a stack to store operands (or values).
2. Scan the given expression and do following for every scanned element.  
   a) If the element is a number, push it into the stack  
   b) If the element is a operator, pop operands for the operator from stack.
3. Evaluate the operator and push the result back to the stack.
4. When the expression is ended, the number in the stack is the final answer.

**Code:**

#include<stdio.h>

#include<conio.h>

#include<string.h>

#define MAX 50

int stack[MAX];

char post[MAX];

int top=-1;

void pushstack(int tmp);

void calculator(char c);

void main()

{

int i;

clrscr();

printf("Insert a postfix notation :: ");

gets(post);

for(i=0;i<strlen(post);i++)

{

if(post[i]>='0' && post[i]<='9')

{

pushstack(i);

}

if(post[i]=='+' || post[i]=='-' || post[i]=='\*' ||

post[i]=='/' || post[i]=='^')

{

calculator(post[i]);

}

}

printf("\n\nResult :: %d",stack[top]);

getch();

}

void pushstack(int tmp)

{

top++;

stack[top]=(int)(post[tmp]-48);

}

void calculator(char c)

{

int a,b,ans;

a=stack[top];

stack[top]='\0';

top--;

b=stack[top];

stack[top]='\0';

top--;

switch(c)

{

case '+':

ans=b+a;

break;

case '-':

ans=b-a;

break;

case '\*':

ans=b\*a;

break;

case '/':

ans=b/a;

break;

case '^':

ans=b^a;

break;

default:

ans=0;

}

top++;

stack[top]=ans;

}

**Post Experiment Questions:**

1. What is the complexity of the above program.
2. Why postfix and prefix notations are used?

**EXPERIMENT NO: 7**

**Environment:** Linux (Ubunto 14)

**Tools/ Language:**  C

**Objective:**  Program to implement Tower of Hanoi problem using Recursion.

**Algorithm:**

**Hanoi(n: integer; source, dest, by: char);**

1. if (n=1) then
2. writeln('Move the plate from ', source, ' to ', dest)
3. else begin
4. Hanoi(n-1, source, by, dest);
5. writeln('Move the plate from ', source, ' to ', dest);
6. Hanoi(n-1, by, dest, source);
7. end;

**Code:**

#include <stdio.h>

void towers(int, char, char, char);

int main()

{

int num;

printf("Enter the number of disks : ");

scanf("%d", &num);

printf("The sequence of moves involved in the Tower of Hanoi are :\n");

towers(num, 'A', 'C', 'B');

return 0;

}

void towers(int num, char frompeg, char topeg, char auxpeg)

{

if (num == 1)

{

printf("\n Move disk 1 from peg %c to peg %c", frompeg, topeg);

return;

}

towers(num - 1, frompeg, auxpeg, topeg);

printf("\n Move disk %d from peg %c to peg %c", num, frompeg, topeg);

towers(num - 1, auxpeg, topeg, frompeg);

}

**Post Experiment Questions:**

1. Explain Tower of Hanoi problem.
2. Explain Recursion in details with one example.
3. Define Tail Recursion.

**EXPERIMENT NO: 8**

**Environment:** Linux (Ubunto 14)

**Tools/ Language:**  C

**Objective:** Program to demonstrate the implementation of various operations on linear and circular queue.

**Algorithm:**

**Enqueue:**

1. If ( REAR = size ) then //Queue is full

2. print "Queue is full"

3. Exit

4. Else

5. If ( FRONT = 0 ) and ( REAR = 0 ) then //Queue is empty

6. FRONT = 1

7. End if

8. REAR = REAR + 1 // increment REAR

9. Que[ REAR ] = ITEM

10. End if

11. Stop

**Dequeue:**

1. If ( FRONT = 0 ) then

2. print "Queue is empty"

3. Exit

4. Else

5. ITEM = Que [ FRONT ]

6. If ( FRONT = REAR )

7. REAR = 0

8. FRONT = 0

9. Else

10. FRONT = FRONT + 1

11. End if

12. End if

13. Stop

**Code:**

#include < stdio.h>

#include < conio.h>

#include < malloc.h>

#include < process.h>

#include < ctype.h>

#define SIZE 5

void menu();

void display();

int underflow();

int overflow();

void enqueue(int);

void dequeue();

int queue[SIZE];

int front=-1;

int rear=-1;

void main()

{

clrscr();

menu();

}

void menu()

{

int choice,item;

printf("MENU");

printf("\n1. Insert into the queue");

printf("\n2. Delete from queue");

printf("\n3. Display");

printf("\n4. Exit");

printf("\nEnter your choice: ");

scanf("%d",&choice);

switch(choice)

{

case 1:

clrscr();

if(overflow()==0)

{

printf("\nEnter the item tobe inserted: ");

scanf("%d",&item);

enqueue(item);

clrscr();

printf("\nAfter inserting queue is:\n");

}

display();

getch();

clrscr();

menu();

break;

case 2:

clrscr();

if(underflow()==1)

{

dequeue();

if(underflow()==1)

{

printf("\nAfter deletion queue is:\n");

display();

}

}

getch();

clrscr();

menu();

break;

case 3:

clrscr();

if(underflow()==1)

{

printf("The queue is:\n");

display();

}

getch();

clrscr();

menu();

break;

case 4:

exit(1);

default:

clrscr();

printf("Your choice is wrong\n\n");

menu();

}

}

int underflow()

{

if((front==-1)&&(rear==-1))

{

printf("\nQueue is empty");

return(0);

}

else

{

return(1);

}

}

int overflow()

{

if(rear==SIZE-1)

{

printf("\nQueue is full\n");

return(1);

}

else

{

return(0);

}

}

void enqueue(int item)

{

if((front==-1)&&(rear==-1))

{

front=0;

rear=0;

}

else

{

rear=rear+1;

}

queue[rear]=item;

}

void dequeue()

{

if(front==rear)

{

front=-1;

rear=-1;

}

else

{

front=front+1;

}

}

void display()

{

int i;

for(i=front;i<=rear;i++)

{

printf("\nElement %d : %d",i+1,queue[i]);

}

}

**Post Experiment Questions:**

1. What is queue?
2. What is the cost of enqueue and dequeue?
3. What are the applications of queue?

**EXPERIMENT NO: 9**

**Environment:** Linux (Ubunto 14)

**Tools/ Language:**  C

**Objective:**  Program to demonstrate the implementation of insertion and traversals on a binary search tree.

**Algorithm:**

Insert(tree,data)

1. if (!root)
2. root = new Node(data);
3. else if (data < root->data)
4. insert(root->left, data);
5. else if (data > root->data)
6. insert(root->right, data);

Inorder(tree)

1. Traverse the left subtree, i.e., call Inorder(left-subtree)

2. Visit the root.

3. Traverse the right subtree, i.e., call Inorder(right-subtree)

Preorder(tree)

1. Visit the root.

2. Traverse the left subtree, i.e., call Preorder(left-subtree)

3. Traverse the right subtree, i.e., call Preorder(right-subtree)

Postorder(tree)

1. Traverse the left subtree, i.e., call Postorder(left-subtree)

2. Traverse the right subtree, i.e., call Postorder(right-subtree)

3. Visit the root.

**Code:**

#include <stdio.h>

#include <conio.h>

#include <malloc.h>

struct node

{

struct node \*left;

int data;

struct node \*right;

};

void main()

{

void insert(struct node \*\*,int);

void inorder(struct node \*);

void postorder(struct node \*);

void preorder(struct node \*);

struct node \*ptr;

int no,i,num;

ptr = NULL;

ptr->data=NULL;

clrscr();

printf("\nProgram for Tree Traversal\n");

printf("Enter the number of nodes to add to the tree.<BR>\n");

scanf("%d",&no);

for(i=0;i<no;i++)

{

printf("Enter the item\n");

scanf("%d",&num);

insert(&ptr,num);

}

//getch();

printf("\nINORDER TRAVERSAL\n");

inorder(ptr);

printf("\nPREORDER TRAVERSAL\n");

preorder(ptr);

printf("\nPOSTORDER TRAVERSAL\n");

postorder(ptr);

getch();

}

void insert(struct node \*\*p,int num)

{

if((\*p)==NULL)

{

printf("Leaf node created.");

(\*p)=malloc(sizeof(struct node));

(\*p)->left = NULL;

(\*p)->right = NULL;

(\*p)->data = num;

return;

}

else

{

if(num==(\*p)->data)

{

printf("\nREPEATED ENTRY ERROR VALUE REJECTED\n");

return;

}

if(num<(\*p)->data)

{

printf("\nDirected to left link.\n");

insert(&((\*p)->left),num);

}

else

{

printf("Directed to right link.\n");

insert(&((\*p)->right),num);

}

}

return;

}

void inorder(struct node \*p)

{

if(p!=NULL)

{

inorder(p->left);

printf("\nData :%d",p->data);

inorder(p->right);

}

else

return;

}

void preorder(struct node \*p)

{

if(p!=NULL)

{

printf("\nData :%d",p->data);

preorder(p->left);

preorder(p->right);

}

else

return;

}

void postorder(struct node \*p)

{

if(p!=NULL)

{

postorder(p->left);

postorder(p->right);

printf("\nData :%d",p->data);

}

else

return;

}

**Post Experiment Questions:**

1. What are the pre, post and inorder traversal of a tree.
2. Do a breadth first traversal of a tree.
3. How can we make tree from post and inorder traversal.
4. What is tree?
5. What are the advantages of BST?

**EXPERIMENT NO: 10**

**Environment:** Linux (Ubunto 14)

**Tools/ Language:**  C

**Objective:** Program to implement Dijkstra’s Algorithm to find the shortest path between source and destination.

**Algorithm:**

1 Dijkstra(Graph, source):

2

3 create vertex set Q

4

5 for each vertex v in Graph:

6 dist[v] ← INFINITY

7 prev[v] ← UNDEFINED

8 add v to Q

9

10 dist[source] ← 0

11

12 while Q is not empty:

13 u ← vertex in Q with min dist[u] // Source node will be selected first

14 remove u from Q

15

16 for each neighbor v of u: // where v is still in Q.

17 alt ← dist[u] + length(u, v)

18 if alt < dist[v]: // A shorter path to v has been found

19 dist[v] ← alt

20 prev[v] ← u

21

22 return dist[], prev[]

**Code:**

#include <stdio.h>

#include <limits.h>

#define V 9

int minDistance(int dist[], bool sptSet[])

{

int min = INT\_MAX, min\_index;

for (int v = 0; v < V; v++)

if (sptSet[v] == false && dist[v] <= min)

min = dist[v], min\_index = v;

return min\_index;

}

int printSolution(int dist[], int n)

{

printf("Vertex Distance from Source\n");

for (int i = 0; i < V; i++)

printf("%d \t\t %d\n", i, dist[i]);

}

void dijkstra(int graph[V][V], int src)

{

int dist[V];

bool sptSet[V];

for (int i = 0; i < V; i++)

dist[i] = INT\_MAX, sptSet[i] = false;

dist[src] = 0;

for (int count = 0; count < V-1; count++)

{

int u = minDistance(dist, sptSet);

sptSet[u] = true;

for (int v = 0; v < V; v++)

if (!sptSet[v] && graph[u][v] && dist[u] != INT\_MAX

&& dist[u]+graph[u][v] < dist[v])

dist[v] = dist[u] + graph[u][v];

}

printSolution(dist, V);

}

int main()

{

int graph[V][V] = {{0, 4, 0, 0, 0, 0, 0, 8, 0},

{4, 0, 8, 0, 0, 0, 0, 11, 0},

{0, 8, 0, 7, 0, 4, 0, 0, 2},

{0, 0, 7, 0, 9, 14, 0, 0, 0},

{0, 0, 0, 9, 0, 10, 0, 0, 0},

{0, 0, 4, 0, 10, 0, 2, 0, 0},

{0, 0, 0, 14, 0, 2, 0, 1, 6},

{8, 11, 0, 0, 0, 0, 1, 0, 7},

{0, 0, 2, 0, 0, 0, 6, 7, 0}

};

dijkstra(graph, 0);

return 0;

}

**Post Experiment Questions:**

1. What is the complexity of the Dijkastra’s algorithm?
2. What is the limitations of this algorithm?
3. What are the differences between Dijkastra’s algorithm and Prim’s algorithm?

**EXPERIMENT NO: 11**

**Environment:** Linux (Ubunto 14)

**Tools/ Language:**  C

**Objective:** Program to search a given element as entered by the user using sequential and binary search to search a given element as entered by the user.

**Algorithm:**

1. **Sequential Search:**
   1. start from the leftmost element of an array.
   2. one by one compare x with each element of array.
   3. if x matches with an element, return the index.
   4. If x doesn’t match with any of elements, print not found message.
2. **Binary Search:**
3. Compare x with the middle element.
4. If x matches with middle element, we return the mid index.
5. Else If x is greater than the mid element, then x can only lie in right half subarray after the mid element. So we recur for right half.
6. Else (x is smaller) recur for the left half.

**Code:**

**1. Sequential Search:**

#include <stdio.h>

int search(int arr[], int n, int x)

{

    int i;

    for (i=0; i<n; i++)

        if (arr[i] == x)

         return i;

    return -1;

}

int main(void)

{

   int arr[] = {2, 3, 4, 10, 40};

   int n = sizeof(arr)/ sizeof(arr[0]);

   int x = 10;

   int result = Search(arr, 0, n-1, x);

   (result == -1)? printf("Element is not present in array")

                 : printf("Element is present at index %d", result);

   return 0;

}

**2. Binary Search:**

#include <stdio.h>

int binarySearch(int arr[], int l, int r, int x)

{

   if (r >= l)

   {

        int mid = l + (r - l)/2;

        if (arr[mid] == x)  return mid;

        if (arr[mid] > x) return binarySearch(arr, l, mid-1, x);

       return binarySearch(arr, mid+1, r, x);

   }

    return -1;

}

int main(void)

{

   int arr[] = {2, 3, 4, 10, 40};

   int n = sizeof(arr)/ sizeof(arr[0]);

   int x = 10;

   int result = binarySearch(arr, 0, n-1, x);

   (result == -1)? printf("Element is not present in array")

                 : printf("Element is present at index %d", result);

   return 0;

}

**Post Experiment Questions:**

1. What is time complexity of Binary Search?
2. Can Binary Search be used for linked lists?
3. We know that binary search is more efficient than linear search. Why would we ever use linear search?

**EXPERIMENT NO: 12**

**Environment:** Linux (Ubunto 14)

**Tools/ Language:**  C

**Objective:** Implementation of various sorting algorithms like Selection Sort, Bubble Sort, Insertion Sort, Merge Sort, Quick Sort and Heap Sort.

**Algorithm:**

**Selection sort:**

1. for i = 1:n,
   1. k = i
   2. for j = i+1:n, if a[j] < a[k], k = j
   3. swap a[i,k]
2. end

**Bubble Sort:**

1. for i = 1:n,
2. swapped = false
3. for j = n:i+1,
4. if a[j] < a[j-1],
   1. swap a[j,j-1]
   2. swapped = true
5. break if not swapped
6. end

**Insertion Sort:**

1. for i = 2:n,
   1. for (k = i; k > 1 and a[k] < a[k-1]; k--)
   2. swap a[k,k-1]
2. end

**Heap Sort:**

#sort

1. for i = 1:n,
   1. swap a[1,n-i+1]
   2. sink(a,1,n-i)
2. end

# sink from i in a[1..n]

sink(a,i,n):

1. lc = 2\*i
2. if lc > n, return # no children
3. rc = lc + 1
4. mc = (rc > n) ? lc : (a[lc] > a[rc]) ? lc : rc
5. if a[i] >= a[mc], return # heap ordered
6. swap a[i,mc]
7. sink(a,mc,n)

**Quick Sort:**

# choose pivot

swap a[1,rand(1,n)]

# 2-way partition

k = 1

for i = 2:n, if a[i] < a[1], swap a[++k,i]

swap a[1,k]

→ invariant: a[1..k-1] < a[k] <= a[k+1..n]

# recursive sorts

sort a[1..k-1]

sort a[k+1,n]

**Merge Sort:**

# split in half

m = n / 2

# recursive sorts

sort a[1..m]

sort a[m+1..n]

# merge sorted sub-arrays using temp array

b = copy of a[1..m]

i = 1, j = m+1, k = 1

while i <= m and j <= n,

a[k++] = (a[j] < b[i]) ? a[j++] : b[i++]

→ invariant: a[1..k] in final position

while i <= m,

a[k++] = b[i++]

→ invariant: a[1..k] in final position

**Code:**

**/\* Selection Sort\*/**

#include<stdio.h>

int main(){

int s,i,j,temp,a[20];

printf("Enter total elements: ");

scanf("%d",&s);

printf("Enter %d elements: ",s);

for(i=0;i<s;i++)

scanf("%d",&a[i]);

for(i=0;i<s;i++){

for(j=i+1;j<s;j++){

if(a[i]>a[j]){

temp=a[i];

a[i]=a[j];

a[j]=temp;

}

}

}

printf("After sorting is: ");

for(i=0;i<s;i++)

printf(" %d",a[i]);

return 0;

}

**/\* Bubble sort \*/**

#include <stdio.h>

int main()

{

int array[100], n, c, d, swap;

printf("Enter number of elements\n");

scanf("%d", &n);

printf("Enter %d integers\n", n);

for (c = 0; c < n; c++)

scanf("%d", &array[c]);

for (c = 0 ; c < ( n - 1 ); c++)

{

for (d = 0 ; d < n - c - 1; d++)

{

if (array[d] > array[d+1]) /\* For decreasing order use < \*/

{

swap = array[d];

array[d] = array[d+1];

array[d+1] = swap;

}

}

}

printf("Sorted list in ascending order:\n");

for ( c = 0 ; c < n ; c++ )

printf("%d\n", array[c]);

return 0;

}

**/\* insertion sort \*/**

#include <stdio.h>

int main()

{

int n, array[1000], c, d, t;

printf("Enter number of elements\n");

scanf("%d", &n);

printf("Enter %d integers\n", n);

for (c = 0; c < n; c++) {

scanf("%d", &array[c]);

}

for (c = 1 ; c <= n - 1; c++) {

d = c;

while ( d > 0 && array[d] < array[d-1]) {

t = array[d];

array[d] = array[d-1];

array[d-1] = t;

d--;

}

}

printf("Sorted list in ascending order:\n");

for (c = 0; c <= n - 1; c++) {

printf("%d\n", array[c]);

}

return 0;

}

**/\* Heap Sort \*/**

#include <stdio.h>

void main()

{

int heap[10], no, i, j, c, root, temp;

printf("\n Enter no of elements :");

scanf("%d", &no);

printf("\n Enter the nos : ");

for (i = 0; i < no; i++)

scanf("%d", &heap[i]);

for (i = 1; i < no; i++)

{

c = i;

do

{

root = (c - 1) / 2;

if (heap[root] < heap[c])

{

temp = heap[root];

heap[root] = heap[c];

heap[c] = temp;

}

c = root;

} while (c != 0);

}

printf("Heap array : ");

for (i = 0; i < no; i++)

printf("%d\t ", heap[i]);

for (j = no - 1; j >= 0; j--)

{

temp = heap[0];

heap[0] = heap[j /\* swap max element with rightmost leaf element \*/

heap[j] = temp;

root = 0;

do

{

c = 2 \* root + 1; /\* left node of root element \*/

if ((heap[c] < heap[c + 1]) && c < j-1)

c++;

if (heap[root]<heap[c] && c<j){

temp = heap[root];

heap[root] = heap[c];

heap[c] = temp;

}

root = c;

} while (c < j);

}

printf("\n The sorted array is : ");

for (i = 0; i < no; i++)

printf("\t %d", heap[i]);

printf("\n Complexity : \n Best case = Avg case = Worst case = O(n logn) \n");

}

**/\* Quick Sort \*/**

#include<stdio.h>

void swap (int a[], int left, int right)

{

int temp;

temp=a[left];

a[left]=a[right];

a[right]=temp;

}//end swap

void quicksort( int a[], int low, int high )

{

int pivot;

// Termination condition!

if ( high > low )

{

pivot = partition( a, low, high );

quicksort( a, low, pivot-1 );

quicksort( a, pivot+1, high );

}

} //end quicksort

int partition( int a[], int low, int high )

{

int left, right;

int pivot\_item;

int pivot = left = low;

pivot\_item = a[low];

right = high;

while ( left < right )

{

// Move left while item < pivot

while( a[left] <= pivot\_item )

left++;

// Move right while item > pivot

while( a[right] > pivot\_item )

right--;

if ( left < right )

swap(a,left,right);

}

// right is final position for the pivot

a[low] = a[right];

a[right] = pivot\_item;

return right;

}//end partition

// void quicksort(int a[], int, int);

void printarray(int a[], int);

int main()

{

int a[50], i, n;

printf("\nEnter no. of elements: ");

scanf("%d", &n);

printf("\nEnter the elements: \n");

for (i=0; i<n; i++)

scanf ("%d", &a[i]);

printf("\nUnsorted elements: \n");

printarray(a,n);

quicksort(a,0,n-1);

printf("\nSorted elements: \n");

printarray(a,n);

}//end main

void printarray(int a[], int n)

{

int i;

for (i=0; i<n; i++)

printf(" %d ", a[i]);

printf("\n");

}//end printarray

**/\* Merge Sort \*/**

#include<stdio.h>

int arr[8]={1, 2, 3, 4, 5, 6, 7, 8};

int main()

{

int i;

merge\_sort(arr, 0, 7);

printf("Sorted array:");

for(i = 0; i < 8; i++)

printf("%d", arr[i]);

return 0;

}

int merge\_sort(int arr[],int low,int high)

{

printf("\nmerge\_sort initialization\n");

int mid;

if(low < high)

{

mid = (low + high) / 2;

// Divide and Conquer

merge\_sort(arr, low, mid);

printf("\n merge\_sort first\n");

merge\_sort(arr, mid + 1, high);

printf("\n merge\_sort second\n");

// Combine

merge(arr, low, mid, high);

printf("\nmerging\n");

}

return 0;

}

int merge(int arr[], int l, int m, int h)

{

int arr1[10], arr2[10];

int n1, n2, i, j, k;

n1 = m - l + 1;

n2 = h - m;

for(i = 0; i < n1; i++)

arr1[i] = arr[l + i];

for(j = 0; j < n2; j++)

arr2[j] = arr[m + j + 1];

arr1[i] = 9999;

arr2[j] = 9999;

i = 0;

j = 0;

for(k = l; k <= h; k++)

{

if(arr1[i] <= arr2[j])

arr[k] = arr1[i++];

else

arr[k] = arr2[j++];

}

return 0;

}

**Post Experiment Questions:**

1. Distinguish between sorting and searching.
2. Name various internal and external sorting techniques.
3. Name various sorting techniques that make use of divide and conquer strategies.
4. Elaborate the working of Heap sort with one example.