

EXPERIMENT – 1   
   
  
  
**QUES 1.**Find the sum of all array elements using recursion.  
  
**ALGORITHM:**  
1. START   
2.Define a function sum its parameters   
3 Define a recursive function and call it in the main function  
4. Define the function main and create an array   
5. Take the size of array   
6. Enter the elements of array  
7. End

#include <stdio.h>

int sum(int a[50], int num)

{

static int result = 0,i;

if (i < num)

{

result += a[i];

i++;

sum(a, num);

}

return result;

}

int main()

{

int arr[50], num, addi = 0;

printf("\nEnter the size of the array: ");

scanf("%d", &num);

printf("Enter the elements of the array: \n");

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

{

printf("Enter the element %d :", i + 1);

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

}

printf("Sum of the array elements: %d", sum(arr, num));

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

{

addi = addi + arr[i];

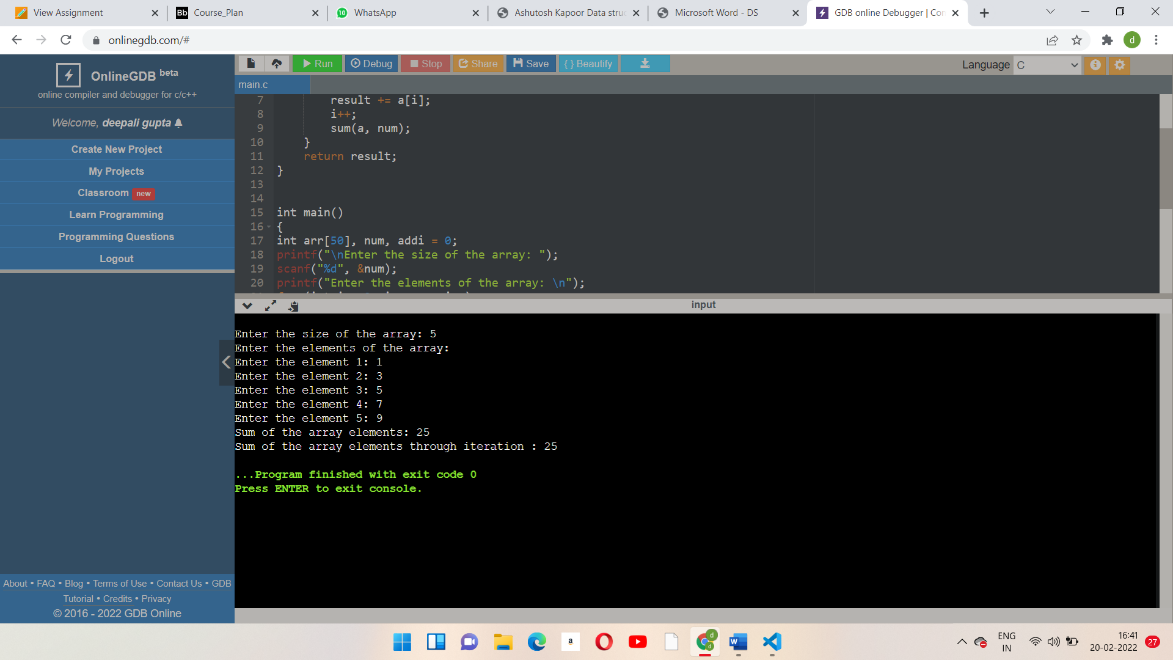
}

printf("\nSum of the array elements through iteration : %d", addi);

}

| Sr no. | No. of elements | Element1 | Element2 | Element3 | Element4 | Element 5 |
| --- | --- | --- | --- | --- | --- | --- |
| 1. | 5 | 1 | 3 | 5 | 7 | 9 |

**TEST CASES :  
OUTPUT:**



**QUES 2.** Create a menu-driven program that works with an integer array ‘a1’ with ‘n’ numbers. A count of the total number of elements in ‘a1’ should be always maintained. (switch-case)   
  
a. Function insert() gets a reference of ‘a1’ through a function call. It obtains a number & the appropriate position to be inserted in ‘a1’ from the user. If an odd number is entered another chance should be given to take input. If elements are available in the given position suitable shifting of elements should be made respectively. Check for overflow. All elements should be together.  
  
 b. Function delete() gets a reference of ‘a1’ and the position in ‘a1’ from where the element is to be removed through a function call. The count is updated and the remaining elements are rearranged accordingly. Check for underflow.  
  
 c. Function copy() gets a reference of ‘a1’ and copies its entire contents into another integer array ‘a2’ of size ‘l’.   
Assume l>n. Further insertion can be made using insert().   
  
d. Function merge() the contents of ‘a1’ and ‘a2’ into a new array  
  
  
**ALGORITHM :**1**.**start   
2. Defining all the variables   
3.taking the size of array and its elements   
4.position where we want to insert the element   
5.taking input of new element and element to be deleted .  
6. Using operations on arrays and if condition with nested for loop get printed all the things .  
7. End

#include<stdio.h>

int main()

{

    int a1[100], num, C, i\_position, d\_position, n\_element, i, m = 0;

     printf("\nEnter the size of the array: ");

    scanf("%d", &num);

    printf("Enter the elements of the array: \n");

    for (int i = 1; i < num; i++)

{

    printf("Enter the element %d: ", i);

 scanf("%d",&a1[1]);

}

printf("Enter the position where you have to insert a new element :");

scanf("%d", &i\_position);

printf("Enter the new element :");

scanf("%d", &n\_element);

for (i=num + 1; i > -i\_position; i--)

{

    a1[i] = a1[i - 1];

}

a1[i\_position] = n\_element;

printf("Array al after insertion is:");

for (i = 1; i < num + 1; i++)

{  printf("%d", a1[1]);

}

num = num + 1;

 printf("\nEnter the position where you have to delete the element: ");

scanf("%d", &d\_position);

for (i = d\_position; i < num; i++)

{

    a1[i] = a1[i + 1];

}

printf(" Array al after deletion of an element :");

for (i = 1; i < num; i++)

{

    printf("%d ", a1[inum = num - 1;

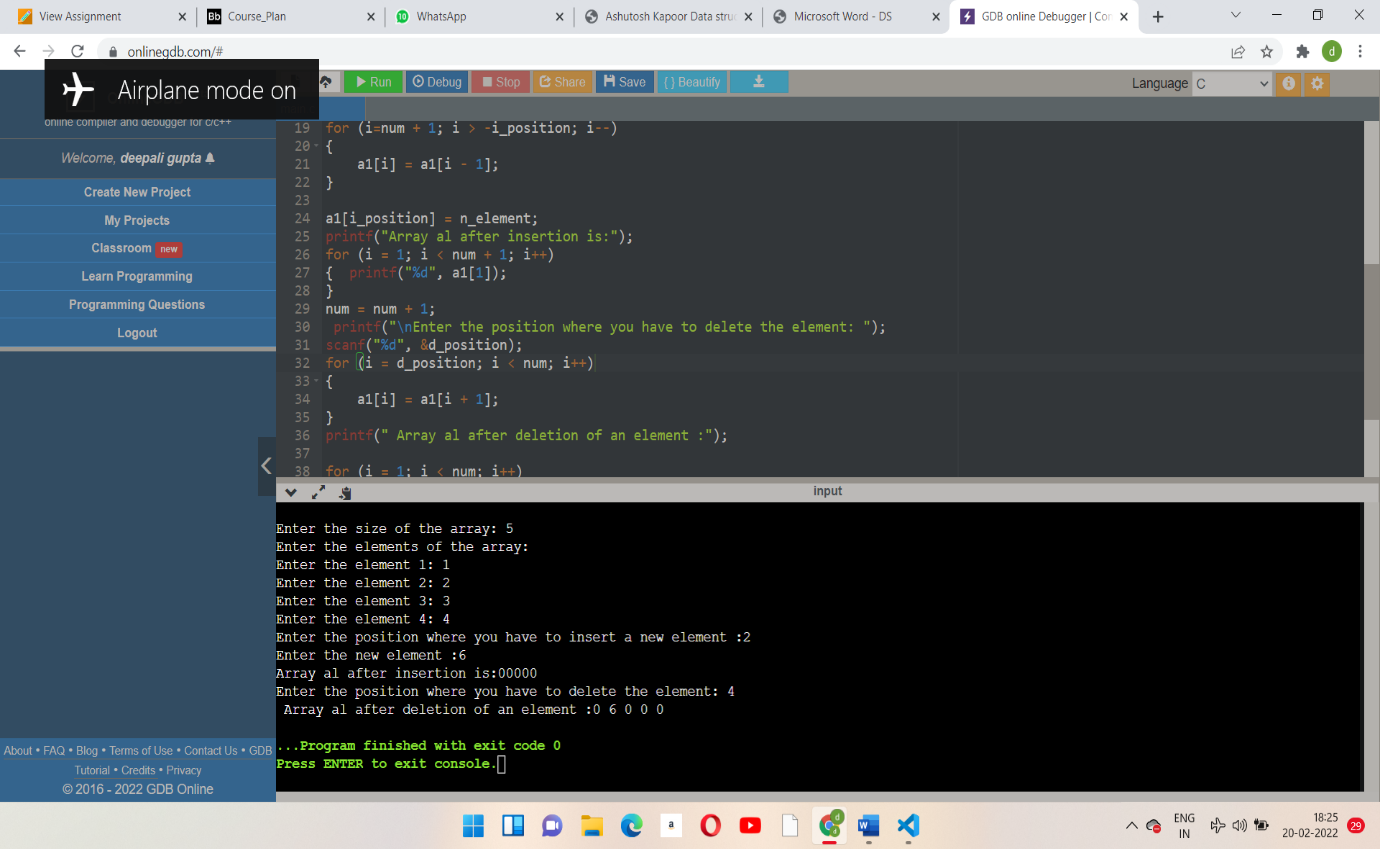
return 0;

}

**TEST CASES :**

| SR.NO | ELEMENT 1 | ELEMENT 2 | ELEMENT 3 | ELEMENT 4 |
| --- | --- | --- | --- | --- |
| 1. | 1 | 2 | 3 | 4 |

**OUTPUT :**

****

**QUES 3.** Convert uppercase string to lowercase using for loop (do not use built-in functions).  
  
ALGORITHM :  
1. Start   
2. Create a character string using array  
3. Take unput of uppercase letter   
4. Using fgets command take input upto newline character   
5. Use if condition and take conversions and print it   
6. End

#include<stdio.h>  
#include <string.h>

int main()

{

char arr[30];

printf("Enter the uppercase string : ");

fgets(arr, 30, stdin);

for (int i = 0; i<strlen(arr); i++)

{

    if (arr[i] >=65 && arr[1] <=90)

        arr[i] = arr[i] + 32;

}

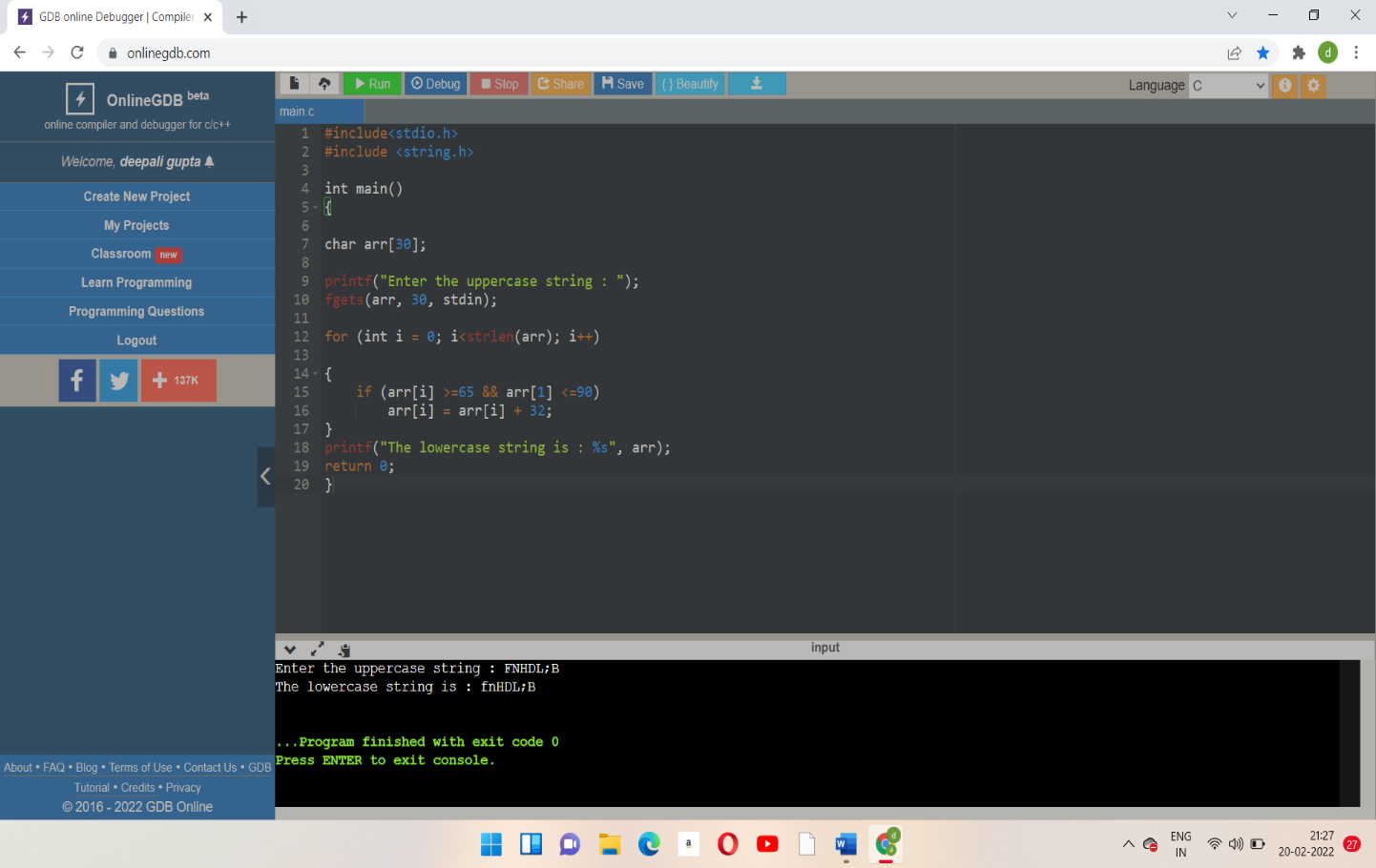
printf("The lowercase string is : %s", arr);

return 0;

}

**TEST CASES :**

| SR.NO. | ELEMENT 1 | ELEMENT 2 | ELEMENT 3 | ELEMENT 4 | ELEMENT 5 |
| --- | --- | --- | --- | --- | --- |
| 1. | DEEPALI | SHUBHAM | VANSHIKA | GAURI | SHIVAM |

**OUTPUT :  
  
  
  
  
  
  
  
  
  
  
  
  
QUES 4.** Find the product of two matrices using pointers.  
  
**ALGORITHMS :**1.start   
2. Defining all the variables   
3.taking inputs from user of size of row of matrix and column of matrix .  
4.taking inputs of elements of matrix using for loop  
5.using nested for loop get the matrix and multiply  
6. End

#include <stdio.h>

int main()

{

    int matrix[50][50],a,b,row,col,row\_sum[50],col\_sum[50];

    printf("\nEnter the size of the row of the matrix(max 5): ");

    scanf("%d", &row);

    printf("Enter the size of the column of the matrix(max 5): ");

    scanf("%d", &col);

    if ((row < 1) && (col < 1))

    {

        printf("The size of the row and the column of the matrix cannot be negative!");

    }

    else if (row < 1)

    {

        printf("The size of the row of a matrix cannot be negative!\n");

    }

    else if (col < 1)

    {

        printf("The size of the column of the matrix cannot be negative !");

    }

    else

    {

        printf("enter the elements of matrix: \n");

        for (a = 0; a < row; a++)

        {

            for (b=0; b<column; b++)

            {

                printf("enter the elements as[% d][% d]: ", a, b);

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

            }

        }

        for (a = 0; a < row; a++)

        {

            row\_sum[a] = 0;

            for (b = 0; b < column; b++)

            {

                row\_sum[a] += matrix[a][b];

            }

        }

        for (b = 0; b < column; b++)

        {

            col\_sum[b] = 0;

            for (a = 0; a < row; a++)

            {

                col\_sum[b] += matrix[a][b];

            }

        }

        for (a=0; a<row; a++)

        {

            printf("\n sum of row % d is % d", a + 1, row\_sum[a]);

        }

        for (b=0; b<column; b++)

        {

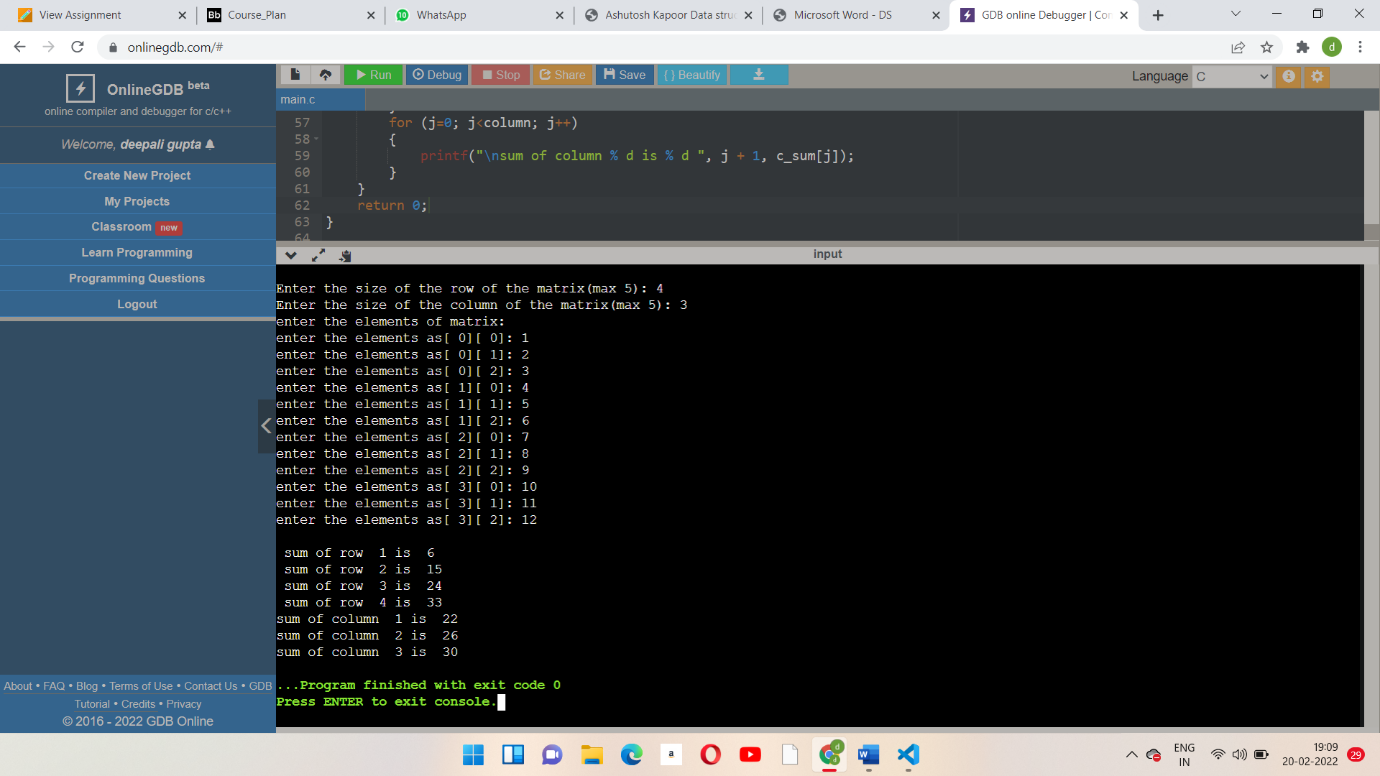
            printf("\n sum of column % d is % d ", b + 1, col\_sum[b]);

        }

    }

    return 0;

}

**OUTPUT :  
** EXPERIMENT – 2   
  
Q1. Design a structure to represent a complex number. Get two complex numbers and find the sum of them using a function to which structure is passed as argument.  
  
ALGORITHM :  
1. Start   
2. Defining the function complex\_num  
3. Defining a structure complex\_num  
4. Defining the function main and initializing the variables   
5. Taking all the inputs and printing all the outputs .  
6. End   
   
**CODE :**#include <stdio.h>  
Complex\_num add\_complex (complex\_num cmplx1,complex\_num cmplx2)

{

Complex\_num final;

final.real = cmplx1.real + cmplx2.real;

final.img = cmplx1.imgi + cmplx2.imgi;

return final;

}

struct complex\_num

{

float real, imgi;

}complex;

int main()

{

Complex\_num cmplx1, cmplx2,cmplx\_final;

printf(“Enter the first complex number: \n”);

printf(“Enter the real part: “);

scanf(“%f”,&cmplx1.real);

printf(“\nEnter the imaginary part: “);

scanf(“%f”,&cmplx1.imgi);

printf(“\nComplex number 1 = %.2f + (%.2f)I”,cmplx1.real , cmplx1.imgi);

printf(“\nEnter the second complex number: \n”);

printf(“Enter the real part: “);

scanf(“%f”,&cmplx2.real);

printf(“\nEnter the imaginary part: “);

scanf(“%f”,&cmplx2.imgi);

printf(“\nComplex number 1 = %.2f + (%.2f)I”,cmplx2.real , cmplx2.imgi);

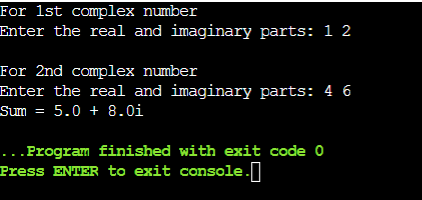
cmplex\_final = add\_complex(cmplx1,cmplx2);

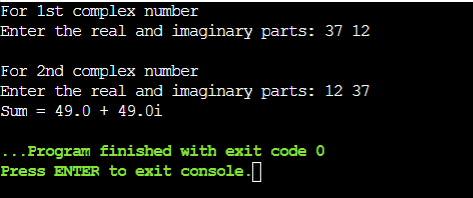
printf(“\nSUM of both complex numbers = %0.2f + %0.2f I”, cmplx\_final.real, cmplx\_final.imgi);

return 0;  
}

Test Case:

| Sr no. | Complex Number 1 | | Complex number 2 | | Final Complex Number |
| --- | --- | --- | --- | --- | --- |
|  | Real Part | Imaginary part | Real Part | Imaginary part |  |
| 1 | 12.59 | 25.63 | 78.45 | 14.65 | 91.04 + 40.28 i |

OUTPUT :  
  
  




Q2. Design a structure named ‘student\_record’ to store student details like name, sap- id, enrollment-number, birth-date, registration-date, and marks of five subjects. The members, birth date, and registration date are defined using another structure named ‘date’ to store date details like day, month, and year. Get data of ‘n’ students, compute the average marks obtained by each of them and print their grade sheets.

**Note: Use Nested Structures, each student record is dynamically allocated.  
  
ALGORITHM :  
1. Start   
2. Declare a structure named registration , structure date of birth, structure marks , structure student marks , structure student record   
3. Defining the function main and initializing all the variables .  
4. Using for loop and taking inputs from user   
5. Printing all the outputs   
6. End**

Code:

#include <stdio.h>

#include <string.h>

#include <stdlib.h>  
#include <math.h>

struct registration

{

int date, year;

char month[20];

}reg;

struct date\_of\_birth

{

int date, year;

char month[10];

}dob;

struct marks

{

int maths,phy,chem,comp,bio;

}mark;

struct student\_record

{

char name[30];

int sapid, enroll\_no;

struct registeration reg;

struct date\_of\_birth dob;

struct marks mark;

}stu\_rec;

int main()

{

int i,r;

struct student\_record stu\_rec;

printf ("Enter the number of records to be stored: ");

scanf("%d",&r);

struct student\_record \*stu;

stu = (struct student\_record \*) malloc(n \* sizeof(struct student\_record));

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

{

printf("\nEnter the name of student: ");

scanf("%s",stu[i].name);

printf("Enter the sapid of the student: ");

scanf("%d",&stu->sapid);  
  
 printf("Enter the registeration month: ");

scanf("%s",&stu->reg.month);

printf("Enter the enrollment number: ");

scanf("%d",&stu->enroll\_no);  
   
 printf("Enter the sapid of the student: ");

scanf("%d",&stu->sapid);  
  
   
 printf("Enter the registeration date and year resp.: ");

scanf("%d %d",&stu->reg.date,&stu->reg.year);

printf("Enter month of birth: ");

scanf("%s",&stu->dob.month);

printf("Enter the date and year of birth resp.: ");

scanf("%d %d",&stu->dob.date,&stu->dob.year);

printf("\nEnter the marks as following(maths,physics,chemistry,computer,biology): ");

scanf("%d %d %d %d %d",&stu->mark.maths,&stu->mark.phy,&stu->mark.chem,&stu->mark.comp,&stu->mark.bio);

int total = stu->mark.maths + stu->mark.phy + stu->mark.chem + stu->mark.comp + stu->mark.bio;

float avg = total/5;

printf("\nName: %s",stu[i].name);

printf("\nSAPID: %d",stu->sapid);

printf("\nEnrollment number: %d",stu->enroll\_no);

printf("\nRegisteration Date: %s %d %d",stu->reg.month,stu->reg.date,stu->reg.year);

printf("\nDate of Birth: %s %d %d",stu->dob.month,stu->dob.date,stu->dob.year);

printf("\nAverage marks: %f\n",avg);

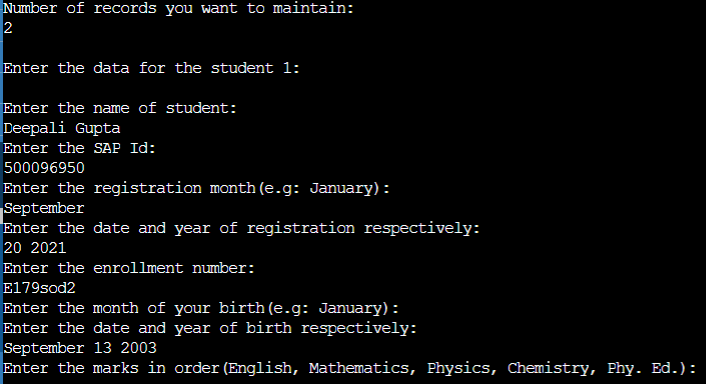
}

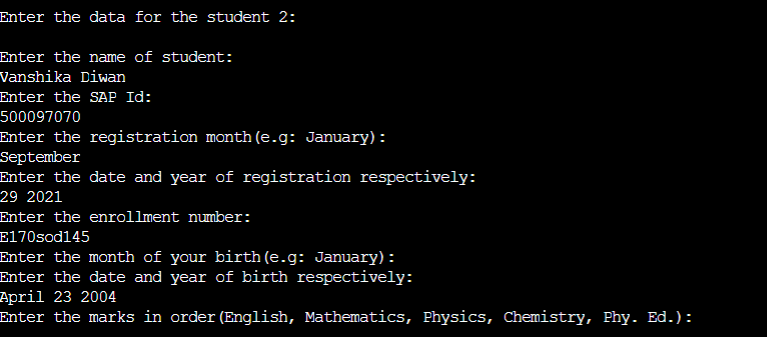
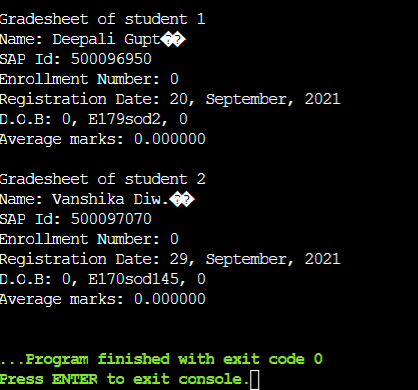
return 0;

}  
  
  
Test Case:

| Sr no. | Name | SAPID | Enroll\_no | Reg date | Date of Birth |
| --- | --- | --- | --- | --- | --- |
| 1 | Deepali | 500097111 | 2142211260 | Oct 01 2021 | May 09 2003 |
| 2 | vanshika | 500096953 | 2142211209 | Sept 27 2021 | May 15 2003 |
| 3 | afreen | 500096959 | 2142211216 | Sept 24 2021 | Oct 25 2003 |

| Sr no. | Name | Marks | | | | |
| --- | --- | --- | --- | --- | --- | --- |
| maths | physics | Chemistry | computer | biology |
| 1 | Deepali | 96 | 92 | 91 | 93 | 90 |
| 2 | Vanshika | 95 | 93 | 92 | 90 | 90 |
| 3 | afreen | 93 | 95 | 92 | 90 | 90 |

Output :  
  


Q4. Design a union ‘product’ to store the details of the product purchased like product name, price per unit, the number of quantities purchased, and amount spent. Get the name, price per unit, and the number of quantities of the product purchased. Calculate the amount spent on the product and then display all the details of the procured product.  
  
ALGORITHM :  
1. Start   
2. Starting main function   
3. Defining a union named prct  
4. Initializing the variables   
5. Declaring pointer variables   
6. Taking all inputs from the user and print all the things   
7. End

Code:

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

int main()

{

union product

{

char prct\_name[25];

int qunty;

float price,amnt;

};

union prct \*t , p;  
 int a,b;

t = &p;

printf("Enter the product name: ");

scanf("%[^\n]s", p.prct\_name);

printf("Product name: %s\n",t->prct\_name);

printf("\nEnter the quantity: ");

scanf("%d",&p.qunty);

b = t->qunty;

printf("Quantity: %d\n",t->qunty);

printf("\nEnter the price: ");

scanf("%f",&p.price);

a = t->price;

printf("price: %f\n",t->price);

p.amnt = a\*b;

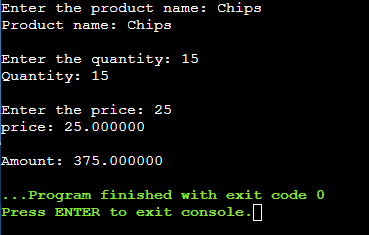
printf("\nAmount: %f",t->amnt);

return 0;

}

Test Case:

| Sr. no | Product Name | Quantity | Price | Amount |
| --- | --- | --- | --- | --- |
| 1 | oreo | 9 | 20.8 | 180.000000 |

output:  


Q3. Consider two quadratic equations (Eq1, Eq2) with two positive roots each. Their roots are related as given below:

1. One root is common for both equations (choose the biggest positive root during derivation), (ii) The uncommon positive root of Eq1 and Eq2 is the same as the two different positive roots of a third given Quadratic Equation (say x2−15x+56=0).
2. Sum of the product (SOP) of all four roots taken two at a time is given.
3. Find the sum of all the four roots and the quadratic equations Eq1 and Eq2

**ALGORITHM :**1. Start   
2. Defining all the libraries   
3. Defining a structure and defining all the functions   
4. Defining main function , taking all the inputs of real and imaginary part of a complex number .  
5. Applying conditions according to the expected results using if condition.   
6. Printing all the inputs .   
7. End Code:

#include <stdio.h>

#include <math.h>

#include <stdlib.h>

int chk=0;

struct eqn

{

int a,b,c,root1,root2;

struct eqn\* next;

};

int dplay(struct eqn\* en)

{

printf("The quadratic equation is: %dx^2 + %dx + %d",en->a,en->b,en->c);

return 0;

}

int cal\_eqn(struct eqn\* en)

{

en->a = 1;

en->b = -(en->root1 + en->root2);

en->c = (en->root1\*en->root2);

return 0;

}

int cal\_root(struct eqn\* en1, struct eqn\* en2, struct eqn\* sop,int sum)

{

float d;

d= ((en1->root2 + en2->root2)\*(en1->root2 + en2->root2)) - (4\*(en1->root2\*en2->root2));

if(d>=0)

{

sop->root1 = -((en1->root2 + en2->root2) + sqrt(d))/2;

sop->root2 = -((en1->root2 + en2->root2) - sqrt(d))/2;

}

else

{

chk=1;

}

return 0;

}

int main()

{

struct eqn eq1, eq2, eq3, sop;

float sum, d;

printf("Enter the coffecient of x^2: ");

scanf("%d",&eq3.a);

printf("Enter the coffecient of x: ");

scanf("%d",&eq3.b);

printf("Enter the constant term: ");

scanf("%d",eq3.c);

d = ((eq3.b\*eq3.b) - (4\*eq3.a\*eq3.c-sum));

if(d<0)

{

printf("The roots are complex!");

exit(0);

}

if(d==0)

{

eq3.root1 = -(eq3.b)/2\*eq3.a;

eq3.root2 = -(eq3.b)/2\*eq3.a;

if(eq3.root1<0 ||eq3.root2<0)

{

printf("Roots are negaive! ");

exit(0);

}

}

if(d>0)

{

eq3.root1 = (-(eq3.b) - sqrt(d))/(2\*eq3.a);

eq3.root1 = (-(eq3.b) + sqrt(d))/(2\*eq3.a);

if(eq3.root1<0 ||eq3.root2<0)

{

printf("Roots are negaive! );

exit(0);

}

}

printf("\nEnter sum of products: ");

scanf("%f", &sum);

eq1.root2=eq3.root1;

eq2.root2=eq3.root2;

cal\_root(&eq1, &eq2, &sop, sum);

if(check==1)

{

printf("no solution ");

exit(0);

}

if(sop.root2>0)

{

printf("\nWhen common root is: %f", sop.root2);

eq1.root1=eq2.root1=sop.root2;

cal\_eqn(&eq1);

cal\_eqn(&eq2);

dplay(&eq1);

dplay(&eq2);

}  
 if(sop.root1>0)

{

printf("\nWhen common root is: %f", sop.root1);

eq1.root1=eq2.root1=sop.root1;

cal\_eqn(&eq1);

cal\_eqn(&eq2);

dplay(&eq1);

dplay(&eq2);

}

return 0;  
}

EXPERIMENT – 3  
  
**QUES 1.** Implement a single Linked List data structure and its operations like insert and delete in the beginning/end and nth position of the list, and display the items stored in the linked list.  
  
ALGORITHM:  
FOR INSERTION :  
1. IN THE BEGINNING :  
2. IN THE END :  
3. IN THE MIDDLE:  
  
FOR DELETION:  
1.FROM TYE BEGINNING   
2. FORM THE END   
3. FROM THE MIDDLE

CODE :

#include<stdio.h>

#include<stdlib.h>

struct node

{

    int n;

    struct node \*next;

}\*new, \*first=0,\*ptr,\*temp;

void insertend();

void insertmid();

void insertbegin();

void deleteend();

void deletemid();

void deletebegin();

void createblock();

void display();

int count=0;

int main()

{

    int choice;

    L1:printf("\nchoices:\n 1.insert at the end \n 2.insert at the mid \n 3. insert at the begin\n 4. display,any : exit 5.delete from begin\n 6.\ndelete from end\n 7.delete from mid\n 8.display and exit");

    printf("Enter ur choice :");

    scanf("%d",&choice);

    switch(choice)

    {

        case1:

            insertend();

            break;

        case2:

            insertmid();

            break;

        case3:

            insertbegin();

        case4:

            display();

            break;

        default:

        printf("close program\n");

        exit(EXIT\_SUCCESS);

    }

    goto L1;

    return 0;

}

void createblock()

{

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

    printf("enter the data:");

    scanf("%d",&new->n);

    new->next=0;

    count++;

}

void insertend()                                                                   {

    createblock();

    if(first==0)

    {

    first=new;

    }

    else

    {

        for(ptr=first;ptr->next!=0;ptr=ptr->next);  //traversal

        ptr->next=new;

    }

}

void insertbegin()

{

    createblock();

    if(first==0)

    {

        first=new;

    }

    else

    {

        new->next=first;

        first=new;

    }

}

void insertmid()

{

    int pos;

    printf("\nenter position\n")

    scanf("%d",&pos);

    if(pos<1)

    {

    printf("\nwrong input\n");

    }

    else(pos>count+1)

    printf("\n wrong input\n")

    else if(pos==1)

    insertbegin();

    else

    {

        int i=1;

        for(ptr=first;i<pos-1;ptr=ptr->next,i++);  //traversal

        if(ptr->next==0)

        {

        insertend();

        }

        else

        {

            createblock();

            temp=ptr->next;

            ptr->next=new;

            new->next=temp;

        }

    }

void display()

{

    if(first==0)

   {

    printf("list is empty\n");

   }

    else

    {

        printf("list %d elements:",count);

        for(ptr=first;ptr!=0;ptr=ptr->next)   //traversal

        {

            printf("%d",ptr->n);

        }

        printf("\n")

    }

}

void deletebegin()

{

    //for empty linked list

    if(first==0)

    {

        printf("Linked list has no elements..\n");

    }

    //one element in the linked list

    else if(first->next==0)

    {

        printf("Data of deleted node: %d\n",first->no);

        free(first);

        first=0;

        count=0;

    }

    else

    {

        printf("Data of deleted node: %d\n",first->no);

        temp = first->next;

        free(first);

        first = temp;

        count = count-1;

    }

}

void deleteend()

{

    //for empty linked list

    if(first==0)

    {

        printf("Linked list has no elements..\n");

    }

    //one element in the linked list

    else if(first->next==0)

    {

        printf("Data is deleted node: %d\n",first->no);

        free(first);

        first=0;

        count=0;

    }

    //more than one elements in the linked list

    else

    {

        for (ptr=first; ptr->next->next !=0; ptr=ptr->next);

        printf("Data of deleted node: %d\n",ptr->next->no);

        free(ptr->next);

        ptr->next=0;

        count = count-1;

    }

}

void  deletemiddle()

{

    int pos;

    printf("Enter the position: ");

    scanf("%d ", &pos);

    if(pos<1)

    {

        printf("wrong input...\n");

    }

    else if(pos>count)

    {

        printf("Wrong input...\n");

    }

    else if (pos==1)

    {

        deletebegin();

    }

    else if (pos==count)

    {

        deleteend();

    }

    else

    {

        int i=1;

        for(ptr=first;i<pos-1;ptr = ptr->next,i++);

        printf("Data of deleted node: %d\n",ptr->next->no);

        temp = ptr->next;

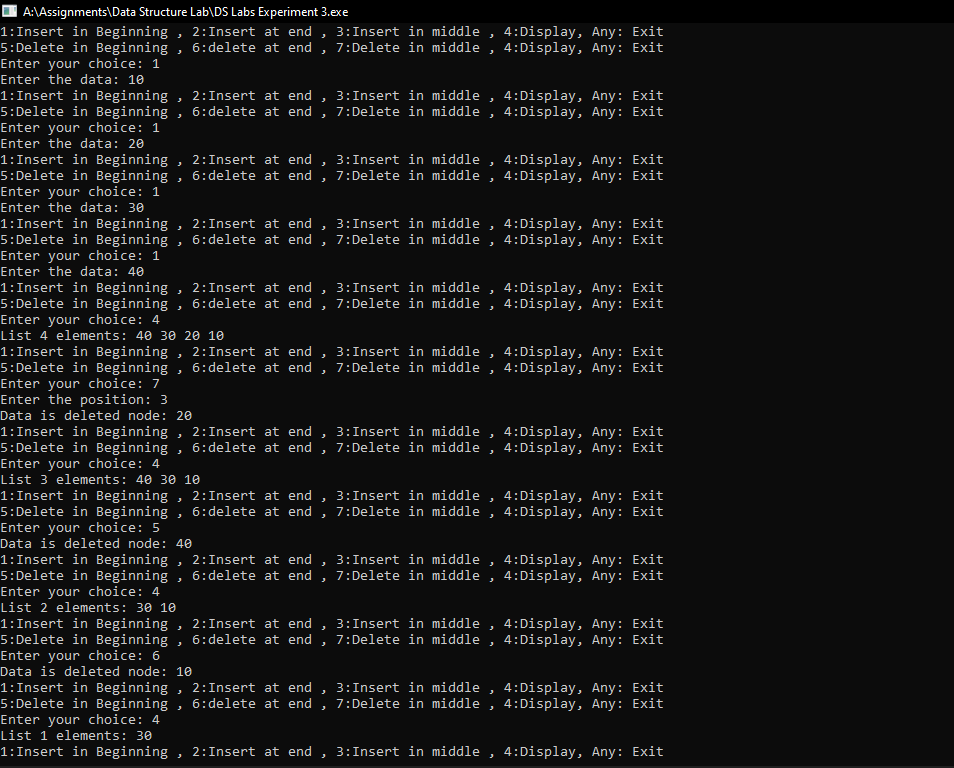
        ptr->next = ptr->next->next;

        free(temp);

        count = count - 1;

    }

}

output:  


QUES 2. Using a single linked list and functions implement Stack and its operations like insert, delete, and display.  
CODE :

#include<stdio.h>  
#include<stdlib.h>  
void push();  
void pop();

void display();

struct node

{

    int val;

    struct node\*next;

}\*head; // it has insertion at the begin and deletion at the end .

void main()

{

    int choice=0;

      printf("\nchoose one from below option....\n");

        printf("\n1.push\n2.pop\n3.show\n4.exit");

        printf("enter your choice:")

        scanf("%d",&choice);

        switch(choice)

        {

            case1:

            {

                push();

                break;

            }

             case2:

            {

                pop();

                break;

            }

             case3:

            {

                display();

                break;

            }

             case4:

            {

                printf("exiting");

                break;

            }

            default:

            printf("please enter valid choice");

        };

}

void push()

{

    int value;

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

    if(ptr==NULL)

    {

        printf("Cannot push into stack..\n");

    }

    else

    {

        printf("Enter the value:");

        scanf("%d",&value);

        if(head == NULL)

        {

            ptr->value = value;

            ptr->next = NULL;

            head = ptr;

        }

        else

        {

            ptr->value = value;

            ptr->next =head;

            head = ptr;

        }

        printf("Value pushed successfully...\n");

    }

}

void pop()

{

    int element;

    struct node \*ptr;

    if(head==NULL)

    {

        printf("Stack is empty \nNo element of pop out...\n ");

    }

    else

    {

        element = head->value;

        ptr = head;

        head = head->next;

        free(ptr);

        printf("Last Element successfully popped/deleted from the stack...\n");

    }

}

void display()

{

    int i;

    struct node \*ptr;

    ptr = head;

    if(ptr==NULL)

    {

        printf("The stack is empty.. Nothing to display...\n");

    }

    else

    {

        while(ptr!=0)

        {

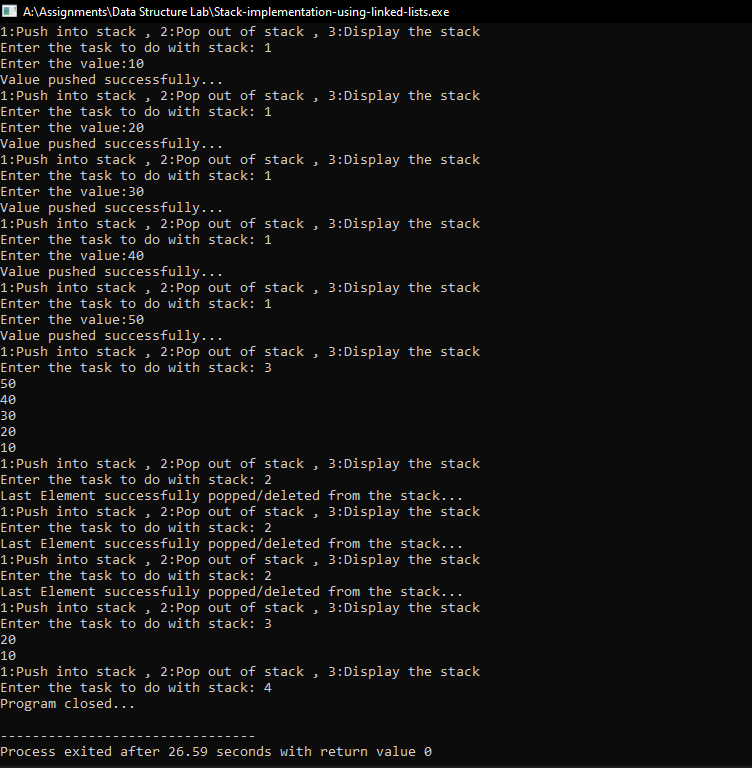
            printf("%d\n",ptr->value);

            ptr = ptr->next;

        }

    }

}

output:  
  
  
  
LIST OF PRACTICE EXECISES   
1. Add two polynomials using Linked List  
  
  
  
   
EXPERIMENT – 4  
  
QUES 1. Using array and functions implement Stack and its operations like insert, delete, and display  
  
ALGORITHM :  
1) Start

2) Create an empty stack using array and functions  
3) One by one create an operation like insert to stack.

4) One by one pop an operation like delete.

5) One by one pop an operation like display.

6) Merge them all in one program

7) Stop

CODE :   
#include<stdio.h>

#include<stdlib.h>

void push();

void pop();

void display();

struct node

{

int val;

struct node\*next;

}\*head; // it has insertion at the begin and deletion at the end .

void main()

{

int choice=0;

while(choice!=4)

{

printf("\nchoose one from below option....\n");

printf("\n1.push\n2.pop\n3.show\n4.exit");

printf("enter your choice:")

scanf("%d",&choice);

switch(choice)

{

case1:

{

push();

break;

}

case2:

{

pop();

break;

}

case3:

{

display();

break;

}

case4:

printf("exiting");

break;

}

default:

printf("please enter valid choice");

};

}

}

void push()

{

#include<stdio.h>

#include<stdlib.h>

#include<conio.h>

int stack[100],i,j,choice=0,n,top=-1;

void push();

void pop();

void show();

void main()

{

printf("enter the elements in the stack");

scanf("%d",&n);

printf("\*\*\*\*\*\*\*stack operation using array\*\*\*\*\*");

printf("\n---------------------------------\n");

printf("Chose one from the below options...\n");

printf("\n1.Push\n2.Pop\n3.Show\n4.Exit");

printf("\n Enter your choice \n");

scanf("%d",&choice);

switch(choice)

{

case 1:

{

push();

break;

}

case 2:

{

pop();

break;

}

case 3:

{

show();

break;

}

case 4:

{

printf("Exiting....");

break;

}

default:

{

printf("Please Enter valid choice ");

}

}

}

void push()

{

int val;

if(top==n)

printf("\noverflow");

else

{

printf("enter the value");

scanf("%d",&val);

top=top+1;

stack[top]=val;

}

}

void pop()

{

if(top==-1)

printf("underflow\n");

else

{

printf("The popped element is %d",stack[top]);

top--;

}

}

void show()

{

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

{

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

}

if(top==-1)

printf("\nstack is empty");

}

QUES 2. Reverse a string using a stack implemented with Dynamic 1D Array.  
  
ALGORITHM :  
1) Start

2) Create an empty stack.

3) One by one push all characters of string to stack.

4) One by one pop all characters from stack and put them back to string. and using some inbuilt functions .  
5) Stop

CODE:  
#include <stdio.h>

#include <string.h>

void swapstr(char \*A, char \*B)

{char temp = \*A;

\*A = \*B;

\*B = temp;  
}

void reversestr(char \*str, int i, int j){

if(i<j)

{swapstr(&str[i],&str[j]);

reversestr(str, i+1, j-1);

}

}  
int main(){

char str[] = "Deepali";

reversestr(str,0, strlen(str)-1);

printf("%s",str);

return 0;

}

OUTPUT:  


LIST OF PRACTICE ACTIVITIES  
2. Convert infix to postfix expression using stack implemented using Linked List.  
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 ",

EXPERIMENT – 5  
  
QUES 1. Using array and functions implement Queue data structure and its operations like insert, delete, and display.  
  
ALGORITHM:  
1.defining a array queue   
2.taking input of size of array and choice  
3. for inserting firstly checking the overflow and then checking if queue id empty or less than size and inserting the element   
4.for deletion checking the underflow, then if only one element, and then deleting the first element entered.  
5. for displaying , traversing the queue and printing all the elements .  
CODE:  
#include<stdio.h>

#include<stdlib.h>

#include<conio.h>

int queue[size],front=-1,rear=-1,count=0;

void enqueue(int\*queue,int\*size)

void dequeue(int\*queue)

void display(int\*queue)

void main()

{

int size;

printf("enter the size of array:");

scanf("%d",&size);

int choice;

printf("enter your choice:");

scanf("%d",&choice);

printf("\noptions:\n1.insert an element\n2.delete an element\n3.display");

switch(choice)

{

case 1:

enqueue(queue,size);

break;

case 2:

dequeue(queue);

break;

case 3:

display(queue);

break;

default:

printf("closing program!!!!");

}

}

void enqueue(int\*queue,int size)

{

if(rear==size-1)

{

printf("queue overflow");

}

else

{

if(front==-1)

{

front++;

rear++;

printf("enter the element to inssert in queue");

scanf("%d",queue[rear]);

}

}

}

void dequeue(int\*queue)

{

int i;

if(front==-1)

{

printf("queue underflow");

}

else if

{

if(front==rear=0)

printf("deleted element is :%d",queue[front]);

rear--;

}

else

{

printf("deleted element is %d",queue[front]);

for(i=0;i<=rear;i--)

{

q[i] = q[i+1];

}

rear--;

}

}

void display(int\*queue)

{ int i;

if (front == - 1)

printf("Queue is empty \n");

else

{

printf("Queue elements are : \n");

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

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

printf("\n");

}

}

QUES 2.Using array and functions implement Circular Queue data structure and its operations like insert, delete, and display.  
  
ALGORITHMS :  
1.defining array   
2. taking input of size and choice from user   
3.for enqueuing checking the overflow and then inserting using a formula of finding index because using index like if we have size 5 and diving it with the count like 0 if it is , it will give remainder as 0 and element will be inserted there and suppose count=5 then remainder will be again zero again pointing to first element after traversing the complete queue which will form the circular queue.   
and same we apply for dequeing .  
4. and for displaying traversing again and printing all the elements .   
  
CODE:  
#include <stdio.h>

#include <stdlib.h>

#include <string.h>

int in = 0, out = 0;

int count = 0;

int size;

void enqueue(int \*q, int size);

void dequeue(int \*q);

void display(int \*q);

int main()

{

printf("Enter the size of the queue: ");

scanf("%d",&size);

int q[size];

L1 : printf("1:Insert , 2:Delete , 3:display , Any:Exit\n");

int choice;

printf("Enter your choice: ");

scanf("%d",&choice);

switch (choice){

case 1:

enqueue(q, size);

break;

case 2:

dequeue(q);

break;

case 3:

display(q);

break;

default:

printf("Program closed..");

exit(EXIT\_SUCCESS);

}

goto L1;

return 0;

}

void enqueue(int \*q, int size)

{

if(count == size)

{

printf("overflow\n");

}

else

{

printf("Enter the element: ");

scanf("%d",&q[in]);

in = (in + 1)%size;

count+=1;

}

}

void dequeue(int \*q)

{

if(count==0)

{

printf("The queue is empty..\n");

}

else

{

printf("deleted element is: %d\n",q[in]);

in = (in+1)%size;

count -= 1;

}

}

void display(int \*q)

{

if(count == 0)

{

printf("The Queue is empty..\n");

}

else if(out>in)

{

printf("The elements are: \n");

for(int i=in;i<=out;i++)

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

}

else

{

printf("The Elements are: \n");

for(int i=in;i<size;i++)

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

}

}

QUES 2.Using array and functions implement Priority Queue and its operations (insert, delete, display)  
ALGORITHM :  
1. defining an array  
2. declaring all the functions  
3. taking input from user of size and choice   
4. creating a queue  
5.for inserting element, checking overflow , then if queue is empty then inputted element will be the first one , else checking for the priority .  
5. we will check priority by traversing all the elements if element is greater than the element then it will move to next element else it will be placed there and all other elements will be shifted to right and value of rear will increase   
6. same for dequeing the lower priority element will be deleted at the first and all the elements will be shifted to left and rear value will decrease   
7. displaying is done by traversing and printing all the elements .   
  
  
CODE:  
#include <stdio.h>

#include <stdlib.h>

#define MAX 10

void create\_queue();

void insert\_element(int);

void delete\_element(int);

void check\_priority(int);

void display\_priorityqueue();

int pqueue[MAX];

int front, rear;

void main()

{

int n, choice;

printf("\nEnter 1 to insert element by priority ");

printf("\nEnter 2 to delete element by priority ");

printf("\nEnter 3 to display priority queue ");

printf("\nEnter 4 to exit");

create\_queue();

while (1)

{

printf("\nEnter your choice : ");

scanf("%d", &choice);

switch (choice)

{

case 1:

printf("\nEnter element to insert : ");

scanf("%d", &n);

insert\_element(n);

break;

case 2:

printf("\nEnter element to delete : ");

scanf("%d", &n);

delete\_element(n);

break;

case 3:

display\_priorityqueue();

break;

case 4:

exit(0);

default:

printf("\n Please enter valid choice");

}

}

}

void create\_queue()

{

front = rear = -1;

}

void insert\_element(int data)

{

if (rear >= MAX - 1)

{

printf("\nQUEUE OVERFLOW");

return;

}

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

{

front++;

rear++;

pqueue[rear] = data;

return;

}

else

check\_priority(data);

rear++;

}

void check\_priority(int data)

{

int i, j;

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

{

if (data >= pqueue[i])

{

for (j = rear + 1; j > i; j--)

{

pqueue[j] = pqueue[j - 1];

}

pqueue[i] = data;

return;

}

}

pqueue[i] = data;

}

void delete\_element(int data)

{

int i;

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

{

printf("\nEmpty Queue");

return;

}

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

{

if (data == pqueue[i])

{

for (; i < rear; i++)

{

pqueue[i] = pqueue[i + 1];

}

pqueue[i] = -99;

rear--;

if (rear == -1)

front = -1;

return;

}

}

printf("\n%d element not found in queue", data);

}

void display\_priorityqueue()

{

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

{

printf("\nEmpty Queue ");

return;

}

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

{

printf(" %d ", pqueue[front]);

}

front = 0;

}

EXPERIMENT – 6  
  
QUES1:Implement a Hash table using arrays (dynamic array) and Linear probing as a Collision avoidance strategy. Perform Insert, Delete and Search operations on the Hash table using the above Hash function   
  
ALGORITHM :  
HashCal function   
1. Begin

2. Pass a number to accept the SAP ID as content

3. Take a temp variable and store the passed esteem

4. In a for circle, take the last the 3 digits of the passed SAP ID and add them

5. On the off chance that the absolute of the last 3 digits is more noteworthy than 10, make one more for circle to add the digits of the total

6. Bring aggregate back

7. STOP

Insert function   
1. Begin

2. Pass the hash table cluster as the boundary

3. Take contribution of the SAP id of the understudy

4. Take a variable and call the HashCal capacity to store the group of the sap id

5. Assuming the position is unfilled, enter information at that position.

6. On the off chance that not, make a for circle to navigate till the following void space, when found enter information

7. Else, if the list is full, do not enter more data   
8. STOP  
 Display function  
 1. START  
 2. Take the hash table array as parameter   
3. Use a for loop and print out the table   
4. STOP   
Delete function   
1. START  
 2. Take the hash table array as parameter  
 3. Take input of the index where data needs to be deleted   
4. Make that element of the hash table as -1   
5. Decrement count   
6. STOP  
 Search function  
 1. START  
 2. Take the hash table array as parameter  
 3. Take input of the sap id which is to be searched  
 4. Call the HashCal function to calculate the index of the element  
 5. If the sap id is equal to the element present at that index, give a confirmation prompt   
6. Else print the element is not in the list   
7. STOP  
 Main function  
1. Begin

2. Take the size of the array

3. Announce a hash table of the entered size

4. Utilize a for circle to make every one of the elements of the array invalid

5. Utilize a do while circle to take the contribution of the activity to be completed and call all the capacity properly

6. STOP

CODE :   
#include <stdio.h>

#include <stdlib.h>

int count = 0, size;

int HashCal(int sap)

{

int temp = 0, val = 0, fam = 0, temp1 = 0;

temp = sap;

for (int i = 0; i < 3; i++) // loop to add the last 3 digits of SAP ID

{

val = temp % 10;

fam = val + fam;

temp = temp / 10;

}

if (fam >= 10) // loop to add the digits if the sum of last 3 digits is greater than or equal to 10

{

temp = fam;

for (int i = 0; i < 2; i++)

{

val = temp % 10;

temp1 = val + temp1;

temp = temp / 10;

}

return temp1;

}

else

return fam;

}

void display(int HashT[])

{

printf("\nIndex\n");

for (int i = 0; i <= size - 1; i++)

{

if (i < 10)

{

if (HashT[i] == -1)

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

else

printf(" %d ---> %d\n", i, HashT[i]);

}

else

{

if (HashT[i] == -1)

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

else

printf(" %d ---> %d\n", i, HashT[i]);

}

}

return;

}

void insert(int HashT[])

{

int sap, pos;

printf("Enter the SAP ID of the student: ");

scanf("%d", &sap);

pos = HashCal(sap);

if (HashT[pos] == -1)

{

HashT[pos] = sap;

}

else

{

for (int i = pos + 1; i <= pos + (size - 1); i++)

{

if (i <= size - 1)

{

if (HashT[i] == -1)

{

HashT[i] = sap;

break;

}

}

else

{

if (HashT[i - size] == -1)

{

HashT[i - size] = sap;

break;

}

}

}

}

printf("Data Enterted !!\n");

return;

}

void delete (int HashT[])

{

int index;

printf("Enter the index where you want to delete: ");

scanf("%d", &index);

HashT[index] = -1;

count--;

}

void search(int HashT[])

{

int ele;

printf("Enter the SAP ID to be searched: ");

scanf("%d", &ele);

int pos = HashCal(ele);

if (HashT[pos] == ele)

{

printf("The element '%d' is fount at index) '%d'.\n", HashT[pos], pos);

}

else

{

for (int i = pos + 1; i <= pos + (size - 1); i++)

{

if (i <= size - 1)

{

if (HashT[i] == ele)

{

printf("The element '%d' is fount at index '%d'.\n", HashT[pos], i);

break;

}

}

else

{

if (HashT[i - size] == ele)

{

printf("The element '%d' is fount at index '%d'.\n", HashT[pos], i);

break;

}

}

}

}

return;

}

int main()

{

int choice;

printf("Enter the size of Hash Table: ");

scanf("%d", &size);

int HashT[size];

for (int i = 0; i <= size - 1; i++)

{

HashT[i] = -1;

}

do

{

printf("\n1.Insert\n2.Delete\n3.Search\n4.Display\n0.Exit\nEnter your Choice: ");

scanf("%d", &choice);

if (choice == 1)

{

if (count < size)

{

insert(HashT);

count++;

}

else

{

printf("Hash table is full ! Cannot insert.\n");

}

}

else if (choice == 2)

{

delete (HashT);

}

else if (choice == 3)

{

search(HashT);

}

else if (choice == 4)

{

display(HashT);

}

} while (choice != 0);

return 0;

}

QUES2: Implement a Hash table using arrays (dynamic array) and Linear probing as a Collision avoidance strategy. Perform Insert, Delete and Search operations on the Hash table using the above Hash function (S.No.1). Compute Load Factor (LF). When LF>50% apply rehashing (with new dynamic array size & suitable hash function)  
  
HashCal function   
1. Begin

2. Pass a number to accept the SAP ID as content

3. Take a temp variable and store the passed esteem

4. In a for circle, take the last the 3 digits of the passed SAP ID and add them

5. On the off chance that the absolute of the last 3 digits is more noteworthy than 10, make one more for circle to add the digits of the total

6. Bring aggregate back

7. STOP

Insert function   
1. Begin

2. Pass the hash table cluster as the boundary

3. Take contribution of the SAP id of the understudy

4. Take a variable and call the HashCal capacity to store the group of the sap id

5. Assuming the position is unfilled, enter information at that position.

6. On the off chance that not, make a for circle to navigate till the following void space, when found enter information

7. Else, if the list is full, do not enter more data   
8. STOP  
 Display function  
 1. START  
 2. Take the hash table array as parameter   
3. Use a for loop and print out the table   
4. STOP   
Delete function   
1. START  
 2. Take the hash table array as parameter  
 3. Take input of the index where data needs to be deleted   
4. Make that element of the hash table as -1   
5. Decrement count   
6. STOP  
 Search function  
 1. START  
 2. Take the hash table array as parameter  
 3. Take input of the sap id which is to be searched  
 4. Call the HashCal function to calculate the index of the element  
 5. If the sap id is equal to the element present at that index, give a confirmation prompt   
6. Else print the element is not in the list   
7. STOP  
 Main function  
1. Begin

2. Take the size of the array

3. Announce a hash table of the entered size

4. Utilize a for circle to make every one of the elements of the array invalid

5. Utilize a do while circle to take the contribution of the activity to be completed and call all the capacity properly

6. STOP

CODE:  
#include <stdio.h>

#include<stdlib.h>

int count=0,size;

int HashCal(int sap)

{

int temp=0, val=0, fam=0,temp1=0;

temp=sap;

for(int i=0;i<3;i++) //loop to add the last 3 digits of SAP ID

{

val = temp % 10;

fam = val + fam;

temp = temp/10;

}

if (fam>=10) //loop to add the digits if the sum of last 3 digits is greater than or equal to 10

{

temp = fam;

for(int i=0; i<2;i++)

{

val = temp%10;

temp1 = val+temp1;

temp = temp/10;

}

return temp1;

}

else

return fam;

}

void display(int HashT[])

{

printf("\nIndex\n");

for(int i=0;i<=size-1;i++)

{

if(i<10)

{

if(HashT[i]==-1) printf(" %d ---> \n",i); else printf(" %d ---> %d\n",i,HashT[i]);

}

else

{

if(HashT[i]==-1) printf(" %d --> \n",i); else printf(" %d ---> %d\n",i,HashT[i]);

}

}

return;

}

void insert(int HashT[])

{

int sap,pos;

printf("Enter the SAP ID of the student: "); scanf("%d",&sap);

pos=HashCal(sap);

if(HashT[pos]==-1)

{

HashT[pos]=sap;

}

else

{

for(int i=pos+1;i<=pos+(size-1);i++)

{

if(i<=size-1)

{

if(HashT[i]==-1)

{

HashT[i]=sap;

break;

}

}

else

{

if(HashT[i-size]==-1)

{

HashT[i-size]=sap;

break;

}

}

}

}

printf("Data Enterted !!\n");

return;

}

void delete(int HashT[])

{

int index;

printf("Enter the index where you want to delete: "); scanf("%d",&index);

HashT[index]=-1;

count--;

}

void search(int HashT[])

{

int ele;

printf("Enter the SAP ID to be searched: "); scanf("%d",&ele);

int pos=HashCal(ele);

if(HashT[pos]==ele)

{

printf("The element '%d' is fount at index) '%d'.\n",HashT[pos],pos);

}

else

{

for(int i=pos+1;i<=pos+(size-1);i++)

{

if(i<=size-1)

{

if(HashT[i]==ele)

{

printf("The element '%d' is fount at index '%d'.\n",HashT[pos],i); break;

}

}

else

{

if(HashT[i-size]==ele)

{

printf("The element '%d' is fount at index '%d'.\n",HashT[pos],i); break;

}

}

}

}

return;

}

int main()

{

int choice;

printf("Enter the size of Hash Table: "); scanf("%d",&size);

int HashT[size];

for(int i=0;i<=size-1;i++)

{

HashT[i]=-1;

}

do{

printf("\n1.Insert\n2.Delete\n3.Search\n4.Display\n0.Exit\nEnter your Choice: "); scanf("%d",&choice);

if(choice==1)

{

if(count<size)

{

insert(HashT);

count++;

}

else

{

printf("Hash table is full ! Cannot insert.\n");

}

}

else if(choice==2)

{

delete(HashT);

}

else if(choice==3)

{

search(HashT);

}

else if (choice==4)

{

display(HashT);

}

}while(choice!=0);

return 0;

}

QUES 3: Implement a Hash table using an array of Linked Lists with a Separate Chaining Collision avoidance strategy. Perform Insert, Delete and Search operations on the hash table using a suitable Hash function.  
  
ALGORITHM:  
Hash\_family function()   
1. START   
2. Pass an integer to take the SAP ID as argument   
3. Take a temp variable and store the passed value  
 4. In a for loop, take the last the 3 digits of the passed SAP ID and add them   
5. If the total of the last 3 digits is greater than 10, make another for loop to add the digits of the sum  
 6. Return sum  
 7. STOP   
Insert() function   
1. START   
2. Take input of the SAP ID in a variable   
3. Dynamically create a new node and store the SAP ID in it   
4. Call the hash\_family function in a variable to get the family of SAP ID   
5. Check if the array family index is empty, if yes, store the sap id in the index of the family   
6. Else, use a temp variable to traverse through the list and add the element in the last  
 7. STOP  
 Display() function  
 1. START   
2. Use a for loop from 0 to size   
3. In the loop, use a while loop to display the nodes and traverse through the list   
4. STOP   
Main() function   
1. START   
2. Use a while loop to give prompt for more operations   
3. In the loop, create a switch menu with appropriate cases to take all the functions and display them for the user   
4. STOP.  
   
CODE :  
#include <stdio.h>#define size 10

struct node

{

int sapid;

struct node \*next;

} \* array[size], \*new = NULL, \*temp = NULL, \*ptr = NULL;

int hash\_family(int s)

{

int temp = 0, val = 0, fam = 0, temp1 = 0;

temp = s;

for (int i = 0; i < 3; i++) // loop to add the last 3 digits of SAP ID

{

val = temp % 10;

fam = val + fam;

temp = temp / 10;

}

if (fam >= 10) // loop to add the digits if the sum of last 3 digits is greater than or equal to 10

{

temp = fam;

for (int i = 0; i < 2; i++)

{

val = temp % 10;

temp1 = val + temp1;

temp = temp / 10;

}

return temp1;

}

else

return fam;

}

void insert()

{

int sap, family;

printf("Enter the SAPID: ");

scanf("%d", &sap);

// creating a new node

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

new->sapid = sap;

new->next = NULL;

family = hash\_family(sap);

// checking if the family is empty

if (array[family] == NULL)

{

array[family] = new;

}

else

{

temp = array[family];

while (temp->next != 0)

{

temp = temp->next;

}

temp->next = new;

}

printf("Node inserted successfully in family %d.\n", family);

}

void display()

{

int i;

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

{

ptr = array[i];

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

while (ptr)

{

printf("-->%d ", ptr->sapid);

ptr = ptr->next;

}

printf("\n");

}

}

int main()

{

int choice;

char ch = 'y';

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

{

printf("=========== MAIN MENU ===========\n");

printf("1.Insert into Hash Table\n2.Display the Hash Table\n3.Exit\nEnter your choice: ");

EXPERIMENT – 7  
QUES 1:. Sort the List of Elements stored in a Linked List using the Bubble sort algorithm.  
  
ALGORITHM:  
We start with an empty left hand [sorted array] and the cards face down on the table [unsorted array].  
 2. Then remove one card [key] at a time from the table [unsorted array], and insert it into the correct position in the left hand [sorted array].   
3. To find the correct position for the card, we compare it with each of the cards already in the hand, from right to left.  
  
CODE :  
#include <stdio.h>

#include <stdlib.h>

struct node {

int data;

struct node\* next;

} node;

struct node\* swap(struct node\* ptr1, struct node\* ptr2)

{

struct node\* tmp = ptr2->next;

ptr2->next = ptr1;

ptr1->next = tmp;

return ptr2;

}

int bubbleSort(struct node\*\* head, int count)

{

struct node\*\* h;

int i, j, swapped;

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

h = head;

swapped = 0;

for (j = 0; j < count - i - 1; j++) {

struct node\* p1 = \*h;

struct node\* p2 = p1->next;

if (p1->data > p2->data) {

\*h = swap(p1, p2);

swapped = 1;

}

h = &(\*h)->next;

}

if (swapped == 0)

break;

}

}

void printList(struct node\* n)

{

while (n != NULL) {

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

n = n->next;

}

printf("\n");

}

void insertAtTheBegin(struct node\*\* start\_ref, int data)

{

struct node\* ptr1

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

ptr1->data = data;

ptr1->next = \*start\_ref;

\*start\_ref = ptr1;

}

int main()

{

int arr[] = { 78, 20, 10, 32, 1, 5 };

int list\_size, i;

struct node\* start = NULL;

list\_size = sizeof(arr) / sizeof(arr[0]);

for (i = 0; i < list\_size; i++)

insertAtTheBegin(&start, arr[i]);

printf("Linked list before sorting\n");

printList(start);

bubbleSort(&start, list\_size);

printf("Linked list after sorting\n");

printList(start);

return 0;

}

QUES 2: Sort the List of Elements stored in an Array using the following algorithms: Insertion Sort, Selection Sort, Merge Sort, Quick Sort.

ALGORITHM:  
We start with an empty left hand [sorted array] and the cards face down on the table [unsorted array].  
 2. Then remove one card [key] at a time from the table [unsorted array], and insert it into the correct position in the left hand [sorted array].   
3. To find the correct position for the card, we compare it with each of the cards already in the hand, from right to left.  
  
CODE :  
#include <stdio.h>

#include <conio.h>

void main()

{

int a[10], i, j, k, n;

clrscr();

printf("How many elements you want to sort?\n");

scanf("%d", &n);

printf("\nEnter the Elements into an array:\n");

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

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

for (i = 1; i < n; i++)

{

k = a[i];

for (j = i - 1; j >= 0 && k < a[j]; j--)

a[j + 1] = a[j];

a[j + 1] = k;

}

printf("\n\n Elements after sorting: \n");

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

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

}

FOR SELECTION SORT:  
ALGORITHM:  
Step 1 : Repeat For K = 0 to N – 2 Begin   
Step 2 : Set POS = K   
Step 3 : Repeat for J = K + 1 to N – 1 Begin If A[ J ] < A [ POS ] Set POS = J End Step 5 : Swap A [ K ] with A [ POS ] End  
 Step 6 : Exit   
  
  
  
CODE :  
#include <stdio.h>

void main()

{

int i, j, t, n, min, a[10];

clrscr();

printf("\n How many elements you want to sort? ");

scanf("%d", &n);

printf("Enter elements for an array:");

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

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

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

{

min = i;

for (j = i + 1; j < n; j++)

if (a[j] > a[min])

{

min = j;

}

t = a[i];

a[i] = a[min];

a[min] = t;

}

printf("\nAfter sorting the elements are:");

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

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

}

FOR QUICK SORT :  
ALGORITHMS :  
varlist less, pivotList,   
greater if length(q) ≤ 1 return q   
select a pivot value pivot from q   
for each x in q except the pivot element   
if x < pivot then add x to less  
 if x ≥ pivot then add x to greater add pivot to pivotList  
  
  
CODE:  
#include <stdio.h>

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

void main()

{

int low, high, pivot, t, n, i, j, a[10];

clrscr();

printf("\nHow many elements you want to sort ? ");

scanf("%d", &n);

printf("Enter elements for an array:");

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

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

low = 0;

high = n - 1;

quicksort(a, low, high);

printf("After Sorting the elements are:");

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

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

getch();

}

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

{

int pivot, t, i, j;

if (low < high)

{

pivot = a[low];

i = low + 1;

j = high;

while (1)

{

while (pivot > a[i] && i <= high)

i++;

while (pivot < a[j] && j >= low)

j--;

if (i < j)

{

t = a[i];

a[i] = a[j];

a[j] = t;

}

else

break;

}

a[low] = a[j];

a[j] = pivot;

quicksort(a, low, j - 1);

quicksort(a, j + 1, high);

}

}

FOR MERGE ELEMENTS :  
ALGORITHM :  
1. Divide Step If a given array A has zero or one element, simply return;   
it is already sorted. Otherwise, split A[p .. r] into two sub-arrays A[p .. q] and A[q + 1 .. r], each containing about half of the elements of A[p .. r].   
That is, q is the halfway point of A[p .. r]. 2. Conquer Step Conquer by recursively sorting the two sub-arrays A[p .. q] and A[q + 1 .. r].   
3. Combine Step Combine the elements back in A[p .. r] by merging the two sorted sub-arrays A[p .. q] and A[q + 1 .. r] into a sorted sequence.   
To accomplish this step, we will define a procedure MERGE (A, p, q, r).  
  
CODE :#include <stdio.h>

void disp();

void mergesort(int, int, int);

void msortdiv(int, int);

int a[50], n;

void main()

{

int i;

clrscr();

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

scanf("%d", &n);

printf("\nEnter elements for an array:");

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

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

printf("\nBefore Sorting the elements are:");

disp();

msortdiv(0, n - 1);

printf("\nAfter Sorting the elements are:");

disp();

getch();

}

void disp()

{

int i;

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

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

}

void mergesort(int low, int mid, int high)

{

int t[50], i, j, k;

i = low;

j = mid + 1;

k = low;

while ((i <= mid) && (j <= high))

{

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

t[k++] = a[j++];

else

t[k++] = a[i++];

}

while (i <= mid)

t[k++] = a[i++];

while (j <= high)

t[k++] = a[j++];

for (i = low; i <= high; i++)

a[i] = t[i];

}

void msortdiv(int low, int high)

{

int mid;

if (low != high)

{

mid = ((low + high) / 2);

msortdiv(low, mid);

msortdiv(mid + 1, high);

mergesort(low, mid, high);

}

}

EXPERIMENT – 8  
QUES 1: 1. Store ‘n’ numbers (integers or real) in an array in an unsorted order. Using menu-driven logic perform a search (Linear and Binary Search algorithms) for a given number and report success or failure in the form of a suitable message. (Hint: Before performing a Binary search the array should be sorted using the Bubble sort algorithm.)  
  
CODE:   
nclude <stdio.h>

#include <stdlib.h>

int \* my\_arr = NULL;

int count = 0;

int size = 0;

int main()

{

printf("Enter Size of Array : ");

scanf("%d", &size);

my\_arr= (int \*)malloc(size\*sizeof(int));

while(1){

int choice = 0;

printf("1 - Append To Array\n2 - Linear Search\n3 - Binary Search\n4 - Print Array\n5 - Quit\nChoice : ");

scanf("%d",&choice);

switch(choice){

case 1 : append\_to\_array();break;

case 2 : linear\_search();break;

case 3 : binary\_search();break;

case 4 : print\_array();break;

case 5 : exit(0);

}

printf("\n");

}

return 0;

}

void append\_to\_array(){

if (count == size){

printf("Array is Full..\n");

return;

}

int data = 0;

printf("Enter Number to Append : ");

scanf("%d",&data);

my\_arr[count] = data;

count++;

}

void linear\_search(){

int element = 0;

printf("Enter Element To Find : ");

scanf("%d",&element);

int flag = 0;

for (int i = 0; i < count; i++){

if (my\_arr[i] == element){

printf("%d is at Index %d",element,i);

flag = 1;

break;

}

}

if (flag == 0){

printf("Element not in Array... Exiting...\n");

}

}

void binary\_search(){

bubble\_sort();

int element;

printf("Enter Element To Find : ");

scanf("%d",&element);

int low = 0;

int high = count - 1;

if (my\_arr[high] == element){

printf("Element %d is at Index %d",element, high);

return;

}

if (my\_arr[low] == element){

printf("Element %d is at Index %d",element, low);

return;

}

int mid = (high + low)/2;

while (my\_array[mid] != element){

if (low == high - 1){

if (my\_array[high] == element){

printf("Element %d is at Index %d",element, high);

return;

}

if (my\_array[low] == element){

printf("Element %d is at Index %d",element, low);

return;

}

printf("Element NOT in Array...\n");

return;

}

if (element > my\_array[mid]){

low = mid;

mid = (high + low)/2;

continue;

}

if (element < my\_array[mid]){

high = mid;

mid = (high + low)/2;

continue;

}

}

printf("Element %d is at Index %d",element, mid);

}

void bubble\_sort(){

int flag = 1;

while(flag == 1){

flag = 0;

for (int i = 0 ; i+1 < count ;i++){

if (my\_array[i] > my\_array[i+1]){

flag = 1;

swap(i,i+1);

}

}

}

}

void print\_array(){

printf("\n");

for (int i = 0; i < count; i++){

printf("| %d |",my\_array[i]);

}

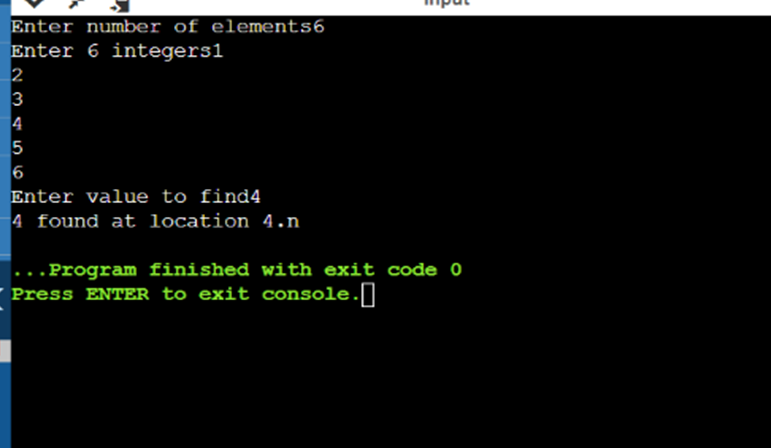
printf("\n");

}

void swap(int i, int j){

int temp = my\_array[i];

my\_array[i] = my\_array[j];  
 my\_array[j] = temp

**OUTPUT :**  
  
QUES 2: Store ‘n’ numbers (integers or real) in a Linked list. Perform a Linear search for a given number and report success or failure in the form of a suitable message.  
  
CODE:  
#include<stdio.h>

#include<stdlib.h>

void create(int);

void search();

struct node

{

int data;

struct node \*next;

};

struct node \*head;

void main ()

{

int choice,item,loc;

do

{

printf("\n1.Create\n2.Search\n3.Exit\n4.Enter your choice?");

scanf("%d",&choice);

switch(choice)

{

case 1:

printf("\nEnter the item\n");

scanf("%d",&item);

create(item);

break;

case 2:

search();

case 3:

exit(0);

break;

default:

printf("\nPlease enter valid choice\n");

}

}while(choice != 3);

}

void create(int item)

{

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

if(ptr == NULL)

{

printf("\nOVERFLOW\n");

}

else

{

ptr->data = item;

ptr->next = head;

head = ptr;

printf("\nNode inserted\n");

}

}

void search()

{

struct node \*ptr;

int item,i=0,flag;

ptr = head;

if(ptr == NULL)

{

printf("\nEmpty List\n");

}

else

{

printf("\nEnter item which you want to search?\n");

scanf("%d",&item);

while (ptr!=NULL)

{

if(ptr->data == item)

{

printf("item found at location %d ",i+1);

flag=0;

}

else

{

flag=1;

}

i++;

ptr = ptr -> next;

}

if(flag==1)

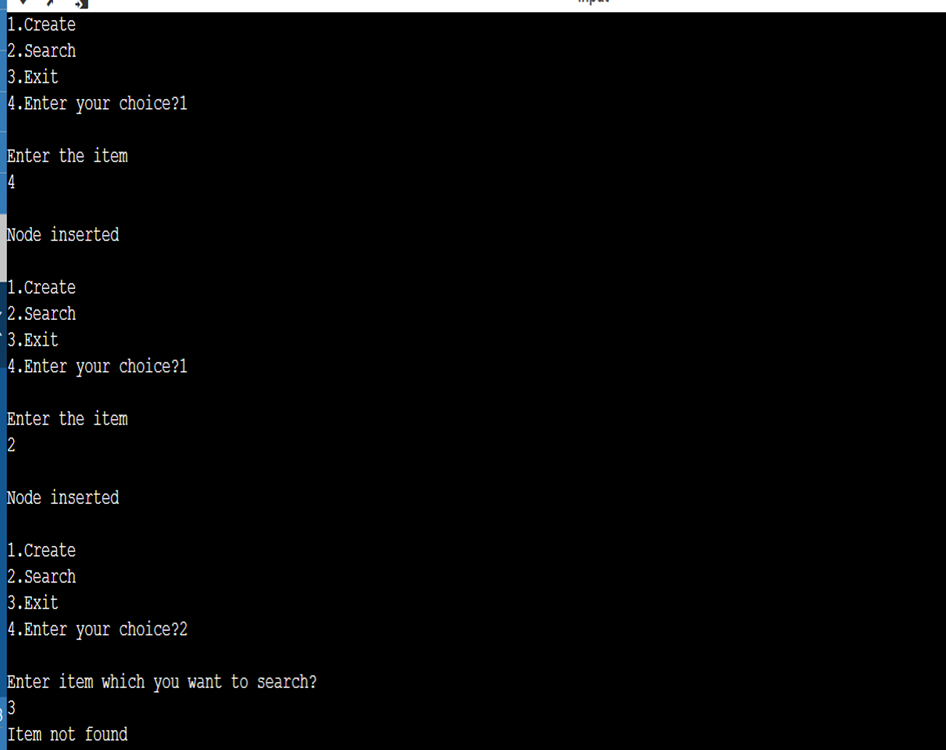
{

printf("Item not found\n");

}

}

}

OUTPUT : 

QUES 3: Store ‘n’ numbers (integers or real) in a Hash Table implemented with Array of Linked Lists. Perform a search for a given number and report success or failure in the form of a suitable message.  
  
CODE :  
#include <stdio.h>

#include <stdlib.h>

int \* hash\_array;

int count = 0;

int size;

int hash\_function(int hash\_key);

int add\_till\_death(int input);

int main()

{

printf("\t\t\t\t\tDSA Experiment - 8.3");

size = 10;

hash\_array = (int \*)malloc(size\*sizeof(int));

for(int i = 0;i < size;i++){

hash\_array[i] = NULL;

}

int choice = 0;

while (1 == 1){

printf("\n1 - To Enter Sap ID\n2 - To Delete SAP\_ID\n3 - Find Hash Key\n4 - Quit\nChoice : ");

scanf("%d",&choice);

switch(choice){

case 1 : insert\_key\_to\_table();break;

case 2 : delete\_key();break;

case 3 : search\_key();break;

case 4 : exit(0);

}

printf("\n");

}

return 0;

}

int hash\_function(int hash\_key){ //RETURN THE ARRAY INDEX FOR THE GIVEN KEY - IN THIS CASE SAP ID

hash\_key = hash\_key - 5000000;

return add\_till\_death(hash\_key);

}

void insert\_key\_to\_table(){ //FRONT END PRETTY FUNCTION

int data = 0;

printf("Enter SAP\_ID To insert : ");

scanf("%d",&data);

insert\_to\_array\_hash\_table(data,hash\_function(data));

}

int insert\_to\_array\_hash\_table(int data,int key){ //TECHNICAL FUNCTION BACKEND

if (hash\_array[key] == NULL){

hash\_array[key] = data;

return;

}

//LINEAR PROBING

int i = 0;

for(i = (key + 1) % size; hash\_array[i]!=NULL && i!=key ; i++,i = i % size); //KEEP GOING TO INDEX THAT IS NULL EVEN AFTER END OF ARRAY

if (i == key){

printf("\nNo Available Space Array is FULL...");

return;

}

hash\_array[i] = data;

}

int add\_till\_death(int input){

int total = 0;

while (input != 0){

total = total + input % 10;

input = input/10;

}

if (total > 9)

total = add\_till\_death(total);

return total;

}

void search\_key(){ //SEARCHING IN ARRAY OF LINKED LIST

int data;

int key\_index;

printf("Enter Entry to Find : ");

scanf("%d", &data);

key\_index = hash\_function(data);

if (hash\_array[key\_index] == data){ //TO MAKE THE FOR LOOP CONDITION USEABLE

printf("\n%d is Present at Index : %d",data,key\_index);

return;

}

int i = key\_index+1;

for (i ; hash\_array[i] != data && i!=key\_index ; i++,i = i % size);

if (i == key\_index){

printf("\nKey not in Table");

return;

}

printf("\n%d is Present at Index : %d",data,i);

}

void delete\_key(){

int data;

printf("\nEnter Key to Delete : ");

scanf("%d",&data);

int key\_index = hash\_function(data);

int i = key\_index + 1;

if (hash\_array[key\_index] == data){

printf("\nDeleted Key : %d",hash\_array[key\_index]);

hash\_array[key\_index] = NULL;

return;

}

for (i ; hash\_array[i] != data && i!= key\_index; i++,i = i % size);

if (i == key\_index){

printf("\nKey not in Table\n");

return;

}

printf("\nDeleted Key : %d",hash\_array[i]);

hash\_array[i] = NULL;

}

EXPERIMENT – 9  
  
QUES 1:2. Construct a binary search tree (BST) using a linked list. 3. Perform Inorder, Preorder, and Postorder Traversal on the BST tree constructed.  
  
**Algorithm:**Algorithm Inorder(tree)   
1. Traverse the left subtree, i.e., call Inorder(left-subtree)   
2. Visit the root.   
3. Traverse the right subtree, i.e., call Inorder(right-subtree) Algorithm Preorder(tree)  
 1. Visit the root.  
 2. Traverse the left subtree, i.e., call Preorder(left-subtree)   
3. Traverse the right subtree, i.e., call Preorder(right-subtree) Algorithm Postorder(tree)  
 1. Traverse the left subtree, i.e., call Postorder(left-subtree)   
2. Traverse the right subtree, i.e., call Postorder(right-subtree)  
 3. Visit the root.   
  
CODE :  
#include <stdio.h>

#include <stdlib.h>

struct node

{

int data;

struct node \*left;

struct node \*right;

};

struct node \*newNode(int data)

{

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

node->data = data;

node->left = NULL;

node->right = NULL;

return (node);

}

void printPostorder(struct node \*node)

{

if (node == NULL)

return;

// first recur on left subtree

printPostorder(node->left);

// then recur on right subtree

printPostorder(node->right);

// now deal with the node

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

}

void printInorder(struct node \*node)

{

if (node == NULL)

return;

/\* first recur on left child \*/

printInorder(node->left);

/\* then print the data of node \*/

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

/\* now recur on right child \*/

printInorder(node->right);

}

void printPreorder(struct node \*node)

{

if (node == NULL)

return;

/\* first print data of node \*/

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

/\* then recur on left sutree \*/

printPreorder(node->left);

/\* now recur on right subtree \*/

printPreorder(node->right);

}

int main()

{

struct node \*root = newNode(1);

root->left = newNode(2);

root->right = newNode(3);

root->left->left = newNode(4);

root->left->right = newNode(5);

printf("\nPreorder traversal of binary tree is \n");

printPreorder(root);

printf("\nInorder traversal of binary tree is \n");

printInorder(root);

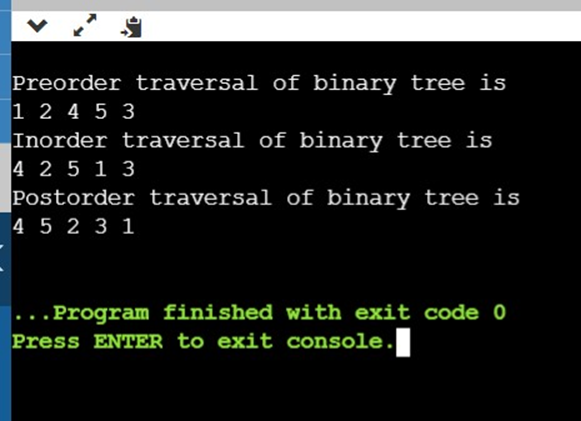
printf("\nPostorder traversal of binary tree is \n");

printPostorder(root);

getchar();

return 0;

}

OUTPUT :  
  
  
  
QUES 2:Given the breadth-first traversal, realize a complete binary tree (CBT) with an array. Implement the following:   
a. Given the index of a node, identify the node’s children and parent. b. Identify and print the available parent-left child-right child node combinations available in the CBT.   
c. Identify and print the leaf nodes available in the CBT.  
  
**Algorithm:**1.Define Node class which has three attributes namely: data left and right. Here, left represents the left child of the node and right represents the right child of the node.   
2.When a node is created, data will pass to data attribute of the node and both left and right will be set to null.  
 3.Define another class which has an attribute root.  
 a.Root represents the root node of the tree and initialize it to null. 4.insert() will add a new node to the tree:   
a.It checks whether the root is null, which means the tree is empty. It will add the new node as root. a.Else, it will add root to the queue. b.The variable node represents the current node.  
 c.First, it checks whether a node has a left and right child. If yes, it will add both nodes to queue.  
 d.If the left child is not present, it will add the new node as the left child.  
 e.If the left is present, then it will add the new node as the right child.  
 5.Inorder() will display nodes of the tree in inorder fashion.   
a.It traverses the entire tree then prints out left child followed by root then followed by the right child.  
  
**CODE :**#include <stdio.h>

#include <stdlib.h>

#include <stdbool.h>

struct node

{

int data;

struct node \*left;

struct node \*right;

};

struct node \*root = NULL;

struct node \*createNode(int data)

{

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

newNode->data = data;

newNode->left = NULL;

newNode->right = NULL;

return newNode;

}

struct queue

{

int front, rear, size;

struct node \*\*arr;

};

struct queue \*createQueue()

{

struct queue \*newQueue = (struct queue \*)malloc(sizeof(struct queue));

newQueue->front = -1;

newQueue->rear = 0;

newQueue->size = 0;

newQueue->arr = (struct node \*\*)malloc(100 \* sizeof(struct node \*));

return newQueue;

}

void enqueue(struct queue \*queue, struct node \*temp)

{

queue->arr[queue->rear++] = temp;

queue->size++;

}

struct node \*dequeue(struct queue \*queue)

{

queue->size--;

return queue->arr[++queue->front];

}

void insertNode(int data)

{

struct node \*newNode = createNode(data);

if (root == NULL)

{

root = newNode;

return;

}

else

{

struct queue \*queue = createQueue();

enqueue(queue, root);

while (true)

{

struct node \*node = dequeue(queue);

if (node->left != NULL && node->right != NULL)

{

enqueue(queue, node->left);

enqueue(queue, node->right);

}

else

{

if (node->left == NULL)

{

node->left = newNode;

enqueue(queue, node->left);

}

else

{

node->right = newNode;

enqueue(queue, node->right);

}

break;

}

}

}

}

void inorderTraversal(struct node \*node)

{

if (root == NULL)

{

printf("Tree is empty\n");

return;

}

else

{

if (node->left != NULL)

inorderTraversal(node->left);

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

if (node->right != NULL)

inorderTraversal(node->right);

}

}

int main()

{

insertNode(1);

printf("Binary tree after insertion: \n");

inorderTraversal(root);

insertNode(2);

insertNode(3);

printf("\nBinary tree after insertion: \n");

inorderTraversal(root);

printf("\nBinary tree after insertion: \n");

inorderTraversal(root);

insertNode(4);

insertNode(5);

insertNode(6);

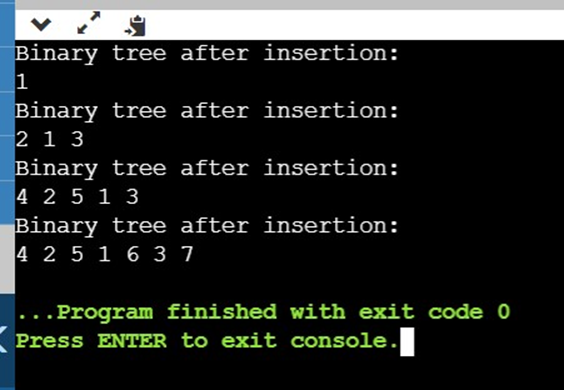
insertNode(7);

printf("\nBinary tree after insertion: \n");

inorderTraversal(root);

return 0;

}

OUTPUT :  
  
  
 EXPERIMENT – 10  
  
QUES 1: 1. Get the number of vertices of a graph and adjacency matrix. Check if the graph is a digraph or an undirected graph. [Hint: The matrix for an undirected graph is symmetric.  
  
Algorithm:  
 1. Create the graph using the given number of edges and vertices.   
2. Create a recursive function that that current index or vertex, visited and recursion stack.   
3. Mark the current node as visited and also mark the index in recursion stack.  
 4. Find all the vertices which are not visited and are adjacent to the current node. Recursively call the function for those vertices, If the recursive function returns true return true.   
5. If the adjacent vertices are already marked in the recursion stack then return true.   
6. Create a wrapper class, that calls the recursive function for all the vertices and if any function returns true, return true.   
7. Else if for all vertices the function returns false return false.  
  
CODE :  
include <stdio.h>

#include <stdlib.h>

#define MAX 100

#define initial 1

#define waiting 2

#define visited 3

int n;

int adj[MAX][MAX]; // Adjacency Matrix

int state[MAX]; // can be initial, waiting or visited

void create\_graph();

void BF\_Traversal();

void BFS(int v);

int queue[MAX], front = -1, rear = -1;

void insert\_queue(int vertex);

int delete\_queue();

int isEmpty\_queue();

int main()

{

create\_graph();

BF\_Traversal();

return 0;

}

void BF\_Traversal()

{

int v;

int connected = 1;

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

state[v] = initial;

BFS(0);

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

{

if (state[v] == initial)

{

connected = 0;

break;

}

}

if (connected)

printf("\nGraph is connected\n");

else

printf("\nGraph is not connected\n");

} /\*End of BF\_Traversal()\*/

void BFS(int v)

{

int i;

insert\_queue(v);

state[v] = waiting;

while (!isEmpty\_queue())

{

v = delete\_queue();

state[v] = visited;

printf("%d ", v);

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

{

/\* Check for adjacent unvisited vertices \*/

if (adj[v][i] == 1 && state[i] == initial)

{

insert\_queue(i);

state[i] = waiting;

}

}

}

printf("\n");

}

void insert\_queue(int vertex)

{

if (rear == MAX - 1)

printf("\nQueue Overflow\n");

else

{

if (front == -1) /\*If queue is initially empty \*/

front = 0;

rear = rear + 1;

queue[rear] = vertex;

}

} /\*End of insert\_queue()\*/

int isEmpty\_queue()

{

if (front == -1 || front > rear)

return 1;

else

return 0;

} /\*End of isEmpty\_queue()\*/

int delete\_queue()

{

int del\_item;

if (front == -1 || front > rear)

{

printf("\nQueue Underflow\n");

exit(1);

}

del\_item = queue[front];

front = front + 1;

return del\_item;

} /\*End of delete\_queue() \*/

void create\_graph()

{

int i, max\_edges, origin, destin;

printf("\nEnter number of vertices : ");

scanf("%d", &n);

max\_edges = n \* (n - 1) / 2;

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

{

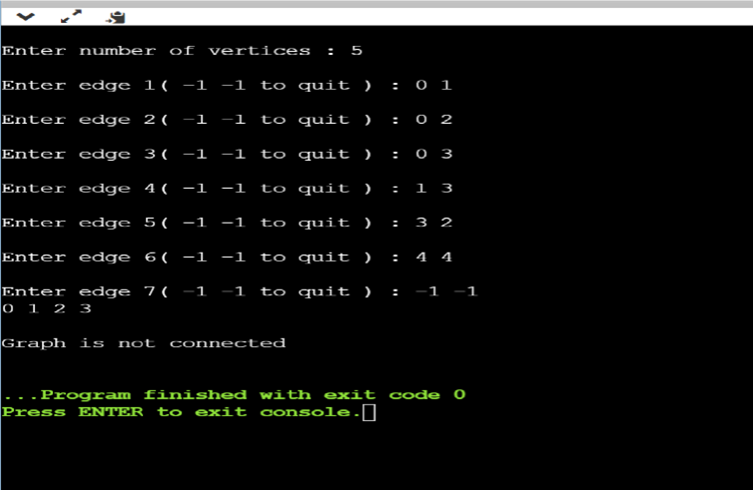
adj[origin][destin] = 1;

adj[destin][origin] = 1;

}

}

}

OUTPUT :  
  
  
QUES 2: Write a menu-driven program to obtain vertex IDs (characters ‘A’, ‘B’, ‘C’, etc.) in a character array and adjacency matrix for a digraph and an undirected graph. Implement separate functions to display: (a) Degree of all vertices of the undirected graph, (b) In-degree and out-degree of all vertices of the digraph, (c) Adjacency matrices.   
  
CODE :  
#include <stdio.h>

#include <stdlib.h>

// A structure to represent an adjacency list node

struct AdjListNode

{

int dest;

struct AdjListNode \*next;

};

// A structure to represent an adjacency list

struct AdjList

{

struct AdjListNode \*head;

};

struct Graph

{

int V;

struct AdjList \*array;

};

// A utility function to create a new adjacency list node

struct AdjListNode \*newAdjListNode(int dest)

{

struct AdjListNode \*newNode =

(struct AdjListNode \*)malloc(sizeof(struct AdjListNode));

newNode->dest = dest;

newNode->next = NULL;

return newNode;

}

// A utility function that creates a graph of V vertices

struct Graph \*createGraph(int V)

{

struct Graph \*graph =

(struct Graph \*)malloc(sizeof(struct Graph));

graph->V = V;

// Create an array of adjacency lists. Size of

// array will be V

graph->array =

(struct AdjList \*)malloc(V \* sizeof(struct AdjList));

// Initialize each adjacency list as empty by

// making head as NULL

int i;

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

graph->array[i].head = NULL;

return graph;

}

// Adds an edge to an undirected graph

void addEdge(struct Graph \*graph, int src, int dest)

{

// Add an edge from src to dest. A new node is

// added to the adjacency list of src. The node

// is added at the beginning

struct AdjListNode \*newNode = newAdjListNode(dest);

newNode->next = graph->array[src].head;

graph->array[src].head = newNode;

// Since graph is undirected, add an edge from

// dest to src also

newNode = newAdjListNode(src);

newNode->next = graph->array[dest].head;

graph->array[dest].head = newNode;

}

// A utility function to print the adjacency list

// representation of graph

void printGraph(struct Graph \*graph)

{

int v;

for (v = 0; v < graph->V; ++v)

{

struct AdjListNode \*pCrawl = graph->array[v].head;

printf("\n Adjacency list of vertex %d\n head ", v);

while (pCrawl)

{

printf("-> %d", pCrawl->dest);

pCrawl = pCrawl->next;

}

printf("\n");

}

}

// Driver program to test above functions

int main()

{

// create the graph given in above fugure

int V = 5;

struct Graph \*graph = createGraph(V);

addEdge(graph, 0, 1);

addEdge(graph, 0, 4);

addEdge(graph, 1, 2);

addEdge(graph, 1, 3);

addEdge(graph, 1, 4);

addEdge(graph, 2, 3);

addEdge(graph, 3, 4);

// print the adjacency list representation of the above graph

printGraph(graph);

return 0;

}

OUTPUT:  
