Department of Computer Science and Engineering

Lab Manual

for

Data Structures Laboratory

CS3311

Regulation 2021

CS3311 – III Semester



ADHIPARASAKTHI ENGINEERING COLLEGE

Melmaruvathur, Chengalpet District- 603319.

INDEX

Expt. No.	Name of the Experiments	Page No.
1A.	Array Implementation of Stack ADT	1
1B.	Array Implementation of Queue ADT	4
1C.	Array Implementation of Circular Queue ADT	8
2.	Implementation of Singly Linked List	13
3A.	Linked List Implementation of Stack ADT	21
3B.	Linked List Implementation of Linear Queue ADT	25
4.	Implementation of Polynomial Manipulation Using Linked List	29
5A.	Implementation of Evaluating Postfix Expressions	34
5B.	Implementation of Infix to Postfix Conversion	37
6.	Implementation of Binary Search Trees	42
7.	Implementation of AVL Trees	49
8.	Implementation of Heaps Using Priority Queues	57
9.	Implementation of Dijkstra's Algorithm	66
10.	Implementation of Prim's Algorithm	69
11.	Implementation of Linear Search And Binary Search	74
12.	Implementation of Insertion Sort And Selection Sort	78
13.	Implementation of Merge Sort	82
14.	Implementation of Open Addressing(Linear Probing And Quadratic Probing)	86

APEC

EX. NO: 1A	ARRAY IMPLEMENTATION OF STACK ADT

Aim:

To write a C program to perform the implementation of stack ADTs using arrays.

Algorithm:

1. Start the program.
2. Declare all the required functions and select the operation to be performed.
3. Allocate the memory location using the malloc function, and if the stack is empty, then the top and bottom elements are the same.
4. For deletion operations, check if the stack has only one element; if so, perform deletion. If the stack is empty, then print that the stack is empty.
5. For the display operation, use the while loop and check if i<top If so, print the elements of the stack.
6. Check if top==0; if so, then the stack is empty.
7. Stop the program.

Program:

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
int top = 0, a[100], max = 50;
void main()
{
        void push();
        void pop();
        void display();
        int ch;
        clrscr();
        printf("\n1.Push\ n2.Pop\ n3.Display\ n4.Exit");
        while (1)
        {
                printf("\nEnter your choice: ");
                scanf(" % d", &ch);
                switch (ch)
                {
                        case 1:
```

APEC

```
                        push();
                        break;
                case 2:
                        pop();
                        break;
                case 3:
                        display();
                        break;
                default:
                        exit();
                }
        }
}

void push()
{
        if (top == max)
                printf("\n Array is full");
        else
        {
                printf("\n Enter the data: ");
                scanf(" % d", &x);
                st[top++] = x;
        }
}

void pop()
{
        if (top == 0)
                printf("\n Array is empty");
        else
        {
                top--;
        }
}

void display()
{
        int i;
        if (top == 0)
           printf("\n Stack is empty");
        else
        {
                printf("\n Array elememts are: ");
                for (i = 0; i < top; i++)
                        printf(" % d\ t", st[i]);
        }
}
```

APEC

Output:

```
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 1
Enter the data: 4
Enter your choice: 1
Enter the data: 6
Enter your choice: 1
Enter the data: 8
Enter your choice: 3
Array elements are: 4 6 8
Enter your choice: 2
Enter your choice: 3
Array elements are: 4 6
```

APEC

Result:

Thus, the C program for the implementation of stack ADT using arrays was executed and verified successfully.

EX. NO: 1B	ARRAY IMPLEMENTATION OF QUEUE ADT

Aim:

To write a C program to perform the array implementation of the queue

Algorithm:

1. Start the program.
2. Declare all the required functions and select the operation to be performed.
3. Allocate the memory location using the malloc function, and if the stack is empty, then the top and bottom elements are the same.
4. For enqueue operation, check if rear == front. If so, then the queue is full. Otherwise, set qu[rear++]=else.
5. For the dequeue operation, use the while loop and check if rear == 0. If so, then the print queue is empty.
6. For display operation, check if front<rear; if so, print the values.
7. Stop the program.

Program:

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
int ch, que[50], max = 50, front = 0, rear = 0;
void main()
{
        void enqueue();
        void dequeue();
        void display();
        clrscr();
        printf("\n1.Insertion(enqueue)\ n2.Deletion(dequeue)\ n3.Display\ n4.Exit");
        while (1)
        {
                printf("Enter your choice:");
                scanf(" % d", &ch);
                switch (ch)
```

APEC

```
                {
                        case 1:
                                enqueue();
                                break;
                        case 2:
                                dequeue();
                                break;
                        case 3:
                                display();
                                break;
                        default:
                                exit(0);
                }
        }
}

void enqueue()
{
        int ele;
        if (rear == max)
           printf("The queue is full”);
        else
        {
                printf("Enter the data:”);
                scanf(“ % d”, &ele); que[rear++] = ele;
        }
}

void dequeue()
{
        int i;
        if (rear == 0)
                printf("The queue is empty”);
        else
        {
                printf(“\n Deleted element is % d, qu[front]);
                front++;
```

APEC

```
        }
}

void display()
{
        int i;
        if (rear == 0)
                printf("The queue is empty”);
        else
        {
           printf(“\n Queue elememts are: ”);
           for (i = front; i < rear; i++)
                        printf(“ % d\ t”, que[i]);
        }
}
```

APEC

Output:

```
1. Insertion (enqueue)
2. Deletion (dequeue)
3. Display
4. Exit
Enter your choice: 1
Enter the data: 6
Enter your choice: 1
Enter the data: 8
Enter your choice: 3
Queue elements are: 6 8
Enter your choice: 2
Deleted element is: 6
Enter your choice: 1
Enter the data: 10
Enter your choice: 3
Queue elements are: 8 10
```

APEC

Result:

Thus the C program for implementation of queue ADT using array was executed and verified successfully.

EX. NO: 1C	ARRAY IMPLEMENTATION OF CIRCULAR QUEUE ADT

Aim:

To write a C program to implement a circular queue using an array.

Algorithm:

Insert an element in a circular queue:

1. If (rear+1)%max = front
   Write “overflow”
   Go to step 4
   [end of if]
2. If front = -1 and rear = -1
   Set front = rear = 0
   Else if rear = max - 1 and front! = 0
   Set rear = 0
   Else
   Set rear = (rear + 1) % max
   [end of if]
3. Set queue[rear] = val
4. Exit

Delete an element from the circular queue:

1. If front = -1
   write "underflow"
   go to step 4.
   [end of if]
2. Set val = queue[front]
3. If front = rear
   Set front = rear = -1
   Else
   If front = max -1
   Set front = 0
   Else

APEC

Set front = front + 1

[end of if]

[end of if]

4. exit

Program:

```
#include <stdio.h>
#define max 6
int queue[max];          // array declaration
int front = -1;
int rear = -1;

// function to insert an element in a circular queue
void enqueue(int element)
{
        if (front == -1 && rear == -1)          // condition to check queue is empty
        {
                front = 0;
                rear = 0;
                queue[rear] = element;
        }
        else if ((rear + 1) % max == front)     // condition to check queue is full
        {
                printf("Queue is overflow..");
        }
        else
        {
                rear = (rear + 1) % max;        // rear is incremented
                queue[rear] = element;          // assigning a value to the queue at the rear
position.
        }
}
```

APEC

```
// function to delete the element from the queue
int dequeue()
{
        if ((front == -1) && (rear == -1))      // condition to check queue is empty
                printf("\nQueue is underflow.");
}
else if (front == rear)
{
        printf("\nThe dequeued element is %d", queue[front]);
        front = -1;
        rear = -1;
}
else
{
        printf("\nThe dequeued element is %d", queue[front]);
        front = (front + 1) % max;
}
}
// function to display the elements of a queue
void display()
{
        int i = front;
        if (front == -1 && rear == -1)
        {
                printf("\n Queue is empty..");
        }
        else
        {
                printf("\nElements in a Queue are :");
                while (i <= rear)
                {
                        printf("%d,", queue[i]);
                        i = (i + 1) % max;
```

APEC

```
        }
    }

int main()
{
    int choice = 1, x;        // variables declaration
    while (choice < 4 && choice != 0)    // while loop
    {
        printf("\n Press 1: Insert an element");
        printf("\nPress 2: Delete an element");
        printf("\nPress 3: Display the element");
        printf("\nEnter your choice");
        scanf("%d", &choice);
        switch (choice)
        {
            case 1:
                printf("Enter the element which is to be inserted");
                scanf("%d", &x);
                enqueue(x);
                break;
            case 2:
                dequeue();
                break;
            case 3:
                display();
        }
    }

    return 0;
}
```

APEC

Output:

```
 Press 1: Insert an element
Press 2: Delete an element
Press 3: Display the element
Enter your choice
1
Enter the element which is to be inserted
10

 Press 1: Insert an element
Press 2: Delete an element
Press 3: Display the element
Enter your choice
1
Enter the element which is to be inserted
20

 Press 1: Insert an element
Press 2: Delete an element
Press 3: Display the element
Enter your choice
1
Enter the element which is to be inserted
30

 Press 1: Insert an element
Press 2: Delete an element
Press 3: Display the element
Enter your choice
3

Elements in a Queue are :10,20,30,
 Press 1: Insert an element
Press 2: Delete an element
Press 3: Display the element
Enter your choice
2

The dequeued element is 10
```

Result:

Thus the C program for implementation of a circular queue using array was executed and verified successfully.

EX. NO: 2	IMPLEMENTATION OF SINGLY LINKED LIST

Aim:

To write a C program to perform a singly linked list.

Algorithm:

1. Start the program
2. Declare all the functions and using the switch case select the operation to be performed.
3. Declare a structure with all the required variables and allocate the memory using the malloc function.
4. In the insert beginning function set y->link=head. For the insert middle check if c<pos if so set z=x; and x=x->link; and increment c.
5. In the delete beginning function check the above function but perform delete operation.
6. In the display function display the array elements
7. Stop the program.

Program:

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
struct node
{
        int data;
        struct node * link;
};
struct node *head, *x, *y, *z;
void main()
{
        void create();
        void insbeg();
        void insmid();
        void insend();
        void delbeg();
        void delmid();
        void delend();
        void display();
        int ch;
```

APEC

```
        clrscr();
        while (1)
        {
                printf("\n 1.Creation");
                printf("\n 2.Insertion at beginning");
                printf("\n 3.Insertion at middle");
                printf("\n 4.Insertion at end");
                printf("\n 5.Deletion at beginning");
                printf("\n 6.Deletion at middle");
                printf("\n 7.Deletion at end");
                printf("\n 8.Display");
                printf("\n 9.Exit");
                printf("\n Enter ur choice:");
                scanf("%d", &ch);
                switch (ch)
                {
                        case 1:
                                create();
                                break;
                        case 2:
                                insbeg();
                                break;
                        case 3:
                                insmid();
                                break;
                        case 4:
                                insend();
                                break;
                        case 5:
                                delbeg();
                                break;
                        case 6:
                                delmid();
                                break;
                        case 7:
                                delend();
                                break;
                        case 8:
                                display();
                                break;
                        default:
                                exit(0);
                }
        }
}
```

APEC

```
void create()
{
        int c;
        head = NULL;
        x = (struct node *) malloc(sizeof(struct node));
        printf("\n Enter the data:");
        scanf("%d", &x->data);
        x->link = NULL;
        head = x;
        printf("\n Do u wish to continue press 1 otherwise 0:");
        scanf("%d", &c);
        while (c != 0)
        {
                y = (struct node *) malloc(sizeof(struct node));
                printf("\n Enter the data:");
                scanf("%d", &y->data);
                y->link = NULL;
                x->link = y;
                x = y;
                printf("\n Do u wish to continue press 1 otherwise 0:");
                scanf("%d", &c);
        }
}

void insbeg()
{
        y = (struct node *) malloc(sizeof(struct node));
        printf("\n Enter the data:");
        scanf("%d", &y->data);
        y->link = head;
        head = y;
}

void insmid()
{
        int pos, c = 1;
        y = (struct node *) malloc(sizeof(struct node));
        printf("\n Enter the data:");
        scanf("%d", &y->data);
        printf("\n Enter the position to be inserted:");
        scanf("%d", &pos);
        x = head;
```

APEC

```
while (c < pos)
        {
                z = x;
                x = x->link;
                c++;
        }

        y->link = x;
        z->link = y;
}

void insend()
{
        y = (struct node *) malloc(sizeof(struct node));
        printf("\n Enter the data:");
        scanf("%d", &y->data);
        y->link = NULL;
        x = head;
        while (x->link != NULL)
        {
                x = x->link;
        }

        x->link = y;
}

void delbeg()
{
        if (head == NULL)
                printf("\n List is empty");
        else
        {
                x = head;
                head = x->link;
                free(x);
        }
}

void delmid()
{
        int pos, c = 1;
        if (head == NULL)
                printf("\n List is empty");
```

APEC

```
        else
        {
                printf("\n Enter the position to be deleted:");
                scanf("%d", &pos);
                x = head;
                while (c < pos)
                {
                        z = x;
                        x = x->link;
                        c++;
                }

                z->link = x->link;
                free(x);
        }
}

void delend()
{
        if (head == NULL)
                printf("\n List is empty");
        else
        {
                x = head;
                while (x->link != NULL)
                {
                        y = x;
                        x = x->link;
                }

                y->link = NULL;
                free(x);
        }
}

void display()
{
        if (head == NULL)
                printf("\n List is empty");
        else
        {
                x = head;
                printf("\n List elements are \n");
                while (x->link != NULL)
```

APEC

```
                {
                        printf("%d->", x->data);
                        x = x->link;
                }

                printf("%d", x->data);
        }
}
```

APEC

Output:

1. Creation
2. Insertion at beginning
3. Insertion at middle
4. Insertion at end
5. Deletion at beginning
6. Deletion at middle
7. Deletion at end
8. Display
9. Exit

Enter ur choice: 1

Enter the data: 10

Enter ur choice: 1

Enter the data: 20

Enter ur choice: 1

Enter the data: 30

Enter ur choice: 1

Enter the data: 40

Enter ur choice: 1

Enter the data: 50

Do u wish to continue press 1 otherwise 0:1

Enter ur choice: 8

List elements are 10→20→30→40→50

Do u wish to continue press 1 otherwise 0:

Enter ur choice: 1

Enter the data: 25

Enter the position to be inserted: 2

Enter ur choice: 8

List elements are 10→20→25→30→40→50

Do u wish to continue press 1 otherwise 0:

Enter ur choice: 6

Enter the position to be deleted: 3

Do u wish to continue press 1 otherwise 0:1

APEC

Enter ur choice: 8

List elements are 10→20→25→40→50

Do u wish to continue press 1 otherwise 0:1

Enter ur choice: 7

Enter ur choice: 8

List elements are 10→20→25→30→40

APEC

Result:

Thus the C program for implementation of singly linked list was executed and verified successfully.

EX. NO: 3A	LINKED LIST IMPLEMENTATION OF STACK ADT

Aim:

To perform the linked list implementation of stack using C program.

Algorithm:

1. Start the program.
2. Declare all the required functions and select the operation to be performed
3. Allocate the memory location using malloc function and if the stack is empty then the top is the bottom element.
4. Insert an element into the stack.
5. For deletion operation checks if the stack has only one element if so perform deletion. If the stack is empty then print that the stack is empty.
6. For the display operation use the while loop and check if x->link! =NULL if so print the elements.
7. Stop the program.

Program:

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

struct node *top = NULL, *x;
void main()
{
        void push();
        void pop();
        void display();
        int ch;
        clrscr();
        while (1)
        {
                printf("\n OPTIONS");
```

APEC

```
                printf("\n 1.Insertion (push)");
                printf("\n 2.Deletion (pop)");
                printf("\n 3.Display");
                printf("\n 4.Exit");
                printf("\n Enter ur choice:");
                scanf("%d", &ch);
                switch (ch)
                {
                        case 1:
                                push();
                                break;
                        case 2:
                                pop();
                                break;
                        case 3:
                                display();
                                break;
                        default:
                                exit(0);
                }
        }
}

void push()
{
        if (top == NULL)
        {
                x = (struct node *) malloc(sizeof(struct node));
                printf("\n Enter the data:");
                scanf("%d", &x->data);
                x->link = NULL;
                top = x;
        }
        else
        {
                x = (struct node *) malloc(sizeof(struct node));
                printf("\n Enter the data:");
                scanf("%d", &x->data);
                x->link = top;
                top = x;
        }
}
```

APEC

```
void pop()
{
        if (top == NULL)
                printf("\n Stack is empty");
        else
        {
                x = top;
                printf("\n Popped Element is %d", x->data);
                top = top->link;
                free(x);
        }
}

void display()
{
        x = top;
        printf("\n Stack Elements are\n");
        while (x->link != NULL)
        {
                printf("%d->", x->data);
                x = x->link;
        }

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

}
```

APEC

Output:

OPTIONS

1. Insertion (push)
2. Deletion (pop)
3. Display
4. Exit

Enter ur choice: 1

Enter the data: 10

Enter ur choice: 1

Enter the data: 20

Enter ur choice: 1

Enter the data: 30

Enter ur choice: 3

Stack Elements are: 10 20 30

Enter ur choice: 2

Popped Element is: 30

Enter ur choice: 3

Stack Elements are: 10 20

APEC

Result:

Thus the C program for linked list implementation of stack was executed and verified successfully.

EX. NO: 3B	LINKED LIST IMPLEMENTATION OF LINEAR QUEUE ADT

Aim:

To write a C program to perform linked list implementation of queue.

Algorithm:

1. Start the program.
2. Declare all the required functions and select the operation to be performed.
3. Allocate the memory location using malloc function and if the queue is empty then the front and the rear element are the same.
4. For deletion operation check if the queue has only one element if so performs deletion. If the queue contains more elements then perform deletion accordingly.
5. For the display operation use the while loop and checks if x->link! = NULL if so prints the elements.
6. For the display operation if rear = = null then print that the queue is empty.
7. Stop the program.

Program:

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
struct node
{
        int data;
        struct node * link;
};
struct node *front = NULL, *rear = NULL, *x;
void main()
{
        void enqueue();
        void dequeue();
        void display();
        int ch;
        clrscr();
        while (1)
        {
                printf("\n OPTIONS");
                printf("\n 1.Insertion (enqueue)");
                printf("\n 2.Deletion (dequeue)");
```

APEC

```
                printf("\n 3.Display");
                printf("\n 4.Exit");
                printf("\n Enter ur choice:");
                scanf("%d", &ch);
                switch (ch)
                {
                        case 1:
                                enqueue();
                                break;
                        case 2:
                                dequeue();
                                break;
                        case 3:
                                display();
                                break;
                        default:
                                exit(0);
                }
        }
}

void enqueue()
{
        if (rear == NULL)
        {
                x = (struct node *) malloc(sizeof(struct node));
                printf("\n Enter the data:");
                scanf("%d", &x->data);
                x->link = NULL;
                rear = x;
                front = x;
        }
        else
        {
                x = (struct node *) malloc(sizeof(struct node));
                printf("\n Enter the data:");
                scanf("%d", &x->data);
                x->link = NULL;
                rear->link = x;
                rear = x;
        }
}
```

APEC

```
void dequeue()
{
        if (front == NULL)
                printf("\n Queue is empty");
        else
        {
                x = front;
                printf("\n Dequeued Element is %d", x->data);
                front = x->link;
                free(x);
        }
}

void display()
{
        x = front;
        printf("\n Queue Elements are\n");
        while (x->link != NULL)
        {
                printf("%d->", x->data);
                x = x->link;
        }

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

}
```

APEC

Output:

```
OPTIONS
1. Insertion (enqueue)
2. Deletion (dequeue)
3. Display
4. Exit
Enter ur choice: 1
Enter the data: 10
Enter ur choice: 1
Enter the data: 20
Enter ur choice: 1
Enter the data: 30
Enter ur choice: 1
Enter the data: 40
Enter ur choice: 3
Queue Elements are 10    20    30    40
Enter ur choice: 2
Dequeued Element is 10
Enter ur choice: 3
Queue Elements are 20    30    40
```

APEC

Result:

Thus the C program for linked list implementation of queue was executed and verified successfully.

EX. NO: 4	IMPLEMENTATION OF POLYNOMIAL MANIPULATION USING LINKED LIST

Aim:

To write a c program to implement the polynomial manipulation using linked list.

Algorithm:

Let p and q be the two polynomials represented by the linked list.

1. While p and q are not null, repeat step 2.

2. If powers of the two terms ate equal then if the terms do not cancel then insert the sum of the terms into the sum Polynomial.

Advance p

Advance q

Else if the power of the first polynomial> power of second

Then insert the term from first polynomial into sum polynomial

Advance p

Else insert the term from second polynomial into sum polynomial

Advance q

3.Copy the remaining terms from the non-empty polynomial into the sum polynomial.

Program:

```
#include <stdio.h>
#include <malloc.h>
#include <conio.h>
struct link
{
        int coeff;
        int pow;
        struct link * next;
};
struct link *poly1 = NULL, *poly2 = NULL, *poly = NULL;
void create(struct link *node)
{
        char ch;
```

```
        do {
                printf("\n enter coeff:");
                scanf("%d", &node->coeff);
                printf("\n enter power:");
                scanf("%d", &node->pow);
                node->next = (struct link *) malloc(sizeof(struct link));
                node = node->next;
                node->next = NULL;
                printf("\n continue(y/n):");
                ch = getch();
        }

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

void show(struct link *node)
{
        while (node->next != NULL)
        {
                printf("%dx^%d", node->coeff, node->pow);
                node = node->next;
                if (node->next != NULL)
                        printf("+");
        }
}

void polyadd(struct link *poly1, struct link *poly2, struct link *poly)
{
        while (poly1->next && poly2->next)
        {
                if (poly1->pow > poly2->pow)
                {
                        poly->pow = poly1->pow;
                        poly->coeff = poly1->coeff;
```

```
                poly1 = poly1->next;
        }
        else if (poly1->pow < poly2->pow)
        {
                poly->pow = poly2->pow;
                poly->coeff = poly2->coeff;
                poly2 = poly2->next;
        }
        else
        {
                poly->pow = poly1->pow;
                poly->coeff = poly1->coeff + poly2->coeff;
                poly1 = poly1->next;
                poly2 = poly2->next;
        }

        poly->next = (struct link *) malloc(sizeof(struct link));
        poly = poly->next;
        poly->next = NULL;
}

while (poly1->next || poly2->next)
{
        if (poly1->next)
        {
                poly->pow = poly1->pow;
                poly->coeff = poly1->coeff;
                poly1 = poly1->next;
        }

        if (poly2->next)
        {
                poly->pow = poly2->pow;
                poly->coeff = poly2->coeff;
```

```
                poly2 = poly2->next;
            }

            poly->next = (struct link *) malloc(sizeof(struct link));
            poly = poly->next;
            poly->next = NULL;
        }
}

main()
{
        char ch;
        do
        {
            poly1 = (struct link *) malloc(sizeof(struct link));
            poly2 = (struct link *) malloc(sizeof(struct link));
            poly = (struct link *) malloc(sizeof(struct link));
            printf("\nenter 1st number:");
            create(poly1);
            printf("\nenter 2nd number:");
            create(poly2);
            printf("\n1st Number:");
            show(poly1);
            printf("\n2nd Number:");
            show(poly2);
            polyadd(poly1, poly2, poly);
            printf("\nAdded polynomial:");
            show(poly);
            printf("\n add two more numbers:");
            ch = getch();
        }
        while (ch == 'y' || ch == 'Y');
}
```

APEC

Output:

Enter 1st number:

Enter coeff: 5

Enter power: 4

Continue(y/n): y

Enter 2st number:

Enter coeff: 6

Enter power: 4

Continue(y/n): n

1st number: 5^4

2nd number: 6^4

Added polynomial: 11^4

APEC

Result:

Thus the C program for implementation of the polynomial manipulation using linked list was executed and verified successfully.

EX. NO: 5A	IMPLEMENTATION OF EVALUATING POSTFIX EXPRESSIONS

Aim:

To write a C program to perform implementation of evaluating postfix expressions.

Algorithm:

1. Create a stack to store operands.
2. Scan the given expression from left to right.
3. a.) If the scanned character is an operand, push it into the stack.

   b) If the scanned character is an operator, POP 2 operands from stack and perform operation and PUSH the result back to the stack.
4. Repeat step 3 till all the characters are scanned.
5. When the expression is ended, the number in the stack is the final result.

Program:

```
#include <stdio.h>
int stack[20];
int top = -1;
void push(int x)
{
        stack[++top] = x;
}
int pop()
{
        return stack[top--];
}
int main()
{
        char exp[20];
        char *e;
        int n1, n2, n3, num;
        printf("Enter the expression :: ");
        scanf("%s", exp);
        e = exp;
        while (*e != '\0')
        {
                if (isdigit(*e))
                {
```

APEC

```
                num = *e - 48;
                push(num);
        }
        else
        {
                n1 = pop();
                n2 = pop();
                switch (*e)
                {
                        case '+':
                                {
                                        n3 = n1 + n2;
                                        break;
                                }

                        case '-':
                                {
                                        n3 = n2 - n1;
                                        break;
                                }

                        case '*':
                                {
                                        n3 = n1 * n2;
                                        break;
                                }

                        case '/':
                                {
                                        n3 = n2 / n1;
                                        break;
                                }
                }

                push(n3);
        }

        e++;
    }
    printf("\nThe result of expression %s  =  %d\n\n", exp, pop());
    return 0;
}
```

APEC

Output:

Enter the expression: 245+*

The result of expression 245+*=18

APEC

Result:

Thus the C program for implementation of evaluating postfix expression was executed and verified successfully.

EX. NO: 5B	IMPLEMENTATION OF INFIX TO POSTFIX CONVERSION

Aim:

To write a C program to convert infix to postfix expression.

Algorithm:

1. Start the program
2. Using the strlen function determine the length of the string
3. Declare three arrays; the decremented value of one array is assigned to the incremented value. This process is repeated until the parenthesis matches.
4. Until t==0 assign the values to sack1
5. Check if (top2==-1) if the condition is satisfied then assign stack2 [++top2]=str[i];
6. Print the result.
7. Stop the program

Program:

```
#include <stdio.h>
#include <conio.h>
#include <string.h>
char str[50], stack1[50], stack2[50];
int i = 0, j, n, top1 = -1, top2 = -1;
int prec(char);
void main()
{
        int t = 0;
        clrscr();
        printf("\nEnter the infix expression:");
        scanf("%s", str);
        printf("%s", str);
        n = strlen(str);
        for (i = 0; i < n; i++)
        {
                t = 0;
```

APEC

```
if (str[i] == ')')
{
        stack1[++top1] = stack2[top2--];
        top2--;
        t = 1;
}

if (str[i] == '(')
{
        stack2[++top2] = '(';
        t = 1;
}

if (t == 0)
{
        if (((str[i] >= 'a') && (str[i] <= 'z')) || ((str[i] >= 'A') && (str[i] <= 'Z')))
                stack1[++top1] = str[i];
        else
        {
                if (top2 == -1)
                        stack2[++top2] = str[i];
                else
                {
                        if (prec(str[i]) > prec(stack2[top2]))
                        {
                                stack2[++top2] = str[i];
                        }
                        else
                        {
                                while (prec(str[i]) <= prec(stack2[top2]))
                                        stack1[++top1] = stack2[top2--];
                                stack2[++top2] = str[i];
                        }
                }
```

APEC

```
                        }
                }
        }

        printf("\nANSWER\n");
        while (top2 >= 0)
                stack1[++top1] = stack2[top2--];
        for (i = 0; i <= top1; i++)
                printf("%c", stack1[i]);
        getch();
}
int prec(char ch)
{
        int tt;
        if ((ch == '+') || (ch == '-'))
                tt = 2;
        if ((ch == '*') || (ch == '/'))
                tt = 3;
        if (ch == '(')
                tt = 1;
        return tt;
}
```

APEC

OutPut:

Enter the infix expression: (a+b)*c/d+e/f

ANSWER: ab+c*d/ef/+

APEC

Result:

Thus the C program for conversion of infix to postfix using stack was executed and verified successfully.

EX. NO: 6	IMPLEMENTATION OF BINARY SEARCH TREES

Aim:

To write a C program for the implementation of a binary search tree.

Algorithm:

1. Declare function create (), search (), delete (), Display ().

2. Create a structure for a tree contains left pointer and right pointer.

3. Insert an element is by checking the top node and the leaf node and the operation will be performed.

4. Deleting an element contains searching the tree and deleting the item.

5. Display the Tree elements.

Program:

```
#include <stdio.h>
#include <conio.h>
#include <process.h>
#include <alloc.h>
struct tree
{
        int data;
        struct tree * lchild;
        struct tree * rchild;
}
*t, *temp;
int element;
void inorder(struct tree *);
void preorder(struct tree *);
void postorder(struct tree *);
struct tree* create(struct tree *, int);
struct tree* find(struct tree *, int);
struct tree* insert(struct tree *, int);
```

APEC

```
struct tree* del(struct tree *, int);
struct tree* findmin(struct tree *);
struct tree* findmax(struct tree *);
void main()
{
	int ch;

	do {
		printf("\n\t\t\tBINARY SEARCH TREE");
		printf("\n\t\t\t****** **********");
		printf("\nMain Menu\n");
		printf("\n1.Create\n2.Insert\n3.Delete\n4.Find\n5.FindMin\n6.FindMax");
		printf("\n7.Inorder\n8.Preorder\n9.Postorder\n10.Exit\n");
		printf("\nEnter ur choice :");
		scanf("%d", &ch);
		switch (ch)
		{
			case 1:
				printf("\nEnter the data:");
				scanf("%d", &element);
				t = create(t, element);
				inorder(t);
				break;
			case 2:
				printf("\nEnter the data:");
				scanf("%d", &element);
				t = insert(t, element);
				inorder(t);
				break;
			case 3:
				printf("\nEnter the data:");
				scanf("%d", &element);
				t = del(t, element);
				inorder(t);
				break;
			case 4:
				printf("\nEnter the data:");
				scanf("%d", &element);
				temp = find(t, element);
				if (temp->data == element)
					printf("\nElement %d is at %d", element, temp);
				else
					printf("\nElement is not found");
				break;
```

APEC

```
                        case 5:
                                temp = findmin(t);
                                printf("\nMax element=%d", temp->data);
                                break;
                        case 6:
                                temp = findmax(t);
                                printf("\nMax element=%d", temp->data);
                                break;
                        case 7:
                                inorder(t);
                                break;
                        case 8:
                                preorder(t);
                                break;
                        case 9:
                                postorder(t);
                                break;
                        case 10:
                                exit(0);
                }
        }
while (ch <= 10);
}

struct tree* create(struct tree *t, int element)
{
        t = (struct tree *) malloc(sizeof(struct tree));
        t->data = element;
        t->lchild = NULL;
        t->rchild = NULL;
        return t;
}

struct tree* find(struct tree *t, int element)
{
        if (t == NULL)
                return NULL;
        if (element < t->data)
                return (find(t->lchild, element));
        else
        if (element > t->data)
                return (find(t->rchild, element));
        else
                return t;
}
```

APEC

```
struct tree* findmin(struct tree *t)
{
	if (t == NULL)
		return NULL;
	else
	if (t->lchild == NULL)
		return t;
	else
		return (findmin(t->lchild));
}

struct tree* findmax(struct tree *t)
{
	if (t != NULL)
	{
		while (t->rchild != NULL)
			t = t->rchild;
	}

	return t;
}

struct tree* insert(struct tree *t, int element)
{
	if (t == NULL)
	{
		t = (struct tree *) malloc(sizeof(struct tree));
		t->data = element;
		t->lchild = NULL;
		t->rchild = NULL;
		return t;
	}
	else
	{
		if (element < t->data)
		{
			t->lchild = insert(t->lchild, element);
		}
		else
		if (element > t->data)
		{
			t->rchild = insert(t->rchild, element);
		}
```

APEC

```
                else
                if (element == t->data)
                {
                        printf("element already present\n");
                }

                return t;
        }
}

struct tree* del(struct tree *t, int element)
{
        if (t == NULL)
                printf("element not found\n");
        else
        if (element < t->data)
                t->lchild = del(t->lchild, element);
        else
        if (element > t->data)
                t->rchild = del(t->rchild, element);
        else
        if (t->lchild && t->rchild)
        {
                temp = findmin(t->rchild);
                t->data = temp->data;
                t->rchild = del(t->rchild, t->data);
        }
        else
        {
                temp = t;
                if (t->lchild == NULL)
                        t = t->rchild;
                else
                if (t->rchild == NULL)
                        t = t->lchild;
                free(temp);
        }

        return t;
}
```

APEC

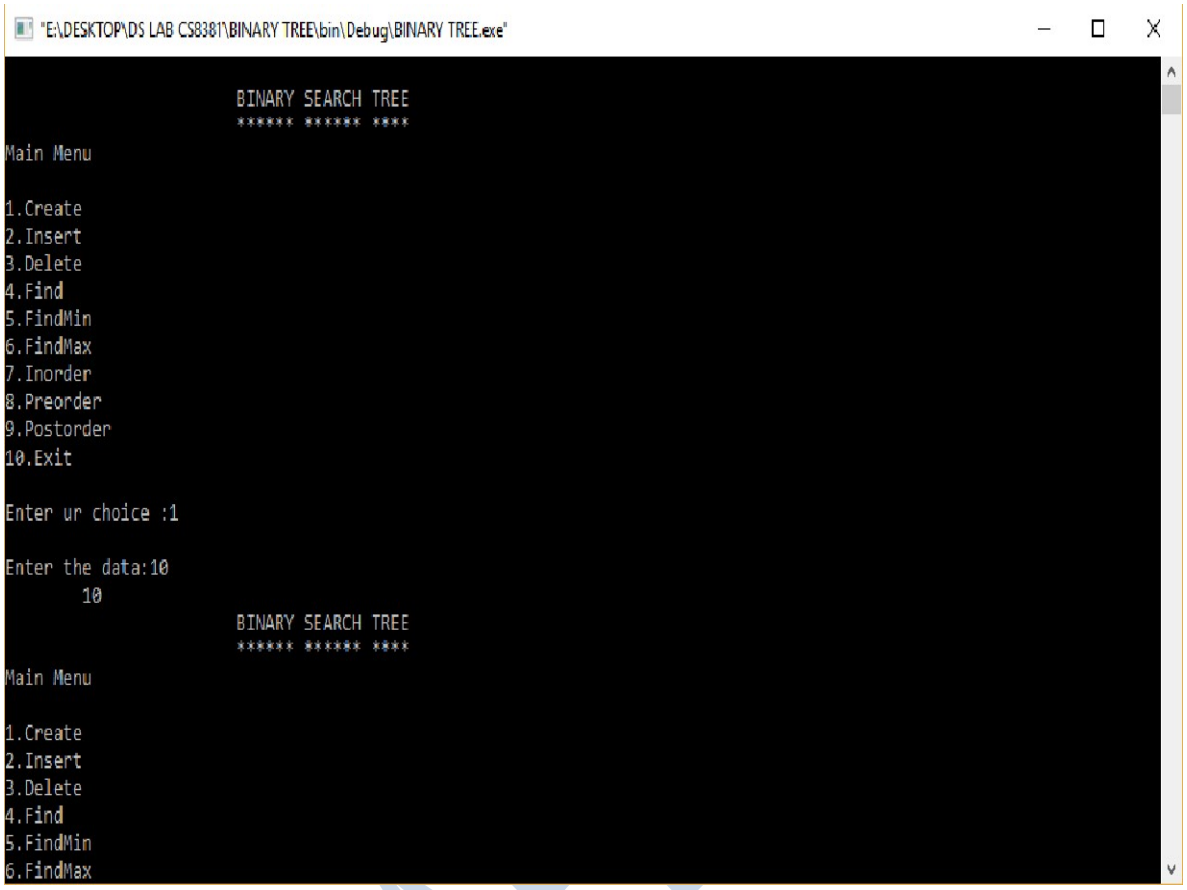
```
void inorder(struct tree *t)
{
        if (t == NULL)
                return;
        else
        {
                inorder(t->lchild);
                printf("\t%d", t->data);
                inorder(t->rchild);
        }
}

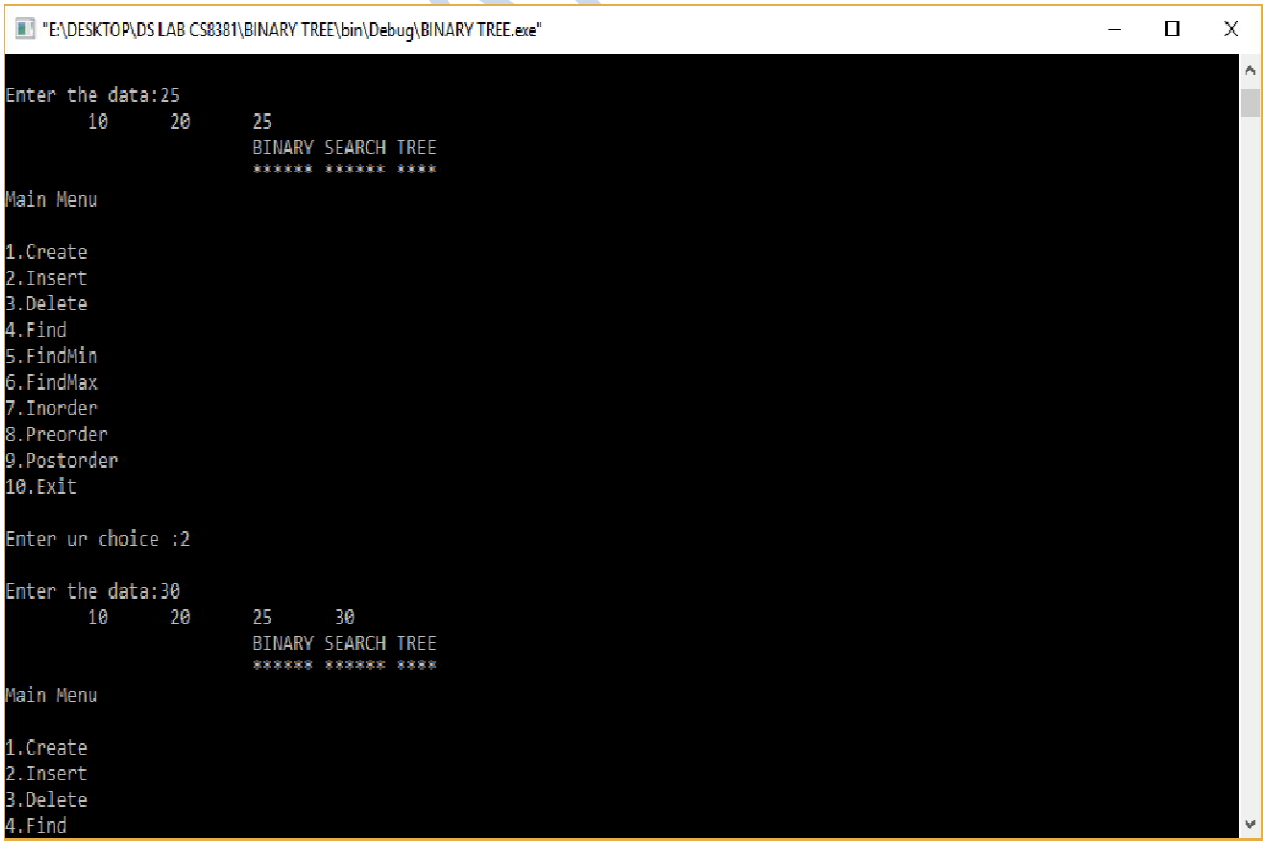
void preorder(struct tree *t)
{
        if (t == NULL)
                return;
        else
        {
                printf("\t%d", t->data);
                preorder(t->lchild);
                preorder(t->rchild);
        }
}

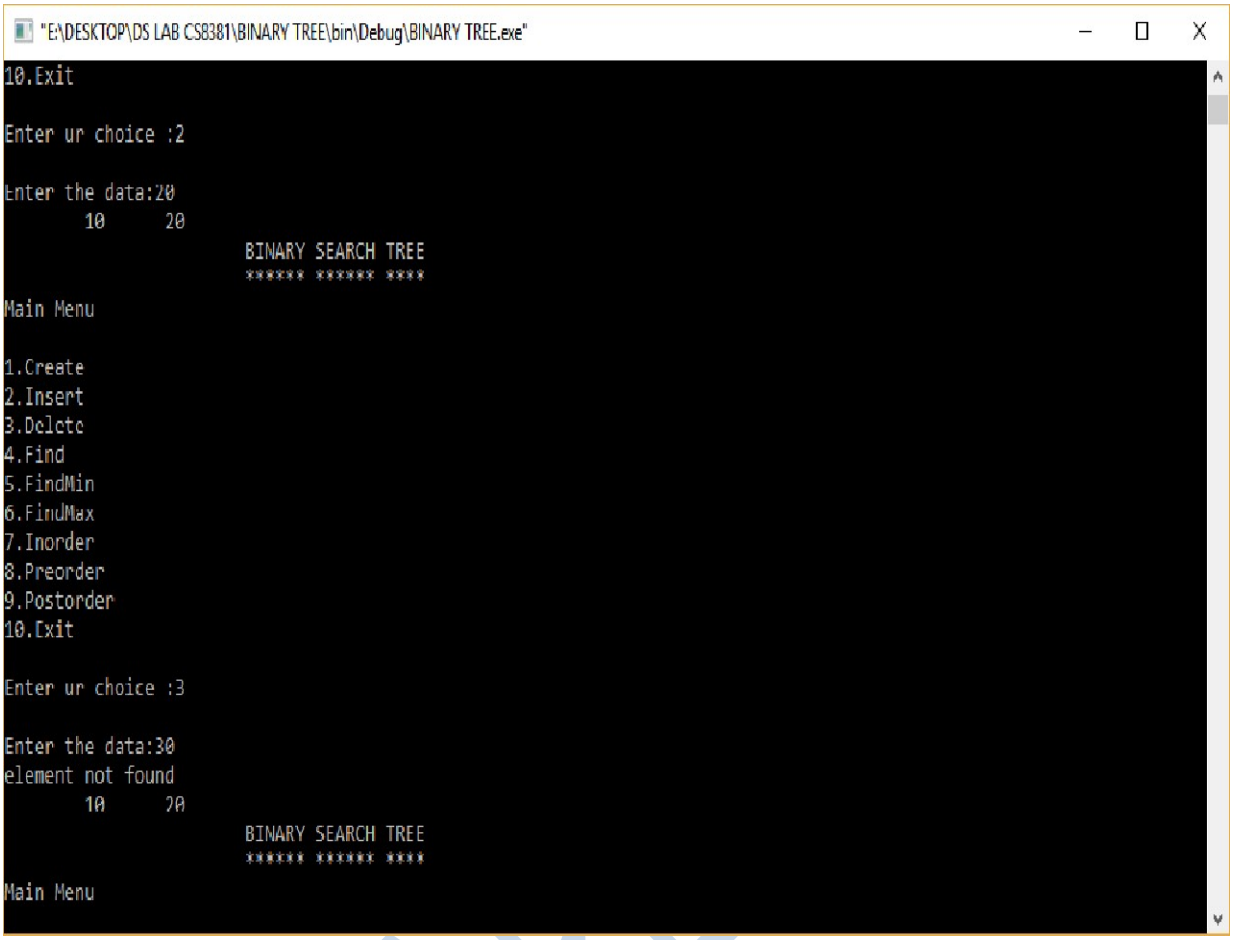
void postorder(struct tree *t)
{
        if (t == NULL)
                return;
        else
        {
                postorder(t->lchild);
                postorder(t->rchild);
                printf("\t%d", t->data);
        }
}
```

APEC

Output:


"E:\DESKTOP\DS LAB CS8381\BINARY TREE\bin\Debug\BINARY TREE.exe"
BINARY SEARCH TREE
****** ****** ****
Main Menu
1.Create
2.Insert
3.Delete
4.Find
5.FindMin
6.FindMax
7.Inorder
8.Preorder
9.Postorder
10.Exit
Enter ur choice :1
Enter the data:10
10
BINARY SEARCH TREE
****** ****** ****
Main Menu
1.Create
2.Insert
3.Delete
4.Find
5.FindMin
6.FindMax


"E:\DESKTOP\DS LAB CS8381\BINARY TREE\bin\Debug\BINARY TREE.exe"
Enter the data:25
10 20 25
BINARY SEARCH TREE
****** ****** ****
Main Menu
1.Create
2.Insert
3.Delete
4.Find
5.FindMin
6.FindMax
7.Inorder
8.Preorder
9.Postorder
10.Exit
Enter ur choice :2
Enter the data:30
10 20 25 30
BINARY SEARCH TREE
****** ****** ****
Main Menu
1.Create
2.Insert
3.Delete
4.Find



Result:

Thus the C program for implementation of binary search tree was executed and verified successfully.

EX. NO: 7	IMPLEMENTATION OF AVL TREES

Aim:

To write a C program to implement insertion in AVL trees.

Algorithm:

1. Initialize all variables and functions.
2. To insert and element read the value.
3. Check whether root is null
4. If yes make the new value as root.
5. Else check whether the value is equal to root value.
6. If yes, print “duplicate value”.
7. Otherwise insert the value at its proper position and balance the tree using rotations.
8. To display the tree values check whether the tree is null.
9. If yes, print “tree is empty”.
10. Else print all the values in the tree form and in order of the tree.
11. Repeat the steps 2 to 10 for more values.
12. End.

Program:

```
#include <stdio.h>
#include <malloc.h>
typedef enum
{
        FALSE, TRUE
}
bool;
struct node
{
        int info;
        int balance;
        struct node * lchild;
        struct node * rchild;
};
```

APEC

```
struct node* insert(int, struct node *, int *);
struct node* search(struct node *, int);
main()
{
        bool ht_inc;
        int info;
        int choice;
        struct node *root = (struct node *) malloc(sizeof(struct node));
        root = NULL;
        while (1)
        {
                printf("1.Insert\n");
                printf("2.Display\n");
                printf("3.Quit\n");
                printf("Enter your choice : ");
                scanf("%d", &choice);
                switch (choice)
                {
                        case 1:
                                printf("Enter the value to be inserted : ");
                                scanf("%d", &info);
                                if (search(root, info) == NULL)
                                        root = insert(info, root, &ht_inc);
                                else
                                        printf("Duplicate value ignored\n");
                                break;
                        case 2:
                                if (root == NULL)
                                {
                                        printf("Tree is empty\n");
                                        continue;
                                }

                                printf("Tree is :\n");
```

APEC

```
                        display(root, 1);
                        printf("\n\n");
                        printf("Inorder Traversal is: ");
                        inorder(root);
                        printf("\n");
                        break;
                case 3:
                        exit(1);
                default:
                        printf("Wrong choice\n");
            } /*End of switch*/
        } /*End of while*/
} /*End of main()*/
struct node* search(struct node *ptr, int info)
{
        if (ptr != NULL)
            if (info < ptr->info)
                ptr = search(ptr->lchild, info);
            else if (info > ptr->info)
            ptr = search(ptr->rchild, info);
        return (ptr);
} /*End of search()*/
struct node* insert(int info, struct node *pptr, int *ht_inc)
{
        struct node * aptr;
        struct node * bptr;
        if (pptr == NULL)
        {
            pptr = (struct node *) malloc(sizeof(struct node));
            pptr->info = info;
            pptr->lchild = NULL;
            pptr->rchild = NULL;
            pptr->balance = 0;
            *ht_inc = TRUE;
            return (pptr);
        }
```

APEC

```
if (info < pptr->info)
{
        pptr->lchild = insert(info, pptr->lchild, ht_inc);
        if (*ht_inc == TRUE)
        {
                switch (pptr->balance)
                {
                        case -1:
                                /*Right heavy */
                                pptr->balance = 0;
                                *ht_inc = FALSE;
                                break;
                        case 0:
                                /*Balanced */
                                pptr->balance = 1;
                                break;
                        case 1:
                                /*Left heavy */
                                aptr = pptr->lchild;
                                if (aptr->balance == 1)
                                {
                                        printf("Left to Left Rotation\n");
                                        pptr->lchild = aptr->rchild;
                                        aptr->rchild = pptr;
                                        pptr->balance = 0;
                                        aptr->balance = 0;
                                        pptr = aptr;
                                }
                                else
                                {
                                        printf("Left to right rotation\n");
                                        bptr = aptr->rchild;
                                        aptr->rchild = bptr->lchild;
```

```
                                        bptr->lchild = aptr;
                                        pptr->lchild = bptr->rchild;
                                        bptr->rchild = pptr;
                                        if (bptr->balance == 1)
                                                pptr->balance = -1;
                                        else
                                                pptr->balance = 0;
                                        if (bptr->balance == -1)
                                                aptr->balance = 1;
                                        else
                                                aptr->balance = 0;
                                        bptr->balance = 0;
                                        pptr = bptr;
                                }

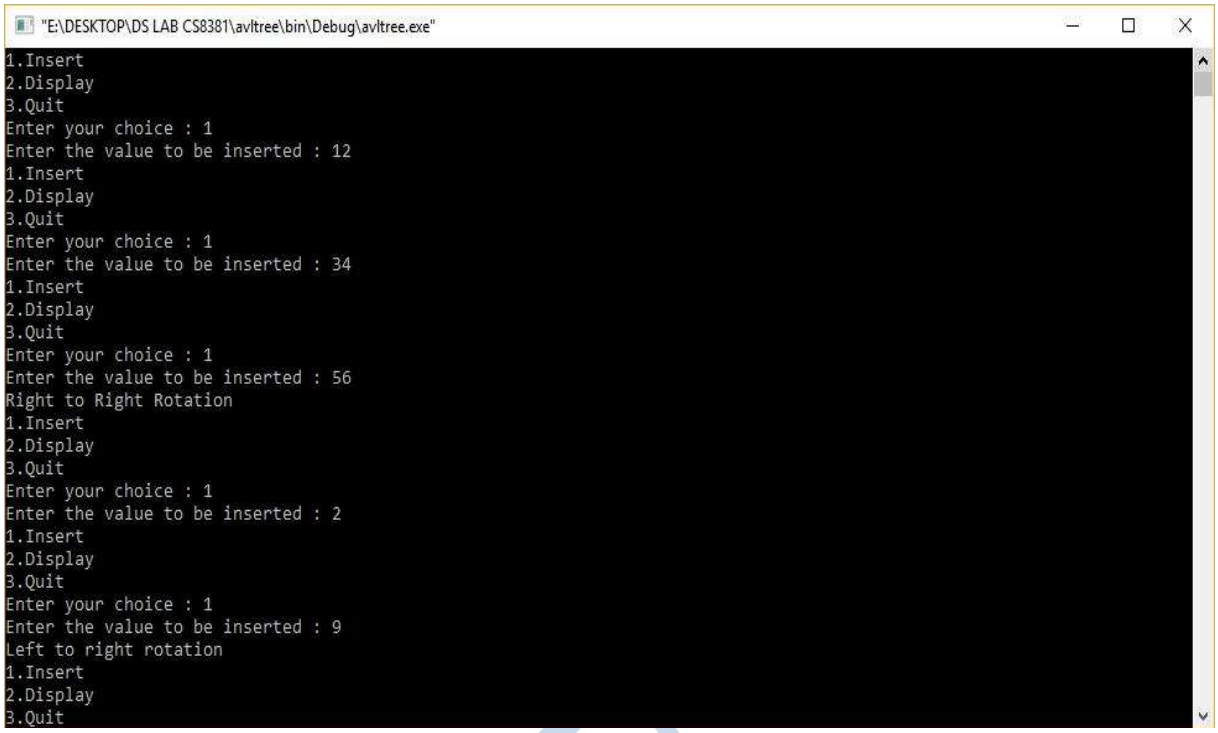
                                *ht_inc = FALSE;
                        } /*End of switch */
                } /*End of if */
        } /*End of if*/
        if (info > pptr->info)
        {
                pptr->rchild = insert(info, pptr->rchild, ht_inc);
                if (*ht_inc == TRUE)
                {
                        switch (pptr->balance)
                        {
                                case 1:
                                        /*Left heavy */
                                        pptr->balance = 0;
                                        *ht_inc = FALSE;
                                        break;
                                case 0:
                                        /*Balanced */
                                        pptr->balance = -1;
```

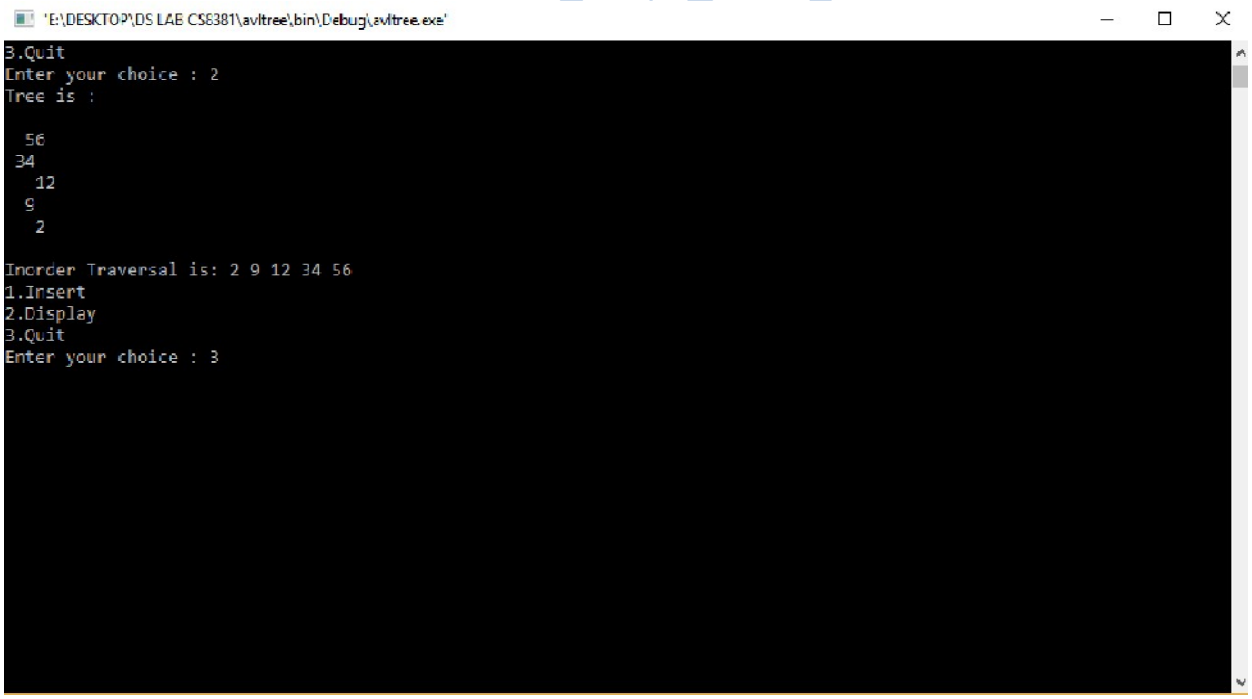
```
                break;
        case -1:
                /*Right heavy */
                aptr = pptr->rchild;
                if (aptr->balance == -1)
                {
                        printf("Right to Right Rotation\n");
                        pptr->rchild = aptr->lchild;
                        aptr->lchild = pptr;
                        pptr->balance = 0;
                        aptr->balance = 0;
                        pptr = aptr;
                }
                else
                {
                        printf("Right to Left Rotation\n");
                        bptr = aptr->lchild;
                        aptr->lchild = bptr->rchild;
                        bptr->rchild = aptr;
                        pptr->rchild = bptr->lchild;
                        bptr->lchild = pptr;
                        if (bptr->balance == -1)
                                pptr->balance = 1;
                        else
                                pptr->balance = 0;
                        if (bptr->balance == 1)
                                aptr->balance = -1;
                        else
                                aptr->balance = 0;
                        bptr->balance = 0;
                        pptr = bptr;
                } /*End of else*/
                *ht_inc = FALSE;
} /*End of switch */
```

```
                } /*End of if*/
        } /*End of if*/
        return (pptr);
} /*End of insert()*/
display(struct node *ptr, int level)
{
        int i;
        if (ptr != NULL)
        {
                display(ptr->rchild, level + 1);
                printf("\n");
                for (i = 0; i < level; i++)
                        printf(" ");
                printf("%d", ptr->info);
                display(ptr->lchild, level + 1);
        } /*End of if*/
} /*End of display()*/
inorder(struct node *ptr)
{
        if (ptr != NULL)
        {
                inorder(ptr->lchild);
                printf("%d ", ptr->info);
                inorder(ptr->rchild);
        }
} /*End of inorder()*/
```

APEC

Output:





Result:

Thus the C program to implement an avl trees produce its pre-sequence, in-sequence, and post-sequence traversals was executed and verified successfully.

EX. NO: 8	IMPLEMENTATION OF HEAPS USING PRIORITY QUEUES

Aim:

To write a C program to implement Priority Queue using Binary Heaps.

Algorithm:

1. Initialize all necessary variables and functions.
2. Read the choices.
3. For insertion, read the element to be inserted.
4. If root is NULL, assign the given element as root.
5. If the element is equal to the root, print “Duplicate value”.
6. Else if element value is less than the root value, insert element at the left of the root.
7. Else insert right side of the root.
8. For deletion, get the priority for maximum or minimum.
9. If maximum, it deletes the root and rearranges the tree.
10. If minimum, it deletes the leaf.
11. End of the program.

Program:

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
enum
{
        FALSE = 0, TRUE = -1
};
struct Node
{
        struct Node * Previous;
        int Data;
        struct Node * Next;
}
```

APEC

```
Current;
struct Node * head;
struct Node * ptr;
static int NumOfNodes;
int PriorityQueue(void);
int Maximum(void);
int Minimum(void);
void Insert(int);
int Delete(int);
void Display(void);
int Search(int);
void main()
{
        int choice;
        int DT;
        PriorityQueue();
        while (1)
        {
                printf("\nEnter ur Choice:");
                printf("\n1.Insert\n2.Display\n3.Delete\n4.Search\n5.Exit\n");
                scanf("%d", &choice);
                switch (choice)
                {
                        case 1:
                                printf("\nEnter a data to enter Queue");

                                scanf("%d", &DT);
                                Insert(DT);
                                break;
                        case 2:
                                Display();
                                break;
                        case 3:
                                {
                                        int choice, DataDel;
                                        printf("\nEnter ur choice:");

                                        printf("\n1.Maximum Priority queue\n2.Minimum
                                        priority Queue\n");
                                        scanf("%d", &choice);
```

APEC

```
                switch (choice)
                {
                        case 1:
                                DataDel = Maximum();
                                Delete(DataDel);

                                printf("\n%d is deleted\n", DataDel);
                                break;

                        case 2:
                                DataDel = Minimum();
                                Delete(DataDel);
                                printf("\n%d is deleted\n", DataDel);
                                break;
                        default:

                                printf("\nSorry Not a correct Choice\n");
                }
        }

        break;
case 4:

        printf("\nEnter a data to Search in Queue:");
        scanf("%d", &DT);

        if (Search(DT) != FALSE)
                printf("\n %d is present in queue", DT);
        else

                printf("\n%d is not present in queue", DT);
        break;
case 5:
        exit(0);
default:

        printf("\nCannot process ur choice\n");
    }
  }
}
```

APEC

```
int PriorityQueue(void)
{
        Current.Previous = NULL;

        printf("\nEnter first element of Queue:");
        scanf("%d", &Current.Data);

        Current.Next = NULL;
        head = &Current;
        ptr = head;

        NumOfNodes++;
        return;
}
int Maximum(void)
{
        int Temp;
        ptr = head;
        Temp = ptr->Data;
        while (ptr->Next != NULL)

        {
                if (ptr->Data > Temp)

                        Temp = ptr->Data;
                ptr = ptr->Next;
        }

        if (ptr->Next == NULL && ptr->Data > Temp)
                Temp = ptr->Data;
        return (Temp);
}
```

APEC

```
int Minimum(void)
{
        int Temp;
        ptr = head;
        Temp = ptr->Data;
        while (ptr->Next != NULL)
        {
                if (ptr->Data < Temp)
                        Temp = ptr->Data;

                ptr = ptr->Next;
        }
        if (ptr->Next == NULL && ptr->Data < Temp)
                Temp = ptr->Data;
        return (Temp);
}
void Insert(int DT)
{
        struct Node * newnode;

        newnode = (struct Node *) malloc(sizeof(struct Node));
        newnode->Next = NULL;
        newnode->Data = DT;

        while (ptr->Next != NULL)
                ptr = ptr->Next;
        if (ptr->Next == NULL)
        {
                newnode->Next = ptr->Next;
                ptr->Next = newnode;
        }
        NumOfNodes++;
}
```

APEC

```
int Delete(int DataDel)
{
        struct Node *mynode, *temp;
        ptr = head;
        if (ptr->Data == DataDel)
        {
                temp = ptr;

                ptr = ptr->Next;
                ptr->Previous = NULL;
                head = ptr;
                NumOfNodes--;
                return (TRUE);
        }
        else
        {
                while (ptr->Next->Next != NULL)
                {
                        if (ptr->Next->Data == DataDel)
                        {
                                mynode = ptr;
                                temp = ptr->Next;
                                mynode->Next = mynode->Next->Next;
                                mynode->Next->Previous = ptr;
                                free(temp);
                                NumOfNodes--;
                                return (TRUE);
                        }
                        ptr = ptr->Next;
                }
                if (ptr->Next->Next == NULL && ptr->Next->Data == DataDel)
                {
                        temp = ptr->Next;
                        free(temp);
```

APEC

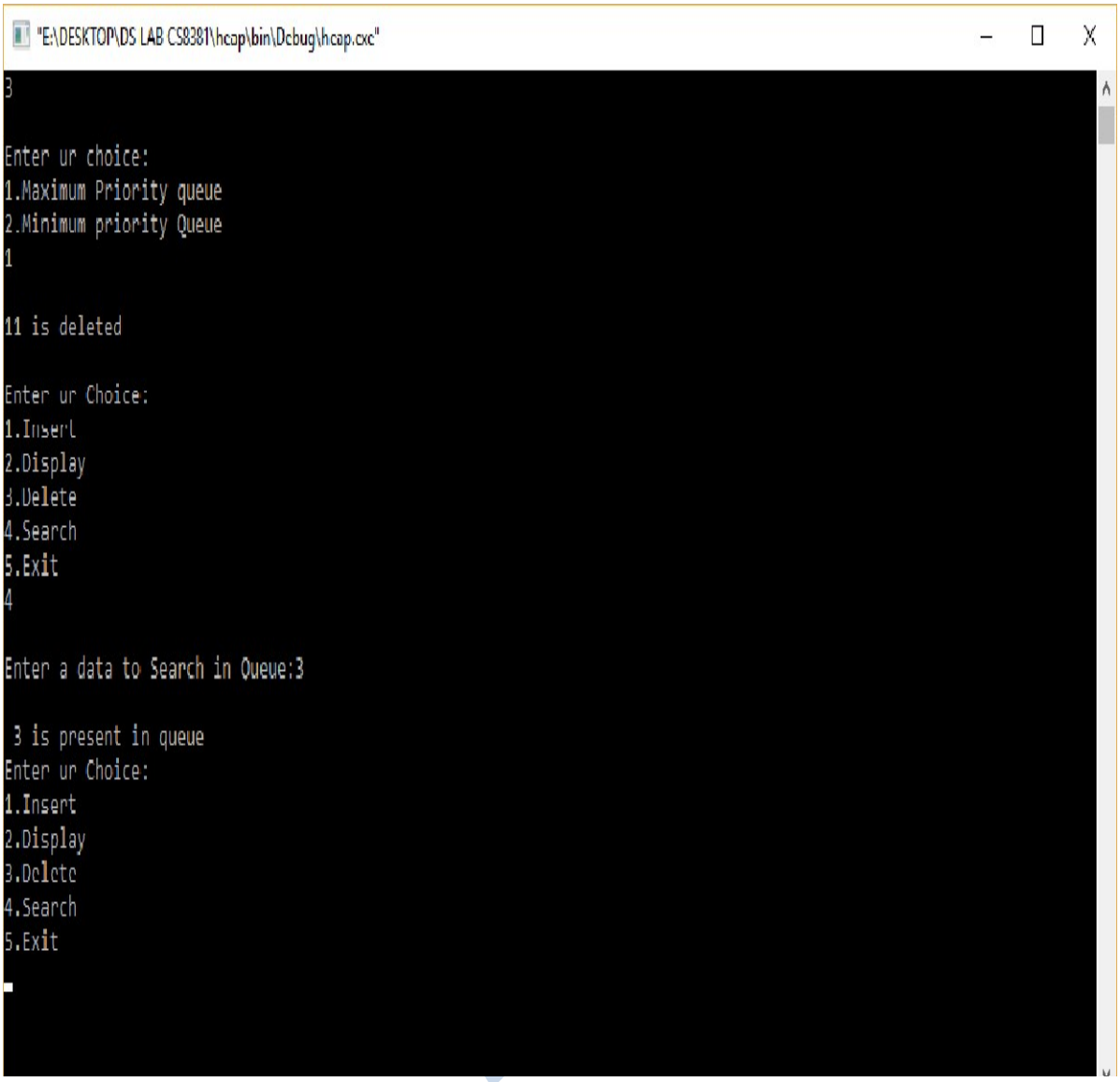
```
                        ptr->Next = NULL;
                        NumOfNodes--;
                        return (TRUE);
                }
        }
        return (FALSE);
}
int Search(int DataSearch)
{
        ptr = head;
        while (ptr->Next != NULL)
        {
                if (ptr->Data == DataSearch)
                        return ptr->Data;
                ptr = ptr->Next;
        }
        if (ptr->Next == NULL && ptr->Data == DataSearch)

                return ptr->Data;
        return (FALSE);
}
void Display(void)
{
        ptr = head;
        printf("\nPriority Queue is as Follows:-\n");
        while (ptr != NULL)
        {
                printf("\t\t%d", ptr->Data);
                ptr = ptr->Next;
        }
}
```

APEC

Output:


"E:\DESKTOP\DS LAB CS8381\heap\bin\Debug\heap.exe"
Enter first element of Queue:3
Enter ur Choice:
1.Insert
2.Display
3.Delete
4.Search
5.Exit
1
Enter a data to enter Queue 11
Enter ur Choice:
1.Insert
2.Display
3.Delete
4.Search
5.Exit
2
Priority Queue is as Follows:-
3 11
Enter ur Choice:
1.Insert
2.Display
3.Delete
4.Search
5.Exit
3



Result:

Thus the C program for Implementation of Heaps using Priority Queues was executed and verified successfully.

EX. NO: 9	IMPLEMENTATION OF DIJKSTRA'S ALGORITHM

Aim:

To write a C program for the implementation of Dijktra's algorithm.

Algorithm:

1. Create a set short Path to store vertices that come in the way of the shortest path tree.
2. Initialize all distance values as INFINITE and assign distance values as 0 for source vertex so that it is picked first.
3. Loop until all vertices of the graph are in the short Path.
4. Take a new vertex that is not visited and is nearest.
5. Add this vertex to short Path.
6. For all adjacent vertices of this vertex update distances. Now check every adjacent vertex of V, if sum of distance of u and weight of edge is else the update it.

Program:

```
#include <limits.h>
#include <stdio.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");
                for (int i = 0; i < V; i++)
                printf("%d \t %d", 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, 6, 0, 0, 0, 0, 0, 8, 0},
                                { 6, 0, 8, 0, 0, 0, 0, 13, 0 },
                                { 0, 8, 0, 7, 0, 6, 0, 0, 2 },
                                { 0, 0, 7, 0, 9, 14, 0, 0, 0 },
                                { 0, 0, 0, 9, 0, 10, 0, 0, 0 },
                                { 0, 0, 6, 14, 10, 0, 2, 0, 0 },
                                { 0, 0, 0, 0, 0, 2, 0, 1, 6 },
                                { 8, 13, 0, 0, 0, 0, 1, 0, 7 },
                                { 0, 0, 2, 0, 0, 0, 6, 7, 0 }
                        };

                dijkstra(graph, 0);
                return 0;
}
```

AEC

Output:

Vertex Distance from Source

0 0

1 6

2 14

3 21

4 21

5 11

6 9

7 8

8 15

APEC

Result:

Thus the C program for implementation of dijkstra's algorithm was executed and verified successfully.

EX. NO: 10	IMPLEMENTATION OF PRIM'S ALGORITHM

Aim:

To write a c program to implement a prim's algorithm.

Algorithm:

This algorithm creates spanning tree with minimum weight from a given weighted graph.

1. Begin
2. Create edge list of given graph, with their weights.
3. Draw all nodes to create skeleton for spanning tree.
4. Select an edge with lowest weight and add it to skeleton and delete edge from edge list.
5. Add other edges. While adding an edge take care that the one end of the edge should always be in the skeleton tree and its cost should be minimum.
6. Repeat step 5 until n-1 edges are added.
7. Return.

Program:

```
#include <stdio.h>
#include <stdlib.h>
#define infinity 9999
#define MAX 20
int G[MAX][MAX], spanning[MAX][MAX], n;
int prims();
int main()
{
    int i, j, total_cost;
    printf("Enter no. of vertices:");
    scanf("%d", &n);
    printf("\nEnter the adjacency matrix:\n");
```

APEC

```
        for (i = 0; i < n; i++)
                for (j = 0; j < n; j++)
                        scanf("%d", &G[i][j]);
        total_cost = prims();
        printf("\nspanning tree matrix:\n");
        for (i = 0; i < n; i++)
        {
                printf("\n");
                for (j = 0; j < n; j++)
                        printf("%d\t", spanning[i][j]);
        }
        printf("\n\nTotal cost of spanning tree=%d", total_cost);
        return 0;
}
int prims()
{
        int cost[MAX][MAX];
        int u, v, min_distance, distance[MAX], from[MAX];
        int visited[MAX], no_of_edges, i, min_cost, j;
        //create cost[][] matrix,spanning[][]
        for (i = 0; i < n; i++)
        for (j = 0; j < n; j++)
        {
                if (G[i][j] == 0)
                        cost[i][j] = infinity;
                else
                        cost[i][j] = G[i][j];
```

APEC

```
                spanning[i][j] = 0;
}
//initialise visited[],distance[] and from[]
distance[0] = 0;
visited[0] = 1;
for (i = 1; i < n; i++)
{
        distance[i] = cost[0][i];
        from[i] = 0;
        visited[i] = 0;
}
min_cost = 0;  //cost of spanning tree
no_of_edges = n - 1;   //no. of edges to be added
while (no_of_edges > 0)
{
        //find the vertex at minimum distance from the tree
        min_distance = infinity;
        for (i = 1; i < n; i++)
                if (visited[i] == 0 && distance[i] < min_distance)
                {
                        v = i;
                        min_distance = distance[i];
                }
        u = from[v];
        //insert the edge in spanning tree
        spanning[u][v] = distance[v];
        spanning[v][u] = distance[v];
```

APEC

```
            no_of_edges--;
            visited[v] = 1;
            //updated the distance[] array
            for (i = 1; i < n; i++)
                  if (visited[i] == 0 && cost[i][v] < distance[i])
                  {
                        distance[i] = cost[i][v];
                        from[i] = v;
                  }
            min_cost = min_cost + cost[u][v];
      }
      return (min_cost);
}
```

APEC

Output:

Enter no. of vertices:6

Enter the adjacency matrix:

0 3 1 6 0 0

3 0 5 0 3 0

1 5 0 5 6 4

6 0 5 0 0 2

0 3 6 0 0 6

0 0 4 2 6 0

spanning tree matrix:

0 3 1 0 0 0

3 0 0 0 3 0

1 0 0 0 0 4

0 0 0 0 0 2

0 3 0 0 0 0

0 0 4 2 0 0

Total cost of spanning tree=13

Result:

Thus the C program for implementation of prim's algorithm was executed and verified successfully

EX. NO: 11	IMPLEMENTATION OF LINEAR SEARCH AND BINARY SEARCH

Aim:

To Write a C Program to perform linear search and binary search.

Algorithm:

Linear Search

1. Read n numbers and search value.
2. If search value is equal to first element then print value is found.
3. Else search with the second element and so on.

Binary Search

1. Read n numbers and search value.
2. If search value is equal to middle element then print value is found.
3. If search value is less than middle element then search left half of list with the same method.
4. Else search right half of list with the same method.

Program:

Linear Search:

```
#include <stdio.h > int main()
{
        int array[100], search, c, n;
        printf("Enter the number of elements in array\n");
        scanf("%d", &n);
        printf("Enter %d integer(s)\n", n);
        for (c = 0; c < n; c++)
                scanf("%d", &array[c]);
        printf("Enter the number to search\n");
        scanf("%d", &search);
        for (c = 0; c < n; c++)
        {
```

APEC

```
            if (array[c] == search) /*if required element found */
            {
                printf("%d is present at location %d.\n", search, c + 1);
                break;
            }
        }
        if (c == n)
        printf("%d is not present in array.\n", search);
        return 0;
}
```

Binary Search:

```
#include <stdio.h>
int main()
{
        int a[10], i, n, m, c = 0, l, u, mid;
        printf("Enter the size of an array: ");
        scanf("%d", &n);
        printf("Enter the elements in ascending order: ");
        for (i = 0; i < n; i++)
        {
            scanf("%d", &a[i]);
        }
        printf("Enter the number to be search: ");
        scanf("%d", &m);
        l = 0, u = n - 1;
        while (l <= u)
        {
            mid = (l + u) / 2;
            if (m == a[mid])
            {
                c = 1;
                break;
            }
```

APEC

```
            else if (m < a[mid]) {}
            else
        }
        u = mid - 1;
        l = mid + 1;
        if (c == 0)
            printf("The number is not found.");
        else
}
printf("The number is found.");
return 0;
```

APEC

Output:

Linear Search:

Enter the number of elements in array : 52 5 4 6 34

Enter the number to search : 6

is present at location 4

Binary Search:

Enter the size of an array: 5

Enter the elements in ascending order: 2 3 67 69 85

Enter the number to be search: 67

The number is found

APEC

Result:

Thus the C program for implementation of linear search and binary search was executed and verified successfully.

EX. NO: 12	IMPLEMENTATION OF INSERTION SORT AND SELECTION SORT

Aim:

To write a C program to perform insertion sort and selection sort.

Algorithm:

Insertion sort:

1. If the element is the first element, assume that it is already sorted. Return 1.
2. Pick the next element, and store it separately in a key.
3. Now, compare the key with all elements in the sorted array.
4. If the element in the sorted array is smaller than the current element, then move to the next element. Else, shift greater elements in the array towards the right.
5. Insert the value.
6. Repeat until the array is sorted.

Selection Sort:

1. Set min to the first location.
2. Search the minimum element in the array.
3. Swap the first location with the minimum value in the array.
4. Assign the second element as min.
5. Repeat the process until we get a sorted array.

Program:

Insertion sort:

```
#include <stdio.h>
#include <conio.h>
void main()
{
        int A[10], n, i;
        void Insert_sort(int A[10], int n);
        clrscr();
        printf("\n\ t\ t Insertion sort");
```

```
        printf("\n How many elements are there ? ");
        scanf(" % d", &n);
        printf("\n Enter the elements\ n");
        for (i = 0; i < n; i++)
        scanf(" % d", &A[i]);
        Insert_sort(A, n);
        getch();
}
Void Insert_sort(int A[10], int n)
{
        int i, j, temp;
        for (i = 0; i < n - 1; i++)
        {
                temp = A[i];
                j = i - 1;
                while ((j >= 0) && (A[j] > temp))
                {
                        A[j + 1] = A[j];
                        j = j - 1;
                }
                A[j + 1] = temp;
        }
        printf("\n The Sorted list of elements is….\n");
        for (i = 0; i < n; i++)
                printf("\n % d", A[i]);
}
```

Selection Sort:

```
#include <stdio.h>
#include <conio.h>
int n;
void main()
{
        int i, A[10];
```

```
        void selection(int A[10]);
        clrscr();
        printf("\n\ t\ t selection sort");
        printf("\n How many elements are there ? ");
        scanf(" % d", &n);
        printf("\n Enter the elements\ n");
        for (i = 0; i < n; i++)
        scanf(" % d", &A[i]);
        seection(A);
        getch();
}
void selection(int A[10])
{
        int i, j, min, temp;
        for (i = 0; i < n - 2; i++)
        {
                Min = i;
                for (j = i + 1; j <= n - 1; j++)
                {
                        if (A[j] < A[Min])
                                Min = j;
                }
                temp = A[i];
                A[i] = A[Min];
                A[Min] = temp;
        }
        printf("\n The Sorted list is….\n");
        for (i = 0; i < n; i++)
        printf("\n % d", A[i]);
}
```

APEC

Output:

Insertion sort:

How many elements are there? 5

Enter the elements:

30
20
10
40
50

The sorted list of elements is…

10
20
30
40
50

Selection Sort:

How many elements are there? 7

Enter the elements:

70
30
20
50
60
10
40

The sorted list is…

10 20 30 40 50 60 70

Result:

Thus the C program for implementation of Insertion sort and selection sort was executed and verified successfully.

APEC

EX. NO: 13	IMPLEMENTATION OF MERGE SORT

Aim:

To write a C program to implement the concept of merge sort.

Algorithm:

1. Start.
2. First you divide the number of elements by 2 and separate them as two.
3. Divide those two which are divided by 2.
4. Divide them until you get a single element.
5. Start comparing the starting two pair of elements with each other and place them in ascending order.
6. When you combine them compare them so that you make sure they are sorted.
7. When all the elements are compared the array will be surely sorted in an ascending order.
8. Stop.

Program:

```
#include <stdio.h>
#include <conio.h>
void merge(int[], int, int, int);
void part(int[], int, int);
void main()
{
        int arr[30];
        int i, size;
        printf("\n\t------- Merge sorting method -------\n\n");
        printf("Enter total no. of elements : ");
        scanf("%d", &size);
        for (i = 0; i < size; i++)
        {
                printf("Enter %d element : ", i + 1);
```

APEC

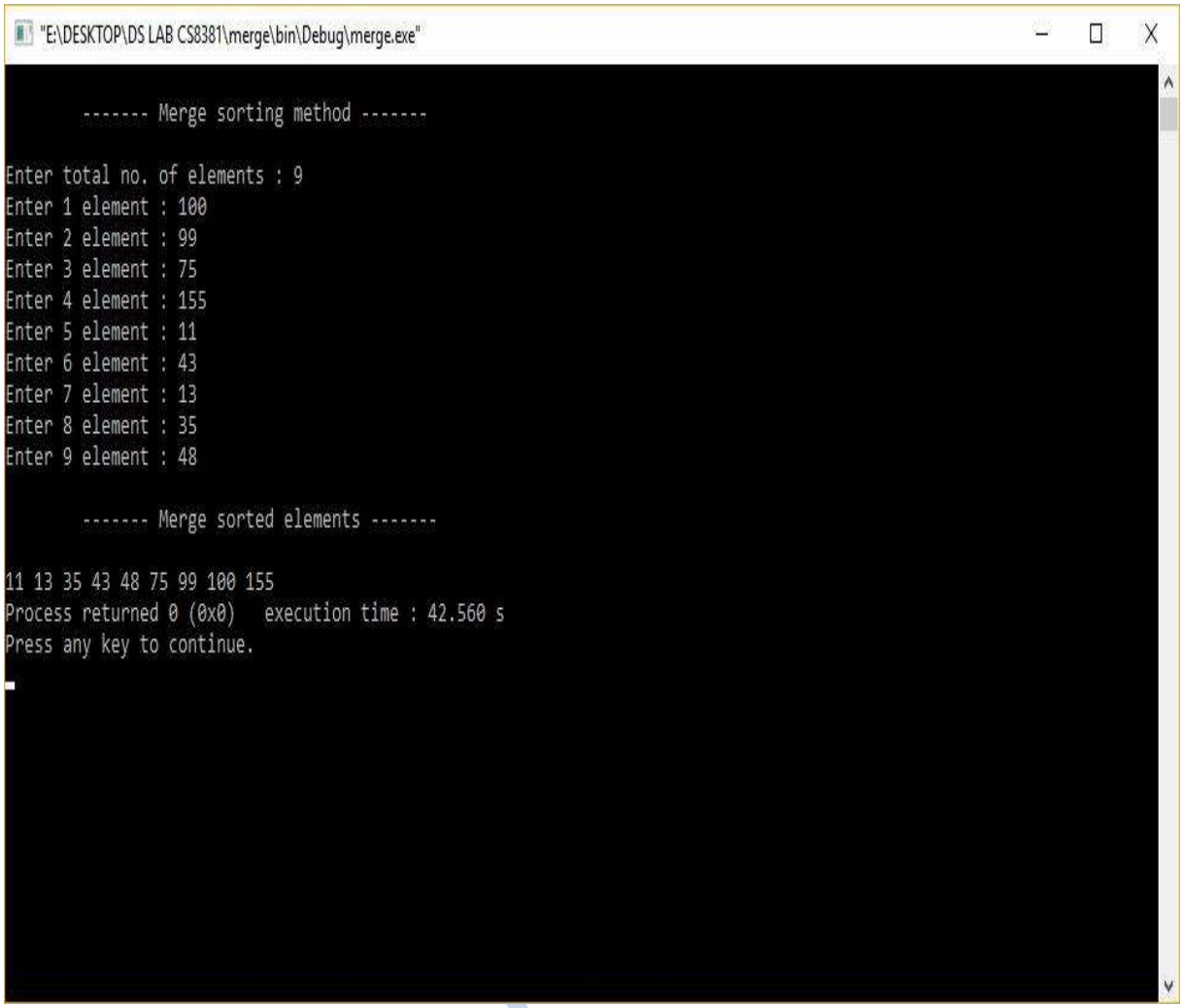
```
                scanf("%d", &arr[i]);
        }
        part(arr, 0, size - 1);
        printf("\n\t------- Merge sorted elements -------\n\n");
        for (i = 0; i < size; i++)
        printf("%d ", arr[i]);
        getch();
}
void part(int arr[], int min, int max)
{
        int mid;
        if (min < max)
        {
                mid = (min + max) / 2;
                part(arr, min, mid);
                part(arr, mid + 1, max);
                merge(arr, min, mid, max);
        }
}
void merge(int arr[], int min, int mid, int max)
{
        int tmp[30];
        int i, j, k, m;
        j = min;
        m = mid + 1;
        for (i = min; j <= mid && m <= max; i++)
        {
                if (arr[j] <= arr[m])
                {
                        tmp[i] = arr[j];
                        j++;
                }
```

APEC

```
            else
            {
                tmp[i] = arr[m];
                m++;
            }
        }
        if (j > mid)
        {
            for (k = m; k <= max; k++)
            {
                tmp[i] = arr[k];
                i++;
            }
        }
        else
        {
            for (k = j; k <= mid; k++)
            {
                tmp[i] = arr[k];
                i++;
            }
        }
        for (k = min; k <= max; k++)
            arr[k] = tmp[k];
}
```

APEC

Output:



Result:

Thus The C program for the concept of merge sort was implemented successfully.

EX. NO: 14	IMPLEMENTATION OF OPEN ADDRESSING (LINEAR PROBING AND QUADRATIC PROBING)

Aim:

To write a C program to implement open addressing (linear probing and quadratic probing)

Algorithm:

Linear Probing:

1. Start the program.
2. Include the necessary header files.
3. Write a display function for displaying complete hash table.
4. Computing the hash key using hash function.
5. Hash table[i] = -1 indicates empty slot.
6. Table [key] = num, if empty slot gets then insert the element.
7. Finally display the elements inserted in the hash table.

Quadratic Probing:

1. Start the program.
2. Include the necessary header files.
3. Write a display function for displaying complete hash table.
4. Computing the hash table using mod function.
5. Table[index]==-1 indicates empty slot is found.
6. Table[index]=num, then insert element at that index.
7. Finally display the elements inserted in the hash table.

APEC

Program:

Linear Probing:

```
#include <stdio.h>
#include <conio.h>
void display(int a[], int n)
{
	for (int i = 0; i < n; i++)
	{
		printf("\n % d % d", i, a[i]);
	}
}
void linear_prob(int table[], int tsize, int num)
{
	int key = num % tsize;
	while (table[key] != -1)
	{
		key = (key + 1) % tsize;
	}
	table[key] = num;
	display(table, tsize);
}
int main()
{
	int SIZE = 10;
	int num;
	int hash_table[SIZE];
	char ans;
	for (int i = 0; i < SIZE; i++)
	{
		hash_table[i] = -1;
	}
```

APEC

```
        do {
                printf("\n Enter the Number: ");
                scanf(" % d", &num);
                Linear_prob(hash_tabe, SIZE, num);
                printf("\n DO you Wish to Continue ? (y / n)");
                ans = getch();
        }
        while (ans == 'y');
        return 0;
}
```

Quadratic Probing:

```
#include <stdio.h>
#include <conio.h>
void display(int a[], int n)
{
        for (int i = 0; i < n; i++)
        {
                printf("\n % d % d", i, a[i]);
        }
}
void Quadratic_prob(int table[], int tsize, int num)
{
        int key = num % tsize;
        if (table[key] == -1)
                table[key] = num;
        else
        {
                for (int i = 0; i < tsize; i++)
                {
                        int index = (key + i *i) % tsize;
                        if (table[index] == -1)
                        {
```

APEC

```
                        table[index] = num;
                        break;
                    }
                }
            }
            display(table, tsize);
}
int main()
{
        int SIZE = 10;
        int num;
        int hash_table[SIZE];
        char ans;
        for (int i = 0; i < SIZE; i++)
        {
                hash_table[i] = -1;
        }
        do {
                printf("\n Enter the Number: ");
                scanf(" % d", &num);
                Quadratic_prob(hash_tabe, SIZE, num);
                printf("\n DO you Wish to Continue ? (y / n)");
                ans = getch();
        }
        while (ans == 'y');
        return 0;
}
```

APEC

Output:

Linear Probing:

Enter the Number: 131

0	-1
1	131
2	-1
3	-1
4	-1
5	-1
6	-1
7	-1
8	-1
9	-1

Do you wish to continue? (y/n)

Enter the Number: 21

0	-1
1	131
2	21
3	-1
4	-1
5	-1
6	-1
7	-1
8	-1
9	-1

APEC

Quadratic Probing:

Enter the Number: 37

0	-1
1	-1
2	-1
3	-1
4	-1
5	-1
6	-1
7	37
8	-1
9	-1

Do you wish to continue? (y/n)

Enter the Number: 90

0	90
1	-1
2	-1
3	-1
4	-1
5	-1
6	-1
7	37
8	-1
9	-1

APEC

Result:

Thus the C program for implementation of open addressing with linear and quadratic probing was executed and verified successfully.