

Name: Abdurrahman Qureshi

Roll No: 242466

Practical No: 1

1) To Implement One Dimensional and Two Dimension Array programs using C.

Theory: 1. WAP in C to store and print an array.

WAP in C to print array in a reverse order.

WAP in C to find Min and Max elements of an array.

WAP in C to find sum of two matrices

Algorithm (1) to store and print an array:

```
BEGIN
INT i,n
READ No Of Elements=N INIT ARRAY[N]
FOR (i=0;i<n;i++)
READ Element And Store In ARRAY[i]
FOR (i=0;i<n;i++)
PRINT(ARRAY[i])
END
```

CODE:

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int n,i;
    printf("Enter the number of elements:");
    scanf("%d",&n);

    int a[n];
    printf("\nEnter the elements of the
array:\n");
    for(i=0; i<n; i++)
    {
        printf("Enter element %d:",i+1);
```

```

scanf("%d",&a[i]);
}
printf("Entered elements array:\n");
for(i=0; i<n; i++)
{
    printf("%d ",a[i]);
}
return 0;
}

```

Output:

```

Abdurrahman Qureshi@GhouledGadget MINGW64 /d/Degree/SEM 3/DSA/Programs (master)
● $ ./exp1.exe
Enter the number of elements:3

Enter the elements of the array:
Enter element 1:5
Enter element 2:7
Enter element 3:6
Entered elements array:
5
Abdurrahman Qureshi@GhouledGadget MINGW64 /d/Degree/SEM 3/DSA/Programs (master)
○ $ 

```

Algorithm (2) to print array in a reverse order:

```

BEGIN
INT i,n
READ No Of Elements=N
INIT ARRAY[N]
FOR (i=0;i<n;i++)
READ element and store in ARRAY[i]
FOR(i=n-1;i>=0;i--)
PRINT(ARRAY[i])
END

```

CODE:

```

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

int main() {
    int n,i;
    int a[10];
    printf("Enter the size of array:");
    scanf("%d",&n);

    printf("Enter elements of the array:");
    for(i=0; i<n; i++)
    {
        scanf("%d",&a[i]);
    }
    printf("Reverse of the array:\n");
    for(i=n-1; i>=0; i--)
    {
        printf("%d ",a[i]);
    }
}

```

```

    }
    return 0;
}

```

Output:

```

Abdurrahman Qureshi@GhouledGadget MINGW64 /d/Degree/SEM 3/DSA/Programs (master)
● $ ./exp1.exe
Enter the size of array:4
Enter elements of the array:5
6
2
1
Reverse of the array:
1 2 6 5
Abdurrahman Qureshi@GhouledGadget MINGW64 /d/Degree/SEM 3/DSA/Programs (master)
○ $ █

```

Algorithm (3) to find Min and Max elements of an array:

```

BEGIN
INT i,n,min,max
READ NO OF ELEMENTS=N
INIT ARRAY1[N]
min=array1[0];
max=array1[0];
FOR (i=0;i<n;i++)
READ ELEMENT AND STORE IN ARRAY1[i]
FOR(i=0;i<n;i++)
if (array1[i]<min)
min=array1[i]
if (array1[i]>max)
max=array1[i]
PRINT(MAX)
PRINT(MIN)
END

```

CODE:

```

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

int main() {
    int a[20], n,i,j,c;
    printf("Enter the number of elements:");
}

```

```

scanf("%d",&n);
printf("Enter the numbers:");
for(i=0;i<n;i++){
    scanf("%d",&a[i]);
}

for(i=0;i<n;i++){
    for(j=i+1;j<n;j++){
        if(a[i]>a[j])
            c=a[i];
        a[i]=a[j];
        a[j]=c;
    }
}

printf("Maximum number is %d and
minimum is %d",a[n-1],a[0]);
return 0;

```

Output:

```

Abdurrahman Qureshi@GhouledGadget MINGW64 /d/Degree/SEM 3/DSA/Programs (master)
$ ./exp1.exe
Enter the number of elements:3
Enter the numbers:5
4
1
Maximum number is 5 and minimum is 1
Abdurrahman Qureshi@GhouledGadget MINGW64 /d/Degree/SEM 3/DSA/Programs (master)
$

```

Algorithm (4) to find sum of two matrices :

```

BEGIN
READ matrix 1 and matrix 2 rows and column
Init a[r][c],b[r][c],d[r][c]
for i=1 to rows[a]
for j=1 to columns [a]
READ matrix 1 values
for i=1 to rows[b]
for j=1 to columns [b]
READ matrix 2 values
for i=1 to rows[d]
for j=1 to columns [d]
d[i,j]= a[i,j]+ b[i,j];
PRINT d[i,j];
END

```

CODE:

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

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

```

int main()
{
    int r,c,i,j;

    printf("Enter numbers of rows and
columns:");
    scanf("%d%d",&r,&c);
    int a[r][c];
    int b[r][c];
    int d[r][c];
    printf("\nEnter elements of matrix 1\n");
    for(i=0; i<r; i++)
    {
        for(j=0; j<c; j++)
        {
            printf("Enter
a%d%d:",i+1,j+1);
            scanf("%d",&a[i][j]);
        }
    }
    printf("\nEnter elements of matrix 2\n");
    for(i=0; i<r; i++)
    {
        for(j=0; j<c; j++)
        {
            printf("Enter
b%d%d:",i+1,j+1);
            scanf("%d",&b[i][j]);
        }
    }

    printf("\nMatrix 1\n");
    for(i=0; i<r; i++)
    {
        for(j=0; j<c; j++)
        {
            printf("%d\t",a[i][j]);
        }
        printf("\n");
    }
    printf("\nMatrix 2\n");
    for(i=0; i<r; i++)
    {
        for(j=0; j<c; j++)
        {
            printf("%d\t",b[i][j]);
        }
        printf("\n");
    }
    printf("\nSum of the matrices:\n");
    for(i=0; i<r; i++)
    {
        for(j=0; j<c; j++)
        {
            d[i][j]=a[i][j]+b[i][j];
            printf("%d\t",d[i][j]);
        }
        printf("\n");
    }
    return 0;
}

```

Output:

```
Abdurrahman Qureshi@GhouledGadget MINGW64 /d/Degree/SEM 3/DSA/Programs (master)
$ ./a.exe
Enter numbers of rows and columns:2
2

Enter elements of matrix 1
Enter a11:6
Enter a12:1
Enter a21:4
Enter a22:2

Enter elements of matrix 2
Enter b11:3
Enter b12:8
Enter b21:5
Enter b22:4

Matrix 1
6      1
4      2

Matrix 2
3      8
5      4

Sum of the matrices:
9      9
9      6
```

Tools used :

Software: Dev c++

Hardware: Lab Computers

References: No references used.

Conclusion:

I have understood and used the basic concepts and principles of various linked lists, stacks and queues. Stack is a container of objects that are inserted and removed according to the last-in first-out (LIFO) principle.

A one-dimensional array stores a single list of various elements having a similar data type.

A two-dimensional array stores an array of various arrays, a list of various lists, or an array of various one-dimensional arrays.

We can initialize each type of array at the time of declaration and could access any of their elements after that.

Name: Abdurrahman Qureshi

Roll No: 242466

Practical No: 2

1) To implement stack and Queue using array.

CODE (STACK)

```
#include<stdio.h>
#include<stdlib.h>
#define MAX 4
int i, Top=-1, inp_array[MAX];
void Push();
void Pop();
void show();
void peek();
int main()
{
    int choice;
    while(1)
    {
        printf("\nOperations performed by Stack");
        printf("\n1.Push the element\n2.Pop the element\n3.Show\n4. Peek \n5.End");
        printf("\n\nEnter the choice:");
        scanf("%d",&choice);
        switch(choice)
        {
            case 1: Push();
                    break;
            case 2: Pop();
                    break;
            case 3: show();
                    break;
            case 4: peek();
```

```

                break;
                case 5: exit(0);
                default: printf("\nInvalid choice!!");
            }
        }
    }
}

void Push()
{
    int x;
    if(Top==MAX-1)
    {
        printf("\nOverflow!!");
    }

    else
    {
        printf("\nEnter element to be inserted to the stack:");
        scanf("%d",&x);
        Top=Top+1;
        inp_array[Top]=x;
    }
}

void Pop()
{
    if(Top== -1)
    {
        printf("\nUnderflow!!");
    }
    else
    {
        printf("\nPopped element: %d",inp_array[Top]);
        Top=Top-1;
    }
}

void show()
{
    if(Top== -1)
    {

```



```

        printf("\nUnderflow!!");
    }
    else
    {
        printf("\nElements present in the stack: \n");
        for(i=Top;i>=0;i--)
            printf("%d\n",inp_array[i]);
    }
}

void peek()
{
    if(Top== -1)
    {
        printf("Stack is empty");
    }
    else {
        printf("%d",inp_array[Top]);
    }
}
}

```

OUTPUT (STACK)

```

MINGW64/d/Degree/SEM 3/i x + v
Abdurrahman Qureshi@GhouledGadget MINGW64 /d/Degree/SEM 3/DSA/Programs (master)
$ ./a.exe

Operations performed by Stack
1.Push the element
2.Pop the element
3.Show
4. Peek
5.End

Enter the choice:1

Enter element to be inserted to the stack:45

Operations performed by Stack
1.Push the element
2.Pop the element
3.Show
4. Peek
5.End

Enter the choice:1

Enter element to be inserted to the stack:74

Operations performed by Stack
1.Push the element
2.Pop the element
3.Show
4. Peek
5.End

Enter the choice:1

Enter element to be inserted to the stack:23

Operations performed by Stack
1.Push the element
2.Pop the element
3.Show
4. Peek

```

```
MINGW64:/d/Degree/SEM 3/IT  x  +  v
5.End
Enter the choice:3
Elements present in the stack:
23
74
45
Operations performed by Stack
1.Push the element
2.Pop the element
3.Show
4. Peek
5.End
Enter the choice:2
Popped element: 23
Operations performed by Stack
1.Push the element
2.Pop the element
3.Show
4. Peek
5.End
Enter the choice:3
Elements present in the stack:
74
45
Operations performed by Stack
1.Push the element
2.Pop the element
3.Show
4. Peek
5.End
Enter the choice:5
Abdurrahman Qureshi@GhouledGadget MINGW64 /d/Degree/SEM 3/DSA/Programs (master)
```

Source code (QUEUE)

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

int q[MAX];
int f=-1,r=-1,i,x;
void insert();
void delet();
void peek();
void display();
int main()
{
    int c;

    while(1)
    {
        printf("\n Press 1 for insertion");
        printf("\n Press 2 for deletion");
        printf("\n Press 3 for peek");
        printf("\n Press 4 to display\n");
        scanf("%d",&c);

        switch(c)
```

```

        {
            case 1: insert();
            break;
            case 2: delet();
            break;
            case 3: peek();
            break;
            case 4: display();
            break;
            default: printf("\n invalid choice");

        }
    }
}

void insert()
{
    printf("\n Enter value to insert");
    scanf("%d",&x);
    if (r==MAX-1)
    {
        printf("OVERFLOW");
    }
    else
    {
        if(f== -1)
            f=0;
        r=r+1;
        q[r]=x;
        printf("Inserted element is %d\n",q[r]);
    }
}

void delet()
{
    if (f== -1 || f>r)
    {
        printf("UNDERFLOW");
    }
    else

```

```

        {
            x = q[f];
            printf("deleted value is %d\n",q[f]);
            f=f+1;
        }
    }
void peek()
{
    if (f== -1 || f>r)
    {
        printf("UNDERFLOW");
    }
    else
    {
        printf("rear value is : %d",q[r]);
        printf("front value is : %d",q[f]);
    }
}
void display()
{
    if (f== -1 || f>r)
    {
        printf("UNDERFLOW");
    }
    else {
        printf("Queue elements are \n");
        for(i=f;i<=r;i++)
            printf("%d\t",q[i]);
    }
}
}

```

OUTPUT (QUEUE)

```
Abdurrahman Qureshi@GhouledGadget MINGW64 /d/Degree/SEM 3/DSA/Programs (master)
$ ./exp1.exe

Press 1 for insertion
Press 2 for deletion
Press 3 for peek
Press 4 to display
1
Enter value to insert56
Inserted element is 56

Press 1 for insertion
Press 2 for deletion
Press 3 for peek
Press 4 to display
1
Enter value to insert74
Inserted element is 74

Press 1 for insertion
Press 2 for deletion
Press 3 for peek
Press 4 to display
1
Enter value to insert85
Inserted element is 85

Press 1 for insertion
Press 2 for deletion
Press 3 for peek
Press 4 to display
4
Queue elements are
56 74 85
Press 1 for insertion
Press 2 for deletion
Press 3 for peek
Press 4 to display
```

```
1
Enter value to insert4
Inserted element is 4

Press 1 for insertion
Press 2 for deletion
Press 3 for peek
Press 4 to display
2
deleted value is 56

Press 1 for insertion
Press 2 for deletion
Press 3 for peek
Press 4 to display
4
Queue elements are
74 85 4
Press 1 for insertion
Press 2 for deletion
Press 3 for peek
Press 4 to display
2
deleted value is 74

Press 1 for insertion
Press 2 for deletion
Press 3 for peek
Press 4 to display
4
Queue elements are
85 4
Press 1 for insertion
Press 2 for deletion
Press 3 for peek
Press 4 to display
Queue elements are
85

Abdurrahman Qureshi@GhouledGadget MINGW64 /d/Degree/SEM 3/DSA/Programs (master)
$ |
```

Tools used :

Software: Dev c++

Hardware: Lab Computers

References: Mam notes.

Conclusion:

(A) STACK

Stack implements the LIFO mechanism i.e. the element that is pushed at the end is popped out first.

Push: This operation adds an item to the stack; it will throw an exception if the stack's capacity is full.

Pop: This removes the last inserted element from a stack that is removed in reverse order of their insertion; it will throw an underflow exception if the given stack is empty.

Peek: This operation returns the topmost element of the stack.

(B) QUEUE

It is a type of linear data structure which follow first in first out(FIFO) approach. This means the element that is inserted first will be removed first. The queue

is said to have 2 ends, one from where the element is inserted is known as the REAR end, and the end where the element is deleted is known as FRONT. This means in the queue, one end (REAR) is always used to enqueue, and the other end (FRONT) is used to dequeue.

Name: Abdurrahman Qureshi

Roll No: 242466

Practical No: 3

1) Insertion, deletion and reversing operations with singly linked

CODE

```
#include <stdio.h>

#include <malloc.h>

#include <stdlib.h>
struct node
{
    int data;
    struct node *next;
};
struct node *start = NULL;
void createll();
void Insertbegin();
void Insertend();
void Insertbefore();
void Insertafter();
void DeleteAfter();
void DeleteBegin();
void DeleteEnd();
void display();
int main()
{
    int n;
    while (1)
    {
        printf("\n\nEnter 1 for create linked list\n");
        printf("Enter 2 for insert in the beginning\n ");
        printf("Enter 3 for insert in the end\n ");
        printf("Enter 4 for insert before a given node \n");
        printf("Enter 5 for insert after a given node\n ");
        printf("Enter 6 for deleting at the beginning\n ");
        printf("Enter 7 for delete at the end\n");
        printf("Enter 8 for delete after a given node\n ");
        printf("Enter 9 to display \n");
        printf("ENTER YOUR CHOICE");
```

```

scanf("%d", &n);

switch (n)
{
case 1:
    createll();
    break;
case 2:
    Insertbegin();
    break;
case 3:
    Insertend();
    break;
case 4:
    Insertbefore();
    break;
case 5:
    Insertafter();
    break;
case 6:
    DeleteBegin();
    break;
case 7:
    DeleteEnd();
    break;
case 8:
    DeleteAfter();
    break;
case 9:
    display();
    break;
default:
    {
        printf("Error");
    }
}
}

void createll()
{
    struct node *avail, *ptr;
    avail = (struct node *)malloc(sizeof(struct node));

    if (avail == NULL)
    {
        printf("\n out of space");
    }
    printf("Enter the data:-\n");
    scanf("%d", &avail->data);
    avail->next = NULL;
    if (start == NULL)
    {
        start = avail;
    }
}

```



```

    }
    else
    {
        while (ptr->next != NULL)
        {
            ptr = ptr->next;
        }

        ptr->next = avail;
    }
}

void display()
{
    struct node *ptr;
    if (start == NULL)
    {
        printf("\n Linked list is empty");
    }
    else
    {
        ptr = start;
        printf("Linked list elemens are:-");
        while (ptr != NULL)

        {

            printf("%d\t", ptr->data);

            printf("%d\t", ptr->next);
            ptr = ptr->next;
        }
    }
}

void Insertbegin()
{
    struct node *avail;
    avail = (struct node *)malloc(sizeof(struct node));
    if (avail == NULL)
    {
        printf("\n out of space");
    }
    printf("Enter the data:-\n");
    scanf("%d", &avail->data);
    avail->next = NULL;
    if (start == NULL)
    {
        start = avail;
    }
    else
    {
        avail->next = start;
        start = avail;
    }
}

```

```

}

void Insertend()

{
    struct node *avail, *ptr;

    avail = (struct node *)malloc(sizeof(struct node));
    if (avail == NULL)

    {
        printf("\n out of space");
    }
    printf("Enter the data:-\n");
    scanf("%d", &avail->data);
    avail->next = NULL;
    ptr = start;
    while (ptr->next != NULL)
    {
        ptr = ptr->next;
    }
    ptr->next = avail;
}

void Insertbefore()
{
    struct node *avail, *ptr, *preptr;
    int val;
    avail = (struct node *)malloc(sizeof(struct node));
    if (avail == NULL)
    {
        printf("\n out of space");
    }
    printf("Enter the data values for the node:-\n");
    scanf("%d", &avail->data);

    printf("Enter the node value to add before it:-\n");
    scanf("%d", &val);

    avail->next = NULL;

    ptr = start;
    while (ptr->data != val)
    {
        preptr = ptr;
        ptr = ptr->next;
    }
    preptr->next = avail;
    avail->next = ptr;
}

void Insertafter()
{
    struct node *avail, *ptr, *preptr;

    int val;

```

```

avail = (struct node *)malloc(sizeof(struct node));
if (avail == NULL)
{
    printf("\n out of space");
}
printf("Enter the data values for the node:-\n");
scanf("%d", &avail->data);
printf("Enter the node value to add after it:-\n");
scanf("%d", &val);

avail->next = NULL;

ptr = start;
while (preptr->data != val)
{
    preptr = ptr;
    ptr = ptr->next;
}
preptr->next = avail;
avail->next = ptr;
}
void DeleteBegin()
{
    struct node *ptr;
    if (start == NULL)
    {
        printf("\n Linked list is empty");
    }
    ptr = start;
    start = start->next;
    free(ptr);
}
void DeleteEnd()
{
    struct node *ptr, *preptr;
    if (start == NULL)
    {
        printf("Linked list is empty");
    }
    ptr = start;
    while (ptr->next != NULL)
    {
        preptr = ptr;
        ptr = ptr->next;
    }
    preptr->next = NULL;

    free(ptr);
}
void DeleteAfter()
{
    struct node *ptr, *preptr;
    int val;

```

OUTPUT

```
MINGWWSA/d/egree/SEM3 \x + v
ENTER YOUR CHOICE3
Enter the data:-
65

Enter 1 for create linked list
Enter 2 for insert in the beginning
Enter 3 for insert in the end
Enter 4 for insert before a given node
Enter 5 for insert after a given node
Enter 6 for deleting at the beginning
Enter 7 for delete at the end
Enter 8 for delete after a given node
Enter 9 to display
ENTER YOUR CHOICE9
Linked list elements are:-74 10822384 54 10822448 65 0

Enter 1 for create linked list
Enter 2 for insert in the beginning
Enter 3 for insert in the end
Enter 4 for insert before a given node
Enter 5 for insert after a given node
Enter 6 for deleting at the beginning
Enter 7 for delete at the end
Enter 8 for delete after a given node
Enter 9 to display
ENTER YOUR CHOICE6

Enter 1 for create linked list
Enter 2 for insert in the beginning
Enter 3 for insert in the end
Enter 4 for insert before a given node
Enter 5 for insert after a given node
Enter 6 for deleting at the beginning
Enter 7 for delete at the end
Enter 8 for delete after a given node
Enter 9 to display
ENTER YOUR CHOICE9
Linked list elements are:-54 10822448 65 0

Enter 1 for create linked list
```

```
MINGW64:/d/Degree/SEM 3/I  x  +  v
Enter 5 for insert after a given node
Enter 6 for deleting at the beginning
Enter 7 for delete at the end
Enter 8 for delete after a given node
Enter 9 to display
ENTER YOUR CHOICE6

Enter 1 for create linked list
Enter 2 for insert in the beginning
Enter 3 for insert in the end
Enter 4 for insert before a given node
Enter 5 for insert after a given node
Enter 6 for deleting at the beginning
Enter 7 for delete at the end
Enter 8 for delete after a given node
Enter 9 to display
ENTER YOUR CHOICE9
Linked list elemens are:-54    10822448    65    0

Enter 1 for create linked list
Enter 2 for insert in the beginning
Enter 3 for insert in the end
Enter 4 for insert before a given node
Enter 5 for insert after a given node
Enter 6 for deleting at the beginning
Enter 7 for delete at the end
Enter 8 for delete after a given node
Enter 9 to display
ENTER YOUR CHOICELinked list elemens are:-

Abdurrahman Qureshi@GhouledGadget MINGW64 /d/Degree/SEM 3/DSA/Programs (master)
$ |
```

Tools used :

Software: Dev c++

Hardware: Lab Computers

References: Mam notes.

Conclusion

- insertBeg(): This function simply inserts an element at the front/beginning of the linked list.
- insertEnd(): This function inserts an element at the end of the linked list.
- deleteBeg(): This function simply deletes an element from the front/beginning of the linked list.
- deleteEnd(): This function simply deletes an element from the end of the linked list.

Name: Abdurrahman Qureshi

Roll No: 242466

Practical No: 4

1) Program to demonstrate Linear Search

CODE

```
#include <stdio.h>
#include <stdlib.h>
#define size 20
int main()
{
    int a[size], i, n, x, pos = -1;
    printf("Enter size of array:- \n");
    scanf("%d", &n);
    printf("Enter elements \n");
    for (i = 0; i < n; i++)
    {
        scanf("%d", &a[i]);
    }

    printf("Elements are:- \n");
    for (i = 0; i < n; i++)
    {
        printf("%d\t", a[i]);
    }
    printf("\nEnter elements to search:- \n");
    scanf("%d", &x);
    for (i = 0; i < n; i++)
    {
        if (a[i] == x)
        {
            pos = i + 1;
            printf("%d is found at position %d", x, pos);
        }
    }
    {
        if (pos == -1)
        {
            printf("Elements not found \n");
        }
    }
}
```

```
}
```

OUTPUT

```
Abdurrahman Qureshi@GhouledGadget MINGW64 /d/Degree/SEM 3/DSA/Programs (master)
$ gcc -o expl.exe expl.c

Abdurrahman Qureshi@GhouledGadget MINGW64 /d/Degree/SEM 3/DSA/Programs (master)
$ ./expl.exe
Enter size of array:-
5
Enter elements
1
7
5
9
3
Elements are:-
1      7      5      9      3
Enter elements to search:-
5
5 is found at position 3
Abdurrahman Qureshi@GhouledGadget MINGW64 /d/Degree/SEM 3/DSA/Programs (master)
$ |
```

2) Program to demonstrate Binary Search

CODE

```
#include <stdio.h>
#include <stdlib.h>
#define size 20
int main()
{
    int a[size], i, n, x, pos = -1;
    printf("Enter size of array:- \n");
    scanf("%d", &n);
    printf("Enter elements \n");
    for (i = 0; i < n; i++)
    {
        scanf("%d", &a[i]);
    }

    printf("Elements are:- \n");
    for (i = 0; i < n; i++)
    {
        printf("%d\t", a[i]);
    }
    printf("\nEnter elements to search:- \n");
    scanf("%d", &x);
    for (i = 0; i < n; i++)
    {
        if (a[i] == x)
        {
            pos = i + 1;
            printf("%d is found at position %d", x, pos);
        }
    }
}
```

```

    {
        if (pos == -1)
        {
            printf("Elements not found \n");
        }
    }
}

```

OUTPUT

```

MINGW64/d/Degree/SEM 3/1 x + v
5
5 is found at position 3
Abdurrahman Qureshi@GhouledGadget MINGW64 /d/Degree/SEM 3/DSA/Programs (master)
$ gcc -o expl.exe expl.c
Abdurrahman Qureshi@GhouledGadget MINGW64 /d/Degree/SEM 3/DSA/Programs (master)
$ ./expl.exe
Enter size of an array
4
Enter elements
1
7
6
8
The no. in asending order are
1
6
7
8
Enter elements to search:-
5
Element not found
Element not found
Abdurrahman Qureshi@GhouledGadget MINGW64 /d/Degree/SEM 3/DSA/Programs (master)
$ ./expl.exe
Enter size of an array
6
Enter elements
7
1
5
8
3
2
The no. in asending order are
1
2
3
5
7
8
Enter elements to search:-
8
Element not found
Element not found
8 is found at position 6
Abdurrahman Qureshi@GhouledGadget MINGW64 /d/Degree/SEM 3/DSA/Programs (master)
$ |

```

Tools used :

Software: Dev c++

Hardware: Lab Computers

References: Mam notes.

Conclusion

- Binary search helps to sort the elements.
- Binary Search is consistently faster than Linear Search.
- Linear search is less efficient when we consider the large data sets.
- Binary search is more efficient than the linear search in the case of large data sets.

Name: Abdurrahman Qureshi

Roll No: 242466

Practical No: 5

1) Program to demonstrate Sorting algorithms

SELECTION SORT CODE

```
#include <stdio.h>
#include <stdlib.h>
#define size 20
int small(int a[], int k, int n);
void selection(int a[], int n);
int main()
{
    int a[size], n, i;
    printf("size:- ");
    scanf("%d", &n);
    printf("\nelements are :- \n");
    for (i = 0; i < n; i++)
    {
        scanf("%d", &a[i]);
    }
    selection(a, n);
    printf("Sorted elements are:-\n");
    for (i = 0; i < n; i++)
    {
        printf("%d\t", a[i]);
    }
}

int small(int a[], int k, int n)
{
    int pos = k, s = a[k], i;
    for (i = k + 1; i < n; i++)
    {
        if (a[i] < s)
        {
            s = a[i];
            pos = i;
        }
    }
    return pos;
}

void selection(int a[], int n)
{

```

```

int k, pos, temp;
for (k = 0; k < n; k++)
{
    pos = small(a, k, n);
    temp = a[k];
    a[k] = a[pos];
    a[pos] = temp;
}
}

```

OUTPUT

```

Abdurrahman Qureshi@GhouledGadget MINGW64 /d/Degree/SEM 3/DSA/Programs (master)
$ gcc -o expl.exe expl.c

Abdurrahman Qureshi@GhouledGadget MINGW64 /d/Degree/SEM 3/DSA/Programs (master)
$ ./expl.exe
size:- 5

elements are :-
1
8
6
5
7
Sorted elements are:-
1      5      6      7      8
Abdurrahman Qureshi@GhouledGadget MINGW64 /d/Degree/SEM 3/DSA/Programs (master)
$ |

```

BUBBLE SORT CODE

```

#include <stdio.h>
#include <malloc.h>
#include <stdlib.h>
int main()
{
    int a[20];
    int i, j, temp, n;
    printf("Enter size of an array\n");
    scanf("%d", &n);
    printf("Enter elements\n");
    for (i = 0; i < n; i++)
    {
        scanf("%d", &a[i]);
    }
    for (i = 0; i < n; i++)
    {
        for (j = 0; j < n - 1; j++)
        {
            if (a[j] > a[j + 1])
            {
                temp = a[j];
                a[j] = a[j + 1];
                a[j + 1] = temp;
            }
        }
    }
}

```

```

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

```

OUTPUT

```

Abdurrahman Qureshi@GhouledGadget MINGW64 /d/Degree/SEM 3/DSA/Programs (master)
$ gcc -o exp1.exe exp1.c

Abdurrahman Qureshi@GhouledGadget MINGW64 /d/Degree/SEM 3/DSA/Programs (master)
$ ./exp1.exe
Enter size of an array
4
Enter elements
7
5
3
1
sorted elements are
1    3    5    7
Abdurrahman Qureshi@GhouledGadget MINGW64 /d/Degree/SEM 3/DSA/Programs (master)
$ |

```

INSERTION SORT CODE

```

#include <stdio.h>
#include <stdlib.h>
#define size 10
void insertion_sort(int a[], int n);
int main()
{
    int a[size], i, n;
    printf("\n Enter size of array:\n");
    scanf("%d", &n);
    printf("\n Enter elements:\n");
    for (i = 0; i < n; i++)
    {
        scanf("%d", &a[i]);
    }
    insertion_sort(a, n);
    printf("\n Sorted elements are:\n");
    for (i = 0; i < n; i++)
    {
        printf("%d\t", a[i]);
    }
}

void insertion_sort(int a[], int n)
{
    int i, j, temp;
    for (i = 1; i < n; i++)
    {
        temp = a[i];
        j = i - 1;
        while ((temp < a[j]) && (j >= 0))
        {

```

```

        a[j + 1] = a[j];
        j--;
    }
    a[j + 1] = temp;
}
}

```

OUTPUT

```

Abdurrahman Qureshi@GhouledGadget MINGW64 /d/Degree/SEM 3/DSA/Programs (master)
$ gcc -o expl.exe expl.c

Abdurrahman Qureshi@GhouledGadget MINGW64 /d/Degree/SEM 3/DSA/Programs (master)
$ ./expl.exe

Enter size of array:
5

Enter elements:
4
7
1
6
2

Sorted elements are:
1      2      4      6      7
Abdurrahman Qureshi@GhouledGadget MINGW64 /d/Degree/SEM 3/DSA/Programs (master)
$ |

```

HEAP SORT CODE

```

#include <stdio.h>
void swap(int *a, int *b)
{
    int temp = *a;
    *a = *b;
    *b = temp;
}
void heapify(int arr[], int N, int i)
{
    int largest = i;
    int left = 2 * i + 1;
    int right = 2 * i + 2;
    if (left < N && arr[left] > arr[largest])
        largest = left;
    if (right < N && arr[right] > arr[largest])
        largest = right;
    if (largest != i)
    {
        swap(&arr[i], &arr[largest]);
        heapify(arr, N, largest);
    }
}
void heapSort(int arr[], int N)
{
    for (int i = N / 2 - 1; i >= 0; i--)
        heapify(arr, N, i);
    for (int i = N - 1; i >= 0; i--)

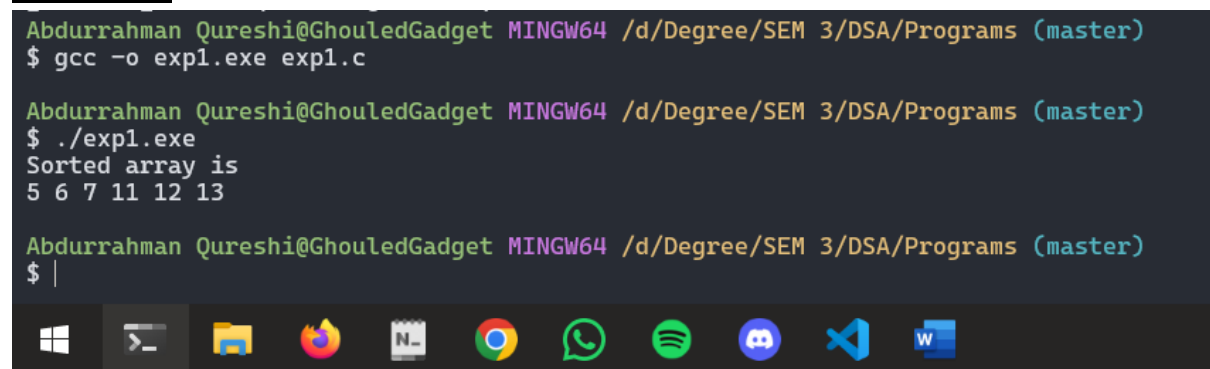
```

```

    {
        swap(&arr[0], &arr[i]);
        heapify(arr, i, 0);
    }
}
void printArray(int arr[], int N)
{
    for (int i = 0; i < N; i++)
        printf("%d ", arr[i]);
    printf("\n");
}
int main()
{
    int arr[] = {12, 11, 13, 5, 6, 7};
    int N = sizeof(arr) / sizeof(arr[0]);
    heapSort(arr, N);
    printf("Sorted array is\n");
    printArray(arr, N);
}

```

OUTPUT



```

Abdurrahman Qureshi@GhouledGadget MINGW64 /d/Degree/SEM 3/DSA/Programs (master)
$ gcc -o expl.exe expl.c

Abdurrahman Qureshi@GhouledGadget MINGW64 /d/Degree/SEM 3/DSA/Programs (master)
$ ./expl.exe
Sorted array is
5 6 7 11 12 13

Abdurrahman Qureshi@GhouledGadget MINGW64 /d/Degree/SEM 3/DSA/Programs (master)
$ |

```

Tools used :

Software: Dev c++

Hardware: Lab Computers

References: Mam notes.

Conclusion

Heap sort, selection sort, insertion sort, and bubble sort are fundamental sorting algorithms.

Heap sort leverages a binary heap for efficient sorting, while selection sort repeatedly finds the smallest element. Insertion sort builds the sorted array incrementally, and bubble sort repeatedly swaps adjacent elements. Each algorithm varies in efficiency and use cases.

Name: Abdurrahman Qureshi

Roll No: 242466

Practical No: 6

1) Demonstrate Insertion and Deletion operation on Binary Search Tree

CODE

```
#include <stdio.h>
#include <stdlib.h>
#include <malloc.h>
struct node
{
    int data;
    struct node *left;
    struct node *right;
};
struct node *tree;
void create_tree();
int insertElement();
void preorderTraversal();
void inorderTraversal();
void postorderTraversal();
int deleteElement();
int main()
{
    int c, val;
    struct node *ptr;
    create_tree(tree);
    do
    {
        printf("\n *****MAIN MENU***** \n");
        printf("\n 1. Insert Element");
        printf("\n 2. Preorder Traversal");
        printf("\n 3. Inorder Traversal");
        printf("\n 4. Postorder Traversal");
        printf("\n 5. Delete an element");
        printf("\n\n Enter your option : ");
        scanf("%d", &c);
        switch (c)
        {
            case 1:
```

```

        insertElement();
        break;
    case 2:
        preorderTraversal();
        break;
    case 3:
        inorderTraversal();
        break;
    case 4:
        postorderTraversal();
        break;
    case 5:
        deleteElement();
        break;
    }
} while (c != 6);
return 0;
}
void create_tree()
{
    tree = NULL;
}
int insertElement()
{
    struct node *ptr, *nodeptr, *parentptr;
    int val;
    ptr = (struct node *)malloc(sizeof(struct node));
    printf("\n Enter the value of the new node : ");
    scanf("%d", &val);
    ptr->data = val;
    ptr->left = NULL;
    ptr->right = NULL;
    if (tree == NULL)
    {
        tree = ptr;
        tree->left = NULL;
        tree->right = NULL;
    }
    else
    {
        parentptr = NULL;
        nodeptr = tree;
        while (nodeptr != NULL)
        {
            parentptr = nodeptr;
            if (val < nodeptr->data)
                nodeptr = nodeptr->left;
            else
                nodeptr = nodeptr->right;
        }
        if (val < parentptr->data)
            parentptr->left = ptr;
        else
            parentptr->right = ptr;
    }
    return tree;
}
void preorderTraversal()

```

```

{
    if (tree != NULL)
    {
        printf("%d\t", tree->data);
        preorderTraversal(tree->left);
        preorderTraversal(tree->right);
    }
}

void inorderTraversal()
{
    if (tree != NULL)
    {
        inorderTraversal(tree->left);
        printf("%d\t", tree->data);
        inorderTraversal(tree->right);
    }
}

void postorderTraversal()
{
    if (tree != NULL)
    {
        postorderTraversal(tree->left);
        postorderTraversal(tree->right);
        printf("%d\t", tree->data);
    }
}

int deleteElement()
{
    struct node *cur, *parent, *suc, *psuc, *ptr;
    int val;
    if (tree->left == NULL)
    {
        printf("\n The tree is empty ");
        return (tree);
    }
    printf("\n Enter the element to be deleted : ");
    scanf("%d", &val);
    parent = tree;
    cur = tree->left;
    while (cur != NULL && val != cur->data)
    {
        parent = cur;
        cur = (val < cur->data) ? cur->left : cur->right;
    }
    if (cur == NULL)
    {
        printf("\n The value to be deleted is not present in the tree");
        return (tree);
    }
    if (cur->left == NULL)
        ptr = cur->right;
    else if (cur->right == NULL)
        ptr = cur->left;
    else
    {
        psuc = cur;
        cur = cur->left;
        while (suc->left != NULL)

```



```

    {
        psuc = suc;
        suc = suc->left;
    }
    if (cur == psuc)
    {
        suc->left = cur->right;
    }
    else
    {
        suc->left = cur->left;
        psuc->left = suc->right;
        suc->right = cur->right;
    }
    ptr = suc;
}
if (parent->left == cur)
    parent->left = ptr;
else
    parent->right = ptr;
free(cur);
return tree;
}

```

OUTPUT

```

MINGW64/d/Degree/SEM 3/1  ×  +  ▾
Abdurrahman Qureshi@GhouledGadget MINGW64 /d/Degree/SEM 3/DSA/Programs (master)
$ ./expl.exe

*****MAIN MENU*****
1. Insert Element
2. Preorder Traversal
3. Inorder Traversal
4. Postorder Traversal
5. Delete an element
Enter your option : 1
Enter the value of the new node : 2

*****MAIN MENU*****
1. Insert Element
2. Preorder Traversal
3. Inorder Traversal
4. Postorder Traversal
5. Delete an element
Enter your option : 1
Enter the value of the new node : 4

*****MAIN MENU*****
1. Insert Element
2. Preorder Traversal
3. Inorder Traversal
4. Postorder Traversal
5. Delete an element
Enter your option : 1
Enter the value of the new node : 9

*****MAIN MENU*****
1. Insert Element
2. Preorder Traversal
3. Inorder Traversal
4. Postorder Traversal
5. Delete an element
Enter your option : 1
Enter the value of the new node : 5

```

```
MINGW64/d/Degree/SEM 3/I x + v
Enter the value of the new node : 5
*****MAIN MENU*****
1. Insert Element
2. Preorder Traversal
3. Inorder Traversal
4. Postorder Traversal
5. Delete an element
Enter your option : 2
2 4 9 5
*****MAIN MENU*****
1. Insert Element
2. Preorder Traversal
3. Inorder Traversal
4. Postorder Traversal
5. Delete an element
Enter your option : 3
2 4 5 9
*****MAIN MENU*****
1. Insert Element
2. Preorder Traversal
3. Inorder Traversal
4. Postorder Traversal
5. Delete an element
Enter your option : 4
5 9 4 2
*****MAIN MENU*****
1. Insert Element
2. Preorder Traversal
3. Inorder Traversal
4. Postorder Traversal
5. Delete an element
Enter your option : 5
Enter the element to be deleted : 2
*****MAIN MENU*****
1. Insert Element
2. Preorder Traversal
3. Inorder Traversal
4. Postorder Traversal
```

```
MINGW64/d/Degree/SEM 3/I x + v
2. Preorder Traversal
3. Inorder Traversal
4. Postorder Traversal
5. Delete an element
Enter your option : 4
5 9 4 2
*****MAIN MENU*****
1. Insert Element
2. Preorder Traversal
3. Inorder Traversal
4. Postorder Traversal
5. Delete an element
Enter your option : 5
Enter the element to be deleted : 2
*****MAIN MENU*****
1. Insert Element
2. Preorder Traversal
3. Inorder Traversal
4. Postorder Traversal
5. Delete an element
Enter your option : 4
5 9 4
*****MAIN MENU*****
1. Insert Element
2. Preorder Traversal
3. Inorder Traversal
4. Postorder Traversal
5. Delete an element
Enter your option : 3
4 5 9
*****MAIN MENU*****
1. Insert Element
2. Preorder Traversal
3. Inorder Traversal
4. Postorder Traversal
5. Delete an element
Enter your option : 4 5 9
Abdurrahman Qureshi@GhouladGadget MINGW64 /d/Degree/SEM 3/DSA/Programs (master)
$ |
```

Tools used :

Software: Dev c++

Hardware: Lab Computers

References: Mam notes.

Conclusion

Binary Search Tree (or BST) is a special kind of binary tree in which the values of all the nodes of the left subtree of any node of the tree are smaller than the value of the node. Also, the values of all the nodes of the right subtree of any node are greater than the value of the node.

Name: Abdurrahman Qureshi

Roll No: 242466

Practical No: 7

1) Demonstrate advance methods of AVL tree.

CODE

```
#include <stdio.h>
typedef struct node
{
    int data;
    struct node *left, *right;
    int ht;
} node;
node *insert(node *, int);
node *Delete(node *, int);
void preorder(node *);
void inorder(node *);
int height(node *);
node *rotateright(node *);
node *rotateleft(node *);
node *RR(node *);
node *LL(node *);
node *LR(node *);
node *RL(node *);
int BF(node *);
int main()
{
    node *root = NULL;
    int x, n, i, op;
    do
    {
        printf("\n1)Create:");
        printf("\n2)Insert:");
        printf("\n3)Delete:");
        printf("\n4)Print:");
        printf("\n5)Quit:");
        printf("\n\nEnter Your Choice:");
        scanf("%d", &op);
        switch (op)
        {
            case 1:
                printf("\nEnter no. of elements:");
```

```

scanf("%d", &n);
printf("\nEnter tree data:");
root = NULL;
for (i = 0; i < n; i++)
{
    scanf("%d", &x);
    root = insert(root, x);
}
break;
case 2:
    printf("\nEnter a data:");
    scanf("%d", &x);
    root = insert(root, x);
    break;
case 3:
    printf("\nEnter a data:");
    scanf("%d", &x);
    root = Delete(root, x);
    break;
case 4:
    printf("\nPreorder sequence:\n");
    preorder(root);
    printf("\n\nInorder sequence:\n");
    inorder(root);
    printf("\n");
    break;
}
} while (op != 5);
return 0;
}
node *insert(node *T, int x)
{
    if (T == NULL)
    {
        T = (node *)malloc(sizeof(node));
        T->data = x;
        T->left = NULL;
        T->right = NULL;
    }
    else
        if (x > T->data) // insert in right subtree
        {
            T->right = insert(T->right, x);
            if (BF(T) == -2)
                if (x > T->right->data)
                    T = RR(T);
                else
                    T = RL(T);
        }
        else if (x < T->data)
        {
            T->left = insert(T->left, x);
            if (BF(T) == 2)
                if (x < T->left->data)
                    T = LL(T);
                else
                    T = LR(T);
        }
}

```

```

    T->ht = height(T);
    return (T);
}
node *Delete(node *T, int x)
{
    node *p;
    if (T == NULL)
    {
        return NULL;
    }
    else if (x > T->data) // insert in right subtree
    {
        T->right = Delete(T->right, x);
        if (BF(T) == 2)
            if (BF(T->left) >= 0)
                T = LL(T);
            else
                T = LR(T);
    }
    else if (x < T->data)
    {
        T->left = Delete(T->left, x);
        if (BF(T) == -2) // Rebalance during windup
            if (BF(T->right) <= 0)
                T = RR(T);
            else
                T = RL(T);
    }
    else
    {
        // data to be deleted is found
        if (T->right != NULL)
        { // delete its inorder successor p=T->right; while(p->left!= NULL) p=p->left; T->data=p->data;
            T->right = Delete(T->right, p->data);
            if (BF(T) == 2) // Rebalance during windup if(BF(T->left)>=0)
                T = LL(T);
            else
                T = LR(T);
        }
        else
            return (T->left);
    }
    T->ht = height(T);
    return (T);
}
int height(node *T)
{
    int lh, rh;
    if (T == NULL)
        return (0);
    if (T->left == NULL)
        lh = 0;
    else
        lh = 1 + T->left->ht;
    if (T->right == NULL)
        rh = 0;
    else
        rh = 1 + T->right->ht;
}

```

```

    if (lh > rh)
        return (lh);
    return (rh);
}
node *rotateright(node *x)
{
    node *y;
    y = x->left;
    x->left = y->right;
    y->right = x;
    x->ht = height(x);
    y->ht = height(y);
    return (y);
}
node *rotateleft(node *x)
{
    node *y;
    y = x->right;
    x->right = y->left;
    y->left = x;
    x->ht = height(x);
    y->ht = height(y);
    return (y);
}
node *RR(node *T)
{
    T = rotateleft(T);
    return (T);
}
node *LL(node *T)
{
    T = rotateright(T);
    return (T);
}
node *LR(node *T)
{
    T->left = rotateleft(T->left);
    T = rotateright(T);
    return (T);
}
node *RL(node *T)
{
    T->right = rotateright(T->right);
    T = rotateleft(T);
    return (T);
}
int BF(node *T)
{
    int lh, rh;
    if (T == NULL)
        return (0);
    if (T->left == NULL)
        lh = 0;
    else
        lh = 1 + T->left->ht;
    if (T->right == NULL)
        rh = 0;
    else

```

```

        rh = 1 + T->right->ht;
    return (lh - rh);
}
void preorder(node *T)
{
    if (T!= NULL)
    {
        printf("%d(Bf=%d)", T->data, BF(T));
        preorder(T->left);
        preorder(T->right);
    }
}
void inorder(node *T)
{
    if (T!= NULL)
    {
        inorder(T->left);
        printf("%d(Bf=%d)", T->data, BF(T));
        inorder(T->right);
    }
}

```

OUTPUT

```

MINGW64/d/Degree/SEM 3/IT x + v
Abdurrahman Qureshi@GhouledGadget MINGW64 /d/Degree/SEM 3/DSA/Programs (master)
$ ./expl.exe
1)Create:
2)Insert:
3)Delete:
4)Print:
5)Quit:
Enter Your Choice:1
Enter no. of elements:5
Enter tree data:1
4
5
6
7
1)Create:
2)Insert:
3)Delete:
4)Print:
5)Quit:
Enter Your Choice:4
Preorder sequence:
4(Bf=-1)1(Bf=0)6(Bf=0)5(Bf=0)7(Bf=0)
Inorder sequence:
1(Bf=0)4(Bf=-1)5(Bf=0)6(Bf=0)7(Bf=0)
1)Create:
2)Insert:
3)Delete:
4)Print:
5)Quit:
Enter Your Choice:2
Enter a data:8
1)Create:
2)Insert:
3)Delete:
4)Print:
5)Quit:
Enter Your Choice:4

```

```
MINGW64/d/Degree/SEM 3/1 x + v
5)Quit:
Enter Your Choice:4
Preorder sequence:
4(Bf=-1)1(Bf=0)6(Bf=0)5(Bf=0)7(Bf=0)
Inorder sequence:
1(Bf=0)4(Bf=-1)5(Bf=0)6(Bf=0)7(Bf=0)
1)Create:
2)Insert:
3)Delete:
4)Print:
5)Quit:
Enter Your Choice:2
Enter a data:8
1)Create:
2)Insert:
3)Delete:
4)Print:
5)Quit:
Enter Your Choice:4
Preorder sequence:
6(Bf=0)4(Bf=0)1(Bf=0)5(Bf=0)7(Bf=-1)8(Bf=0)
Inorder sequence:
1(Bf=0)4(Bf=0)5(Bf=0)6(Bf=0)7(Bf=-1)8(Bf=0)
1)Create:
2)Insert:
3)Delete:
4)Print:
5)Quit:
Enter Your Choice:3
Enter a data:4
Segmentation fault
Abdurrahman Qureshi@GhouledGadget MINGW64 /d/Degree/SEM 3/DSA/Programs (master)
$ 5
bash: 5: command not found
Abdurrahman Qureshi@GhouledGadget MINGW64 /d/Degree/SEM 3/DSA/Programs (master)
$ |
```

Tools used :

Software: Dev c++

Hardware: Lab Computers

References: Mam notes.

Conclusion

- AVL tree is a descendant of Binary Search Tree but overcomes its drawback of increasing complexity if the elements are sorted.
- It monitors the balance factor of the tree to be 0 or 1 or -1.
- In case it tree becomes unbalanced corresponding rotation techniques are performed to balance the tree.

Name: Abdurrahman Qureshi

Roll No: 242466

Practical No: 8

1) Demonstrate the Implementation of addition and deletion of edges in a directed graph using adjacency matrix.

CODE

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 100
int adj[MAX][MAX];
int n;
void create_graph();
void display();
void insert_edge(int origin, int destin);
void del_edge(int origin, int destin);
int main()
{
    int choice, origin, destin;
    create_graph();
    while (1)
    {
        printf("\n1.Insert an edge\n");
        printf("2.Delete an edge\n");
        printf("3.Display\n");
        printf("4.Exit\n");
        printf("\nEnter your choice : ");
        scanf("%d", &choice);
        switch (choice)
        {
            case 1:
                printf("\nEnter an edge to be inserted : ");
                scanf("%d %d", &origin, &destin);
                insert_edge(origin, destin);
                break;
            case 2:
                printf("\nEnter an edge to be deleted : ");
                scanf("%d %d", &origin, &destin);
                del_edge(origin, destin);
                break;
            case 3:
```

```

        display();
        break;
    case 4:
        exit(1);
    default:
        printf("\nWrong choice\n");
        break;
    }
}
return 0;
}
void create_graph()
{
    int i, max_edges, origin, destin;
    printf("\nEnter number of vertices : ");
    scanf("%d", &n);
    max_edges = n * (n - 1);
    for (i = 1; i <= max_edges; i++)
    {
        printf("\nEnter edge %d(-1 -1) to quit : ", i);
        scanf("%d %d", &origin, &destin);
        if ((origin == -1) && (destin == -1))
            break;
        if (origin >= n || destin >= n || origin < 0 || destin < 0)
        {
            printf("\nInvalid edge!\n");
            i--;
        }
        else
            adj[origin][destin] = 1;
    }
}
void insert_edge(int origin, int destin)
{
    if (origin < 0 || origin >= n)
    {
        printf("\nOrigin vertex does not exist\n");
        return;
    }
    if (destin < 0 || destin >= n)
    {
        printf("\nDestination vertex does not exist\n");
        return;
    }
    adj[origin][destin] = 1;
}
void del_edge(int origin, int destin)
{
    if (origin < 0 || origin >= n || destin < 0 || destin >= n || adj[origin][destin] == 0)
    {
        printf("\nThis edge does not exist\n");
        return;
    }
    adj[origin][destin] = 0;
}
void display()
{
    int i, j;

```

```

    for (i = 0; i < n; i++)
    {
        for (j = 0; j < n; j++)
            printf("%4d", adj[i][j]);
        printf("\n");
    }
}

```

OUTPUT

```

Abdurrahman Qureshi@GhouledGadget MINGW64 /d/Degree/SEM 3/DSA/Programs (master)
$ ./expl.exe

Enter number of vertices : 4
Enter edge 1(-1 -1) to quit : 0 1
Enter edge 2(-1 -1) to quit : 1 2
Enter edge 3(-1 -1) to quit : 2 3
Enter edge 4(-1 -1) to quit : 4 3
Invalid edge!
Enter edge 4(-1 -1) to quit : -1 -1

1.Insert an edge
2.Delete an edge
3.Display
4.Exit
Enter your choice : 3
  0  1  0  0
  0  0  1  0
  0  0  0  1
  0  0  0  0

1.Insert an edge
2.Delete an edge
3.Display
4.Exit
Enter your choice : 0 1 0 0
0

Abdurrahman Qureshi@GhouledGadget MINGW64 /d/Degree/SEM 3/DSA/Programs (master)
$ |

```

Tools used :

Software: Dev c++

Hardware: Lab Computers

References: Mam notes.

Conclusion

- An Adjacency matrix is a square matrix used to represent a finite graph.
- It contains the information about the edges and its cost.
- If the value at the Ith row and Jth column are zero, it means an edge does not exist between these two vertices. Else you got the edge and cost of that edge.

Name: Abdurrahman Qureshi

Roll No: 242466

Practical No: 9

1) Demonstrate the Implementation of infix to postfix conversion and evaluation of postfix conversion.

CODE

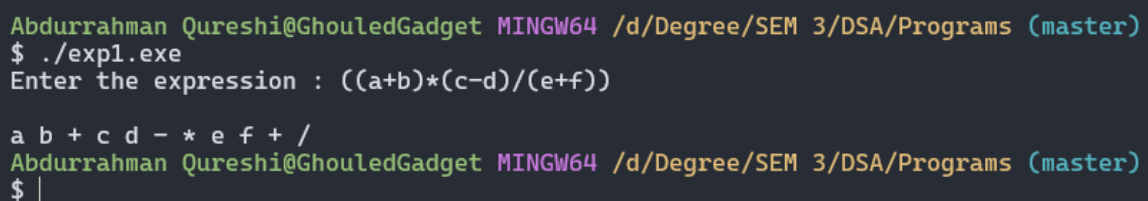
```
#include <stdio.h>
#include <ctype.h>
char stack[100];
int top = -1;
void push(char x)
{
    stack[++top] = x;
}
char pop()
{
    if (top == -1)
        return -1;
    else
        return stack[top--];
}
int priority(char x)
{
    if (x == '(')
        return 0;
    if (x == '+' || x == '-')
        return 1;
    if (x == '*' || x == '/')
        return 2;
    return 0;
}
int main()
{
    char exp[100];
    char *e, x;
    printf("Enter the expression : ");
    scanf("%s", exp);
    printf("\n");
    e = exp;
    while (*e != '\0')
```

```

{
    if (isalnum(*e))
        printf("%c ", *e);
    else if (*e == '(')
        push(*e);
    else if (*e == ')')
    {
        while ((x = pop()) != '(')
            printf("%c ", x);
    }
    else
    {
        while (priority(stack[top]) >= priority(*e))
            printf("%c ", pop());
        push(*e);
    }
    e++;
}
while (top != -1)
{
    printf("%c ", pop());
}
return 0;
}

```

OUTPUT



```

Abdurrahman Qureshi@GhouledGadget MINGW64 /d/Degree/SEM 3/DSA/Programs (master)
$ ./exp1.exe
Enter the expression : ((a+b)*(c-d))/(e+f))

a b + c d - * e f + /
Abdurrahman Qureshi@GhouledGadget MINGW64 /d/Degree/SEM 3/DSA/Programs (master)
$ |

```

Tools used :

Software: Dev c++

Hardware: Lab Computers

References: Mam notes.

Conclusion

Infix to postfix conversion simplifies expressions by eliminating parentheses. Postfix evaluation uses a stack for efficient calculation. This method enhances computational efficiency, especially for evaluating complex expressions in calculators and interpreters.

Name: Abdurrahman Qureshi

Roll No: 242466

Practical No: 10

1) Demonstrate the Implementation of Hashing Function with different collision resolution technique.

CODE

```
#include <stdio.h>
#include <stdlib.h>
struct set
{
    int key;
    int data;
};
struct set *array;
int capacity = 10;
int size = 0;
int i;
int hashFunction(int key)
{
    return (key % capacity);
}
int checkPrime(int n)
{
    int i;
    if (n == 1 || n == 0)
    {
        return 0;
    }
    for (i = 2; i < n / 2; i++)
    {
        if (n % i == 0)
        {
            return 0;
        }
    }
    return 1;
}
int getPrime(int n)
{
    if (n % 2 == 0)
```

```

    {
        n++;
    }
    while (!checkPrime(n))
    {
        n += 2;
    }
    return n;
}

void init_array()
{
    capacity = getPrime(capacity);
    array = (struct set *)malloc(capacity * sizeof(struct set));
    for (i = 0; i < capacity; i++)
    {
        array[i].key = 0;
        array[i].data = 0;
    }
}

void insert(int key, int data)
{
    int index = hashFunction(key);
    if (array[index].data == 0)
    {
        array[index].key = key;
        array[index].data = data;
        size++;
        printf("\n Key (%d) has been inserted \n", key);
    }
    else if (array[index].key == key)
    {
        array[index].data = data;
    }
    else
    {
        printf("\n Collision occurred \n");
    }
}

void remove_element(int key)
{
    int index = hashFunction(key);
    if (array[index].data == 0)
    {
        printf("\n This key does not exist \n");
    }
    else
    {
        array[index].key = 0;
        array[index].data = 0;
        size--;
        printf("\n Key (%d) has been removed \n", key);
    }
}

void display()
{
    int i;
    for (i = 0; i < capacity; i++)
    {

```

```

        if (array[i].data == 0)
        {
            printf("\n array[%d]: / ", i);
        }
        else
        {
            printf("\n key: %d array[%d]: %d \t", array[i].key, i, array[i].data);
        }
    }
}
int size_of_hashtable()
{
    return size;
}
int main()
{
    int choice, key, data, n;
    int c = 0;
    init_array();
    do
    {
        printf("1.Insert item in the Hash Table"
            "\n2.Remove item from the Hash Table"
            "\n3.Check the size of Hash Table"
            "\n4.Display a Hash Table"
            "\n\n Please enter your choice: ");
        scanf("%d", &choice);
        switch (choice)
        {
            case 1:
                printf("Enter key -:\t");
                scanf("%d", &key);
                printf("Enter data -:\t");
                scanf("%d", &data);
                insert(key, data);
                break;
            case 2:
                printf("Enter the key to delete-:");
                scanf("%d", &key);
                remove_element(key);
                break;
            case 3:
                n = size_of_hashtable();
                printf("Size of Hash Table is-:%d\n", n);
                break;
            case 4:
                display();
                break;
            default:
                printf("Invalid Input\n");
        }
        printf("\nDo you want to continue (press 1 for yes): ");
        scanf("%d", &c);
    } while (c == 1);
}

```

OUTPUT


```
Abdurrahman Qureshi@GhouledGadget MINGW64 /d/Degree/SEM 3/DSA/Programs (master)
$ ./expl.exe
1.Insert item in the Hash Table
2.Remove item from the Hash Table
3.Check the size of Hash Table
4.Display a Hash Table

Please enter your choice: 1
Enter key -: 1
Enter data -: 85

Key (1) has been inserted

Do you want to continue (press 1 for yes): 1
1.Insert item in the Hash Table
2.Remove item from the Hash Table
3.Check the size of Hash Table
4.Display a Hash Table

Please enter your choice: 1
Enter key -: 96
Enter data -: 41

Key (96) has been inserted

Do you want to continue (press 1 for yes): 1
1.Insert item in the Hash Table
2.Remove item from the Hash Table
3.Check the size of Hash Table
4.Display a Hash Table

Please enter your choice: 1
Enter key -: 54
Enter data -: 69

Key (54) has been inserted

Do you want to continue (press 1 for yes): 1
1.Insert item in the Hash Table
2.Remove item from the Hash Table
3.Check the size of Hash Table
4.Display a Hash Table

Please enter your choice: 3
Size of Hash Table is:-3

Do you want to continue (press 1 for yes): 4
Abdurrahman Qureshi@GhouledGadget MINGW64 /d/Degree/SEM 3/DSA/Programs (master)
$ |
```

Tools used :

Software: Dev c++

Hardware: Lab Computers

References: Mam notes.

Conclusion

- Hash function is a function that uses the constant-time operation to store and retrieve the value from the hash table, which is applied on the keys as integers and this is used as the address for values in the hash table.
- Hashing is one of the important techniques in terms of searching data provided with very efficient and quick methods using hash function and hash tables.
- Each element can be searched and placed using different hashing methods.
- This technique is very faster than any other data structure in terms of time coefficient.