**DEPARTMENT OF COMPUTER SCIENCE**

**RAJAGIRI COLLEGE OF SOCIAL SCIENCES**

**(Autonomous)**

**KALAMASSERY - KOCHI - 683104**

****

MASTER OF COMPUTER APPLICATIONS

**DATA STRUCTURES LAB RECORD**

**NAME : \_\_\_ALBIN JOSEPH\_\_\_\_\_\_\_\_\_**

**SEMESTER : \_\_\_1ST SEMESTER\_\_\_\_\_\_\_\_\_\_**

**REGISTER NO. : \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

****

**DEPARTMENT OF COMPUTER SCIENCE**

**RAJAGIRI COLLEGE OF SOCIAL SCIENCES**

**(Autonomous)**

**KALAMASSERY - KOCHI – 683104**

**MASTER OF COMPUTER APPLICATIONS**

CERTIFICATE

**NAME : \_\_\_ALBIN JOSEPH\_\_\_\_\_\_\_**

**SEMESTER : \_\_\_1ST SEMESTER\_\_\_\_\_\_\_\_**

**REGISTER NO. : \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

*Certified that this is a bonafide record of work done by the student in the Software Laboratory of Rajagiri Department of Computer Science, Kalamassery.*

Faculty in Charge Dean, Computer Science

Internal Examiner External Examiner

Place : Kalamassery

Date :

**Table of Contents**  **Page**

|  |  |  |
| --- | --- | --- |
|  | Write programs to demonstrate the use of storage classes in C. | **1** |
|  | Use a menu-driven program to insert, search, delete and sort elements in an array using functions (use global variables) | **2** |
|  | Use a menu-driven program to insert, search, delete and sort elements in an array using functions (use only local variables) | **5** |
|  | Search for all the occurrences of an element in an integer array (positions) | **9** |
|  | Sort the array elements in ascending order (minimum three functions: read, disp and sort) | **10** |
|  | Two-dimensional matrix: using functions   1. Addition 2. Subtraction 3. Multiplication 4. Transpose 5. Determinant | **13** |
|  | Display the array elements in the same order using a recursive function | **19** |
|  | Display array elements in reverse order using a recursive function | **20** |
|  | Implement stack operations using arrays. | **21** |
|  | Reverse a string using Stack | **22** |
|  | Convert an expression from infix to postfix using stack | **23** |
|  | Evaluate an expression using stack | **27** |
|  | Define a structure for dates with dd/mm/yyyy. Provide functions for reading, displaying and comparing two dates are equal or not | **32** |
|  | Define a structure for employees with eno,ename, esal and dno. Read n employees information and provide functions for the following:   1. Searching an employee by no 2. Sorting the employees by   Name  Salary   1. Deleting an employee | **35** |
|  | Read a polynomial and display it; use array | **42** |
|  | Add two polynomials using the array itself | **43** |
|  | Read a polynomial and display it; use structure array | **47** |
|  | Add two polynomials | **48** |
|  | Subtract two polynomials | **51** |
|  | Multiply two polynomials | **54** |
|  | Implement a) malloc , b) calloc and c) free functions | **56** |
|  | Use malloc to read n integers and find the mean. | **58** |
|  | Use calloc to read n numbers and find the mode. | **59** |
|  | Declare a structure for Books having author\_name and book\_name. Create an array of books using a pointer variable. Provide functions for reading n books and displaying the same using pointers. | **61** |
|  | Use realloc to implement varchar for any length. | **63** |
|  | Implement Queue using array | **65** |
|  | Implement priority queue | **67** |
|  | Demonstrate a linked list creation and display | **72** |
|  | Write a program with functions  to insert a new node  at the beginning of a Singly Linked List.  At the end of the linked list  after a specified element in a linked list. | **74** |
|  | Write a program with functions to delete a nodeFrom the beginning of the linked list  From the end of the linked list  The node with specified data element | **78** |
|  | Write a program to create a singly linked list of n nodes and display it in reverse order. | **82** |
|  | Sort the elements in a linked list using  changing the values (swapping the values)  Changing the address (Swapping the address) | **86** |
|  | Polynomial using linked list - addition and multiplication | **90** |
|  | Linked list using names - insert, delete, display, sort, reverse, count | **95** |
|  | Linked Stack | **100** |
|  | Linked Queue | **101** |
|  | Circular Linked List | **104** |
|  | Circular Linked Queue | **106** |
|  | Doubly Linked List | **109** |
|  | Circular doubly linked list - store string values as data part | **112** |
|  | Binary search tree insertion and display Traversal using inorder, preorder and postorder using recursion. | **115** |
|  | Binary search tree insertion and display in-order without using recursion | **119** |
|  | Binary search tree insertion and display pre-order without using recursion | **122** |
|  | Binary search tree insertion and display post-order without using recursion | **125** |
|  | Binary search tree insertion using names and display the names in ascending order using inorder traversal. | **129** |
|  | Demonstrate the data structure of adjacent matrix  using arrays | **132** |
|  | Demonstrate the data structure of adjacent matrix  using linked lists | **133** |

|  |
| --- |
| **Program 1** |
| **Write programs to demonstrate the usage of storage classes in C.** |

**Source Code:**

#include <stdio.h>

int a = 5, b; // Global variables

void print() {

    printf("The value of global variable a is %d (assigned)\n", a);

    printf("The value of global variable b is %d (default value is undefined)\n", b);

}

void display() {

    static int i; // Static variable

    printf("Value of static variable i is %d (default initialized value)\n", i);

    static int k = 1;

    printf("Assigned value of static variable k = %d\n", k);

    k++;

}

void reg() {

    int h = 3; // Local variable

    printf("The value of h = %d (local variable)\n", h);

}

int main() {

    int c; // Local variable

    printf("The value of local variable c is %d (garbage value)\n", c);

    print();

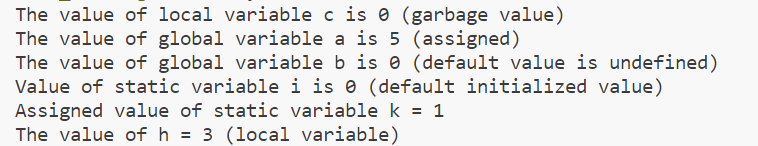
    display();

    reg();

    return 0;

}

**Output:**

****

|  |
| --- |
| **Program 2** |
| **Use a menu-driven program to insert, search, delete and sort elements in an array using functions (use global variables)** |

**Source Code:**

#include <stdio.h>

int ar[10];

int n; // Global variable to track the number of elements in the array

int x; // Global variable to store the search element

void insert(int n) {

    int i;

    for (i = 0; i < n; i++) {

        scanf("%d", &ar[i]);

    }

    printf("\nElements are inserted\n");

}

void display() {

    int i;

    if (n > 0) {

        printf("Array elements are: ");

        for (i = 0; i < n; i++) {

            printf("%d", ar[i]);

            if (i < n - 1)

                printf(", ");

        }

        printf("\n");

    } else {

        printf("Array is empty\n");

    }

}

void delete() {

    if (n == 0) {

        printf("Array is empty\n");

    } else {

        printf("Last element of the array is deleted\n");

        n--;

    }

}

void sort() {

    int i, j;

    for (i = 0; i < n - 1; i++) {

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

            if (ar[j] > ar[j + 1]) {

                int temp = ar[j];

                ar[j] = ar[j + 1];

                ar[j + 1] = temp;

            }

        }

    }

    printf("Array sorted successfully.\n");

}

void search() {

    int i, flag = 0;

    for (i = 0; i < n; i++) {

        if (ar[i] == x) {

            flag = 1;

            printf("Element %d found at index %d.\n", x, i);

            break;

        }

    }

    if (flag == 0) {

        printf("Element %d not found in the array.\n", x);

    }

}

int menu() {

    int ch;

    printf("INSERT-1\nDELETE-2\nDISPLAY-3\nSORT-4\nSEARCH-5\nEXIT-6\nENTER YOUR CHOICE: ");

    scanf("%d", &ch);

    return ch;

}

void process() {

    int ch;

    for (ch = menu(); ch != 6; ch = menu()) {

        switch (ch) {

            case 1:

                printf("Enter the value of n: ");

                scanf("%d", &n);

                if (n <= 0) {

                    printf("Invalid value of n\n");

                    break;

                }

                printf("Enter the elements: ");

                insert(n);

                break;

            case 2:

                delete();

                break;

            case 3:

                display();

                break;

            case 4:

                sort();

                break;

            case 5:

                printf("Enter element to search: ");

                scanf("%d", &x);

                search();

                break;

            default:

                printf("Wrong choice\n");

                break;

        }

    }

}

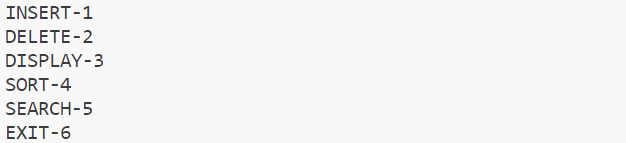
int main() {

    process();

    return 0;

}

**Output:**





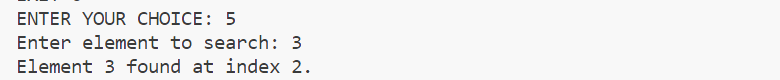












|  |
| --- |
| **Program 3** |
| **Use a menu driven program to insert, search, delete and sort elements in an array using functions (use only local variables).** |

**Source Code:**

#include<stdio.h>

void insert(int ar[], int n)

{

    int i;

    for(i=0;i<n;i++)

    {

        scanf("%d",&ar[i]);

    }

}

void display(int ar[], int n)

{

    int i;

    if(n>0){

        printf("Array elements are: ");

        for(i=0;i<n;i++)

        {

            printf("%d",ar[i]);

            if(i<n-1) printf(", ");

        }

        printf("\n");

    }

    else

    {

        printf("Array is empty\n");

    }

}

int delete(int ar[], int n)

{

    if(n==-1){

        printf("array is empty");

    }

    else{

        printf("last element of array is deleted");

        n=n-1;

    }

    return n;

}

void sort(int ar[], int n)

{

    int i, j;

    for (i = 0; i < n - 1; i++) {

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

            if (ar[j] > ar[j + 1]) {

                int temp = ar[j];

                ar[j] = ar[j + 1];

                ar[j + 1] = temp;

            }

        }

    }

    printf("Array sorted successfully.\n");

    return;

}

void search(int ar[], int n, int x)

{

    int i, flag = 0;

    for (i = 0; i < n; i++) {

        if (ar[i] == x) {

            flag = 1;

            printf("Element %d found at index %d.\n", x, i);

            break;

        }

    }

    if (flag == 0) {

        printf("Element %d not found in the array.\n", x);

    }

    return;

}

int menu()

{

    int ch;

    printf("\nINSERT-1\nDELETE-2\nDISPLAY-3\nSORT-4\nSEARCH-5\nEXIT-6\nENTER YOUR CHOICE: ");

    scanf("%d",&ch);

    return ch;

}

void process()

{

    int ch;

    int x;

    int ar[10];

    int n = -1;

    for(ch=menu();ch!=6;ch=menu())

    {

        switch(ch)

        {

            case 1:

                printf("Enter the value of n: ");

                scanf("%d",&n);

                if(n<=0){

                    printf("Invalid value of n\n");

                    break;

                }

                printf("Enter the elements: ");

                insert(ar,n);

                break;

            case 2:

               n = delete(ar,n);

               break;

            case 3:

                display(ar,n);

                break;

            case 4:

               sort(ar,n);

               break;

            case 5:

                printf("enter element to search");

                scanf("%d",&x);

                search(ar,n,x);

                break;

            default:

               printf("Wrong choice\n");

               break;

        }

    }

}

int main()

{

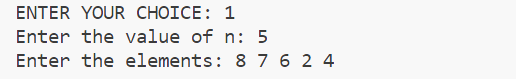
   process();

   return 0;

}

**Output:**

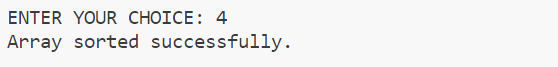


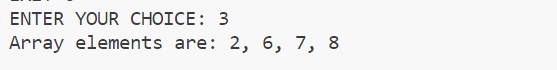


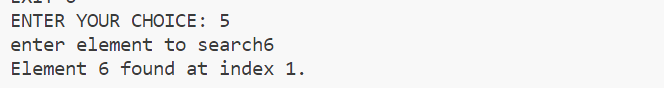












|  |
| --- |
| **Program 4** |
| **Search for all the occurrences of an element in an integer array (positions)** |

**Source Code:**

#include<stdio.h>

int main()

{

    int a[10],i,n,num,p[10];

    int count=0;

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

    scanf("%d",&n);

    printf("Enter array elements: ");

    for(i=0;i<n;i++)

    {

        scanf("%d",&a[i]);

    }

    printf("Enter the array elment to find: ");

    scanf("%d",&num);

    for(i=0;i<n;i++)

    {

        if(a[i]==num)

        {

            p[count]=i;

            count++;

        }

    }

    printf("Occurrence of %d is: %d Times\n", num, count);

    printf("Positions of %d are index: ", num);

    for(i = 0; i < count; i++)

    {

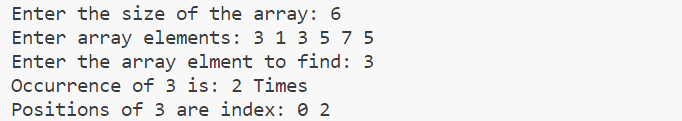
        printf("%d ", p[i]);

    }

    return 0;

}

**Output:**



|  |
| --- |
| **Program 5** |
| **Sort the array elements in ascending order (minimum three functions - read, disp and sort).** |

**Source Code:**

#include<stdio.h>

#define MAX\_SIZE 100

int arr[MAX\_SIZE],n,i;

void read()

{

    printf(" Enter array elements:");

    for(i=0;i<n;i++)

    {

    scanf("%d",&arr[i]);

    }

}

void disp()

{

    printf(" Array elements are:");

    for(i=0;i<n;i++)

    {

    printf(" %d",arr[i]);

    }

}

void sort()

{

    int j,t;

    for(i=0;i<n;i++){

        for(j=0;j<(n-1)-i;j++){

            if(arr[j]>arr[j+1]){

                t=arr[j];

                arr[j]=arr[j+1];

                arr[j+1]=t;

            }

        }

    }

    printf("Array sorted");

}

void del()

{

    int i;

    if(n == -1){

        printf("ARRAY IS EMPTY");

    }

    else{

        printf("Enter the index of the element to delete: ");

        int pos;

        scanf("%d", &pos);

        if(pos < 0 || pos >= n){

            printf("Invalid index");

        }

        else{

            printf("Deleted element is %d ", arr[pos]);

            for(i = pos; i < n - 1; i++){

                arr[i] = arr[i + 1];

            }

            n--;

        }

    }

}

int menu()

{

    int ch;

    printf("\n INSERT-1\n DISPLAY-2\n SORT-3\n DELETE-4\n EXIT-5\n Enter your choice: ");

    scanf("%d",&ch);

    return ch;

}

void process()

{

    int ch;

    for(ch=menu();ch!=5;ch=menu())

    {

        switch(ch)

        {

            case 1:

                printf("enter the number of elements to enter: ");

                scanf("%d",&n);

                read();

                break;

            case 2:

                disp();

                break;

            case 3:

                sort();

                break;

            case 4:

                del();

                break;

            default:

                printf("wrong choice");

                break;

        }

    }

}

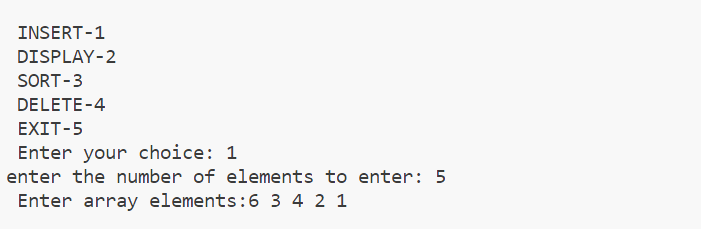
int main()

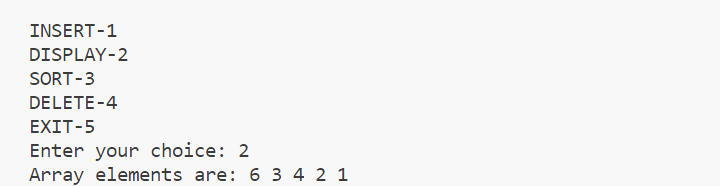
{

    process();

}

**Output:**

****







|  |
| --- |
| **Program 6** |
| **Two-dimensional matrix: using functions**   * 1. **Addition**   2. **Subtraction**   3. **Multiplication**   4. **Transpose**   5. **Determinant** |

**Source Code:**

#include <stdio.h>

#include <process.h>

void add(int a[10][10], int b[10][10], int m, int n)

{

  int i, j, sum[10][10];

  for (i = 0; i < m; i++)

  {

    for (j = 0; j < n; j++)

      sum[i][j] = a[i][j] + b[i][j];

  }

  printf("\n Addition :");

  for (i = 0; i < m; i++)

  {

    printf("\n");

    for (j = 0; j < n; j++)

      printf("%d\t", sum[i][j]);

  }

}

void sub(int a[10][10], int b[10][10], int m, int n)

{

  int i, j, sub[10][10];

  for (i = 0; i < m; i++)

  {

    for (j = 0; j < n; j++)

      sub[i][j] = a[i][j] - b[i][j];

  }

  printf("\n Subtraction :");

  for (i = 0; i < m; i++)

  {

    printf("\n");

    for (j = 0; j < n; j++)

      printf("%d\t", sub[i][j]);

  }

}

void det(int a[10][10], int m, int n)

{

  int det, i, j;

  if (m == 2)

  {

    det = (a[0][0] \* a[1][1]) - (a[0][1] \* a[1][0]);

    printf("%d", det);

  }

  else if (m == 3)

  {

    det = a[0][0] \* ((a[1][1] \* a[2][2]) - (a[2][1] \* a[1][2])) - a[0][1] \* (a[1][0] \* a[2][2] - a[2][0] \* a[1][2]) + a[0][2] \* (a[1][0] \* a[2][1] - a[2][0] \* a[1][1]);

    printf("%d", det);

  }

}

void trans(int a[10][10], int m, int n)

{

  int trans[10][10], i, j;

  for (i = 0; i < m; i++)

  {

    for (j = 0; j < n; j++)

      trans[i][j] = a[j][i];

  }

  for (i = 0; i < m; i++)

  {

    printf("\n");

    for (j = 0; j < n; j++)

      printf("%d\t", trans[i][j]);

  }

}

void mul(int a[10][10], int b[10][10], int m, int n, int p, int q)

{

  int k, prod[10][10], i, j;

  for (i = 0; i < m; i++)

  {

    for (j = 0; j < q; j++)

      prod[i][j] = 0;

  }

  for (i = 0; i < m; i++)

  {

    for (j = 0; j < q; j++)

    {

      for (k = 0; k < n; k++)

        prod[i][j] += a[i][k] \* b[k][j];

    }

  }

  printf("\n Multiplication :");

  for (i = 0; i < m; i++)

  {

    printf("\n");

    for (j = 0; j < n; j++)

      printf("%d\t", prod[i][j]);

  }

}

void main()

{

  int a[10][10], b[10][10], m, i, j, n, p, q, ch;

  printf("\n\t\tMATRIX OPERATIONS");

  printf("\n\t\t-----------------");

  printf("\n Enter row and column of matrix A:");

  scanf("%d%d", &m, &n);

  printf("\n Enter row and column of matrix B:");

  scanf("%d%d", &p, &q);

  printf("\n Enter elements of matrix A:");

  for (i = 0; i < m; i++)

  {

    for (j = 0; j < n; j++)

      scanf("%d", &a[i][j]);

  }

  printf("\n Enter elements of matrix B:");

  for (i = 0; i < p; i++)

  {

    for (j = 0; j < q; j++)

      scanf("%d", &b[i][j]);

  }

  printf("\n----------");

  printf("\nMATRIX A :");

  printf("\n----------");

  for (i = 0; i < m; i++)

  {

    printf("\n");

    for (j = 0; j < n; j++)

      printf("%d\t", a[i][j]);

  }

  printf("\n----------");

  printf("\n MATRIX B :");

  printf("\n----------");

  for (i = 0; i < p; i++)

  {

    printf("\n");

    for (j = 0; j < q; j++)

      printf("%d\t", b[i][j]);

  }

  do

  {

    printf("\n=====\nMenu\n=====\n1.Addition\n2.Subtraction\n3.Multiplication\n4.Determinant\n5.Transpose\n6.Exit\nEnter your choice:");

    scanf("%d", &ch);

    switch (ch)

    {

    case 1:

      if (m == p && n == q)

        add(a, b, m, n);

      else

        printf("Not possible...");

      break;

    case 2:

      if (m == p && n == q)

        sub(a, b, m, n);

      else

        printf("Not possible.....");

      break;

    case 3:

      mul(a, b, m, n, p, q);

      break;

    case 4:

      if (m == n && p == q)

      {

        printf("\n Determinant of matrix A=");

        det(a, m, n);

        printf("\n Determinant of matrix B=");

        det(b, p, q);

      }

      else

        printf("Not possible...");

      break;

    case 5:

      printf("\n Transpose of matrix A:");

      trans(a, m, n);

      printf("\n Transpose of matrix B :");

      trans(b, p, q);

      break;

    case 6:

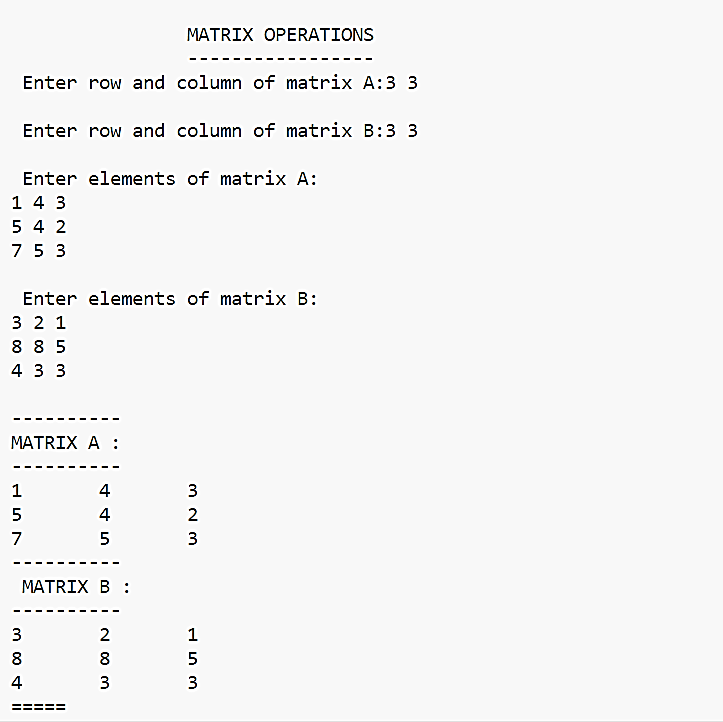
      exit(1);

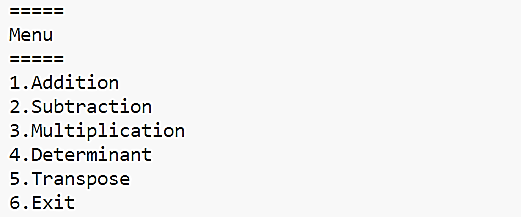
    }

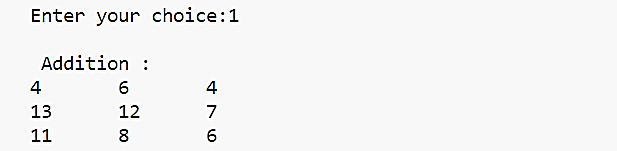
  } while (1);

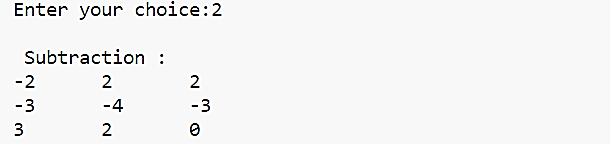
}

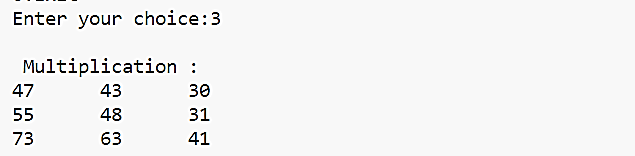
**Output:**

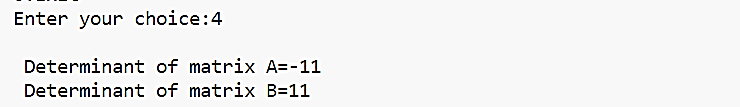


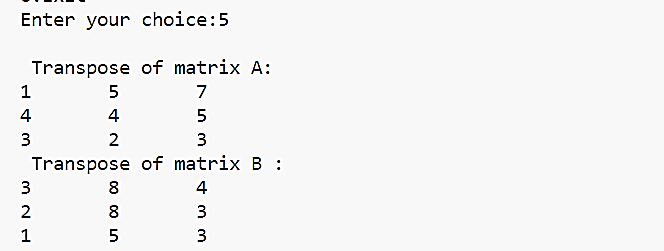












|  |
| --- |
| **Program 7** |
| **Display the array elements in the same order using a recursive function.** |

**Source Code:**

#include<stdio.h>

int dispArr(int a[10],int size,int i){

if(i==size){

return 0;

}

else{

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

i++;

return dispArr(a,size,i);

}

}

void main(){

int size,i,a[10];

printf("\nEnter the size:");

scanf("%d",&size);

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

printf("\nEnter element %d:",i+1);

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

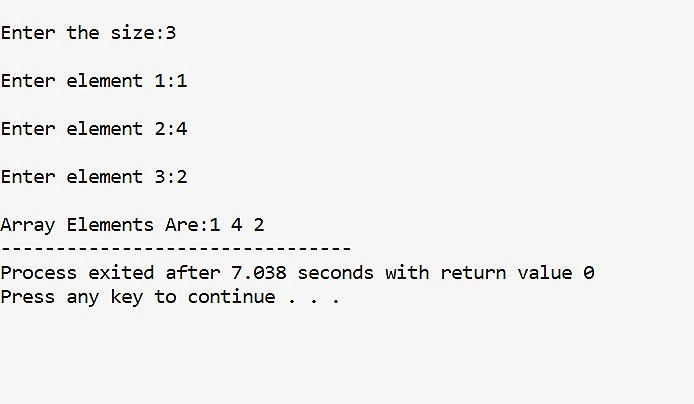
}

printf("\nArray Elements Are:");

dispArr(a,size,0);

}

**Output:**



|  |
| --- |
| **Program 8** |
| **Display array elements in the reverse order using a recursive function.** |

**Source Code:**

#include<stdio.h>

int dispArrRev(int a[10],int size){

if(size==0){

return 0;

}

else{

size=size-1;

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

return dispArrRev(a,size);

}

}

void main(){

int size,i,a[10];

printf("\nEnter the size:");

scanf("%d",&size);

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

printf("\nEnter element %d:",i+1);

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

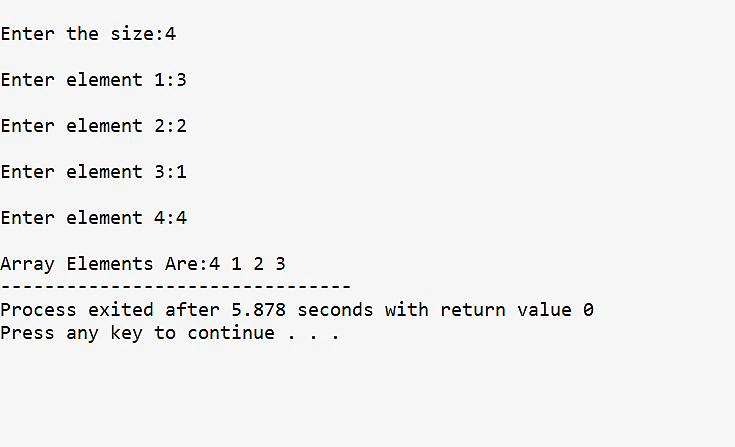
}

printf("\nArray Elements Are:");

dispArrRev(a,size);

}

**Output:**



|  |
| --- |
| **Program 9** |
| **Implement stack operations using array.** |

**Source Code:**

#include <stdio.h>

#define MAX\_SIZE 100

int top = -1;

int stack[MAX\_SIZE];

void push(int element) {

    if (top == MAX\_SIZE - 1) {

        printf("Stack overflow\n");

        return;

    }

    top++;

    stack[top] = element;

    printf("\n%d is pushed",element);

}

void pop() {

    if (top == -1) {

        printf("Stack underflow\n");

    }

    else{

            printf("\n%d is popped",stack[top]);

            top--;

    }

}

void peek()

{

    if(top==-1)

    {

        printf("stack underflow\n");

    }

    else

    {

        printf("\nElement at top is %d\n",stack[top]);

    }

}

int main() {

    push(1);

    push(2);

    push(3);

    peek();

    pop();

    pop();

    pop();

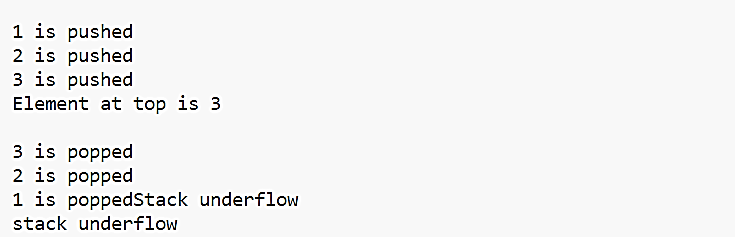
    pop();

    peek();

    return 0;

}

**Output:**

****

|  |
| --- |
| **Program 10** |
| **Reverse a string using Stack.** |

**Source Code:**

#include<stdio.h>

#include<string.h>

char stack[10],top =-1;

char a[10];

int i;

void pop()

{

    printf("Reversed string is");

    for(i=top;top>-1;top--)

    {

        printf(" %c",stack[top]);

    }

}

void push()

{

    for(i=0;i<strlen(a);i++)

    {

        top++;

        stack[top]=a[i];

    }

}

int main()

{

    printf("Enter the string:");

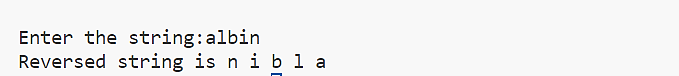
    gets(a);

    push(a);

    pop();

}

**Output:**



|  |
| --- |
| **Program 11** |
| **Convert an expression from infix to postfix using stack** |

**Source Code:**

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#define MAX 100

char stack[MAX];

char infix[MAX], postfix[MAX];

int top = -1;

void push(char);

char pop();

int isEmpty();

void inToPost();

int isOperator(char);

int precedence(char);

void print();

int main()

{

    printf("Enter the infix expression: ");

    gets(infix);

    inToPost();

    print();

    return 0;

}

void inToPost()

{

    int i, j = 0;

    char symbol, next;

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

    {

        symbol = infix[i];

        if (symbol == '(')

        {

            push(symbol);

        }

        else if (symbol == ')')

        {

            while ((next = pop()) != '(')

                postfix[j++] = next;

        }

        else if (isOperator(symbol))

        {

            while (!isEmpty() && precedence(stack[top]) >= precedence(symbol))

                postfix[j++] = pop();

            push(symbol);

        }

        else

        {

            postfix[j++] = symbol;

        }

    }

    while (!isEmpty())

        postfix[j++] = pop();

    postfix[j] = '\0';

}

int isOperator(char c)

{

    return (c == '+' || c == '-' || c == '\*' || c == '/' || c == '^');

}

int precedence(char symbol)

{

    switch (symbol)

    {

    case '^':

        return 3;

    case '/':

    case '\*':

        return 2;

    case '+':

    case '-':

        return 1;

    default:

        return 0;

    }

}

void print()

{

    int i = 0;

    printf("The equivalent postfix expression is: ");

    while (postfix[i])

    {

        printf("%c", postfix[i++]);

    }

    printf("\n");

}

void push(char c)

{

    if (top == MAX - 1)

    {

        printf("Stack overflow\n");

        exit(1);

    }

    top++;

    stack[top] = c;

}

char pop()

{

    char c;

    if (top == -1)

    {

        printf("Stack underflow\n");

        exit(1);

    }

    c = stack[top];

    top = top - 1;

    return c;

}

int isEmpty()

{

    if (top == -1)

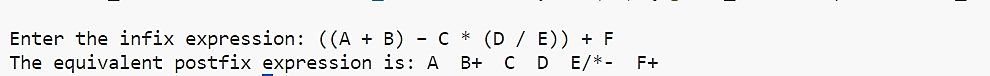
        return 1;

    else

        return 0;

}

**Output:**

****

|  |
| --- |
| **Program 12** |
| **Evaluate an expression using stack.** |

**Source Code:**

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include <ctype.h>

#include <math.h>

#define MAX 10

char stk[MAX];

int top = -1;

void push(char x)

{

    top++;

    stk[top] = x;

}

char pop()

{

    char y = stk[top];

    top--;

    return y;

}

int precedence(char k)

{

    if (k == '^')

    {

        return 3;

    }

    else if (k == '\*' || k == '/')

    {

        return 2;

    }

    else if (k == '+' || k == '-')

    {

        return 1;

    }

    else

    {

        return 0;

    }

}

void conversion()

{

    char infix[MAX], postfix[MAX];

    printf("Enter infix expression: ");

    scanf("%s", infix);

    push('#');

    int i = 0, j = 0;

    char temp, k;

    while (infix[i] != '\0')

    {

        temp = infix[i];

        switch (temp)

        {

        case '(':

            push(temp);

            break;

        case ')':

            k = pop();

            while (k != '(')

            {

                postfix[j] = k;

                j++;

                k = pop();

            }

            break;

        case '^':

        case '\*':

        case '/':

        case '+':

        case '-':

            while (precedence(stk[top]) >= precedence(temp))

            {

                postfix[j] = pop();

                j++;

            }

            push(temp);

            break;

        default:

            postfix[j] = temp;

            j++;

        }

        i++;

    }

    while (top > 0)

    {

        postfix[j] = pop();

        j++;

    }

    postfix[j] = '\0';

    printf("Postfix expression: %s\n", postfix);

    int resultstk[MAX];

    int resTop = -1;

    int operand1, operand2;

    i = 0;

    while (postfix[i] != '\0')

    {

        if (isdigit(postfix[i]))

        {

            push(postfix[i] - '0');

        }

        else

        {

            operand2 = pop();

            operand1 = pop();

            switch (postfix[i])

            {

            case '^':

                push((char)pow(operand1, operand2));

                break;

            case '\*':

                push((char)(operand1 \* operand2));

                break;

            case '/':

                push((char)(operand1 / operand2));

                break;

            case '+':

                push((char)(operand1 + operand2));

                break;

            case '-':

                push((char)(operand1 - operand2));

                break;

            }

        }

        i++;

    }

    int evaluationResult = pop();

    printf("Expression Evaluation Result: %d\n", evaluationResult);

}

int main()

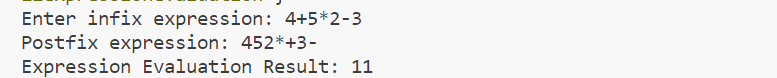
{

    conversion();

    return 0;

}

**Output:**



|  |
| --- |
| **Program 13** |
| **Define a structure for data having**  **dd/mm/yyyy. Provide functions for reading, displaying and comparing two dates are equal or not.** |

**Source Code:**

#include <stdio.h>

struct date

{

    int day1, month1, year1;

    int day2, month2, year2;

};

struct date d;

void insert()

{

    printf("Enter the first date in the format dd/mm/yyyy: ");

    scanf("%d/%d/%d", &d.day1, &d.month1, &d.year1);

    printf("Enter the second date in the format dd/mm/yyyy: ");

    scanf(" %d/%d/%d", &d.day2, &d.month2, &d.year2);

}

void display()

{

    printf("First Date: %d/%d/%d\n", d.day1, d.month1, d.year1);

    printf("Second Date: %d/%d/%d\n", d.day2, d.month2, d.year2);

}

void compare()

{

    if (d.day1 == d.day2 && d.month1 == d.month2 && d.year1 == d.year2)

    {

        printf("Both Date Are Same\n");

    }

    else

    {

        printf("Both Date Are Different\n");

    }

}

int main()

{

    int choice;

    while (1)

    {

        printf("\nDate Comparison Menu:\n");

        printf("1. Insert Dates\n");

        printf("2. Display Dates\n");

        printf("3. Compare Dates\n");

        printf("4. Exit\n");

        printf("Enter your choice: ");

        scanf("%d", &choice);

        switch (choice)

        {

        case 1:

            insert();

            break;

        case 2:

            display();

            break;

        case 3:

            compare();

            break;

        case 4:

            printf("Goodbye!\n");

            return 0;

        default:

            printf("Invalid choice. Please try again.\n");

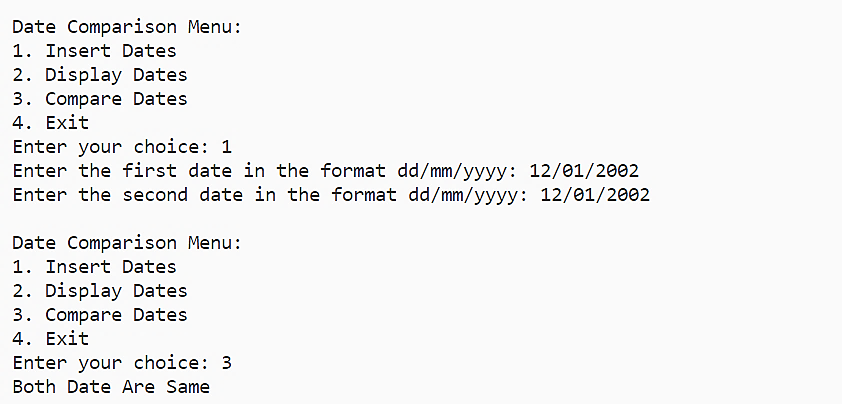
        }

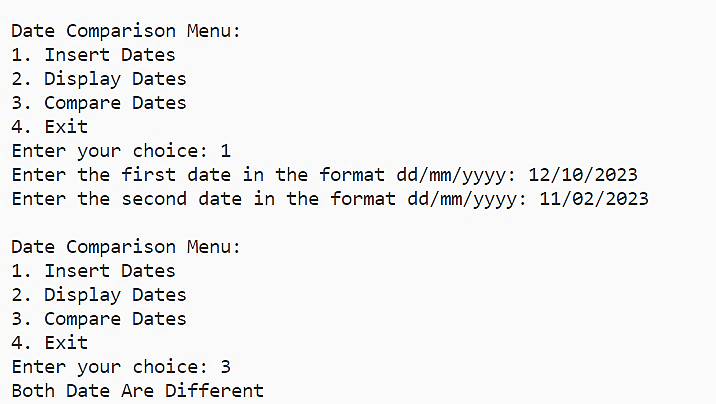
    }

    return 0;

}

**Output:**



****

|  |
| --- |
| **Program 14** |
| **Define a structure for employees with eno,ename, esal and dno. Read n employees information and provide functions for the following:**   * 1. **Searching an employee by no**   2. **Sorting the employees by**      1. **Name**      2. **Salary**   3. **Deleting an employee** |

**Source Code:**

#include <stdio.h>

#include <string.h>

#include <process.h>

struct emp

{

    int eno, esal, dno;

    char ename[10];

};

int i, j, n;

struct emp e[20];

void read()

{

    for (i = 0; i < n; i++)

    {

        printf("\n\tEnter Details of Employee-%d", i + 1);

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

        printf("\nEmployee no:");

        scanf("%d", &e[i].eno);

        printf("\nEmployee name:");

        scanf("%s", &e[i].ename);

        printf("\nEmployee salary:");

        scanf("%d", &e[i].esal);

        printf("\nDno:");

        scanf("%d", &e[i].dno);

    }

}

void search()

{

    int em;

    printf("\n Enter the employee number to search:");

    scanf("%d", &em);

    for (i = 0; i < n; i++)

    {

        if (em == e[i].eno)

        {

            printf("\n SEARCHED EMPLOYEE DETAILS:");

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

            printf("\n Employee No:%d", e[i].eno);

            printf("\n Employee Name:%s", e[i].ename);

            printf("\n Employee Salary:%d", e[i].esal);

            printf("\n DNo:%d", e[i].dno);

        }

    }

}

void sortname()

{

    struct emp t;

    for (i = 0; i < n; i++)

    {

        for (j = i + 1; j < n; j++)

        {

            if (strcmp(e[i].ename, e[j].ename) == 1)

            {

                t = e[i];

                e[i] = e[j];

                e[j] = t;

            }

        }

    }

    printf("\nEMPLOYEE LIST(SORTED USING NAME):");

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

    for (i = 0; i < n; i++)

    {

        printf("\nName:%s", e[i].ename);

        printf("\nEmployee No:%d", e[i].eno);

        printf("\nEmployee Dno:%d", e[i].dno);

        printf("\nEmployee Salary:%d\n", e[i].esal);

    }

}

void sortsal()

{

    struct emp t;

    for (i = 0; i < n; i++)

    {

        for (j = i + 1; j < n; j++)

        {

            if (e[i].esal > e[j].esal)

            {

                t = e[i];

                e[i] = e[j];

                e[j] = t;

            }

        }

    }

    printf("\nEMPLOYEE LIST(SORTED USING SALARY) :");

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

    for (i = 0; i < n; i++)

    {

        printf("\nEmployee Name:%s", e[i].ename);

        printf("\nEmployee No:%d", e[i].eno);

        printf("\nDno:%d", e[i].dno);

        printf("\nEmployee Salary:%d\n", e[i].esal);

    }

}

void delet(int en)

{

    if (n == 0)

        printf("\n No Employee!!!");

    else

    {

        printf("\nLIST OF EMPLOYEES");

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

        for (i = 0; i < n; i++)

        {

            if (en == e[i].eno)

            {

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

                    e[j] = e[j + 1];

            }

        }

        n--;

    }

    for (i = 0; i < n; i++)

    {

        printf("\nEmployee No:%d", e[i].eno);

        printf("\nEmployee Name:%s", e[i].ename);

        printf("\nDno:%d", e[i].dno);

        printf("\nEmployee Salary:%d", e[i].esal);

    }

}

void main()

{

    int ch, ch1, en;

    printf("\n Enter number of employees:");

    scanf("%d", &n);

    read(n);

    do

    {

        printf("\n------\n MENU\n-------\n 1.Search\n 2.Sort\n 3.Delete\n 4.Exit\n Enter your choice:");

        scanf("%d", &ch);

        switch (ch)

        {

        case 1:

            search();

            break;

        case 2:

            printf("\nSorting:");

            printf("\n---------");

            printf("\n 1.Sort by name\n 2.Sort by salary\n Enter your choice: ");

            scanf("%d", &ch1);

            switch (ch1)

            {

            case 1:

                sortname();

                break;

            case 2:

                sortsal();

                break;

            }

            break;

        case 3:

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

            scanf("%d", &en);

            delet(en);

            break;

        case 4:

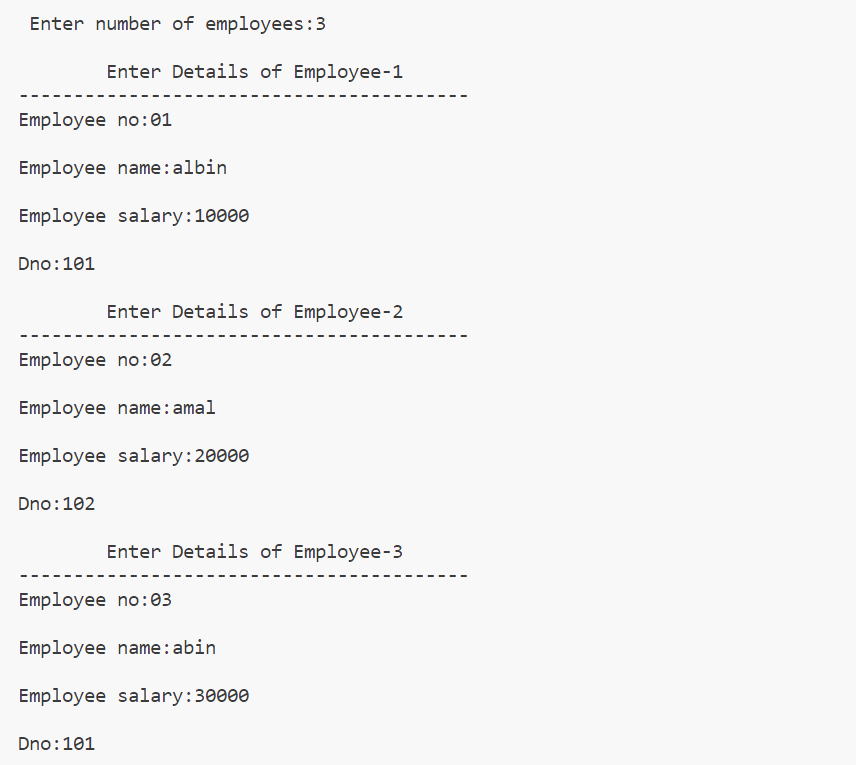
            exit(0);

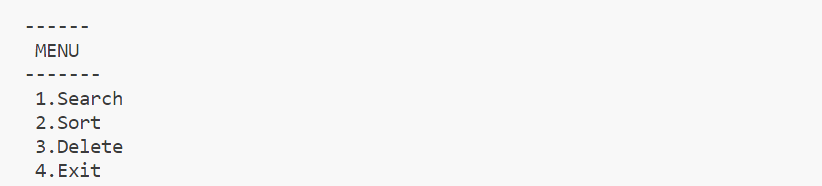
        }

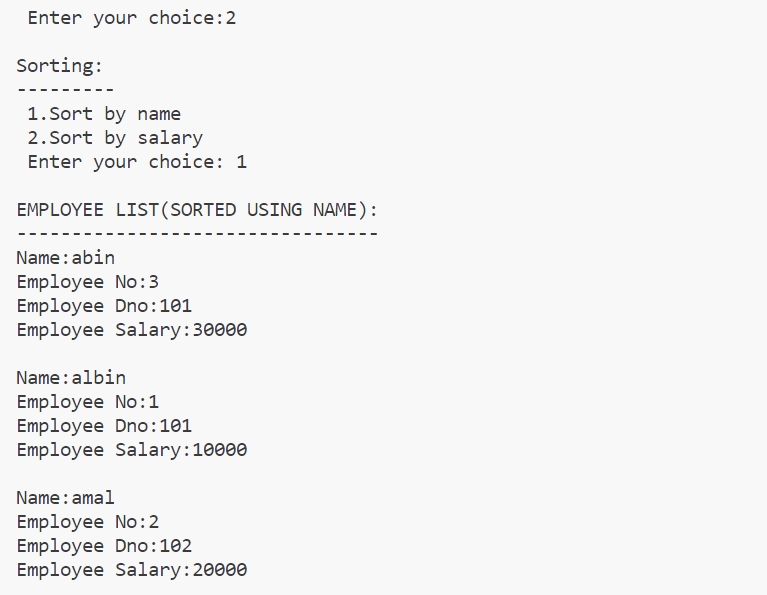
    } while (1);

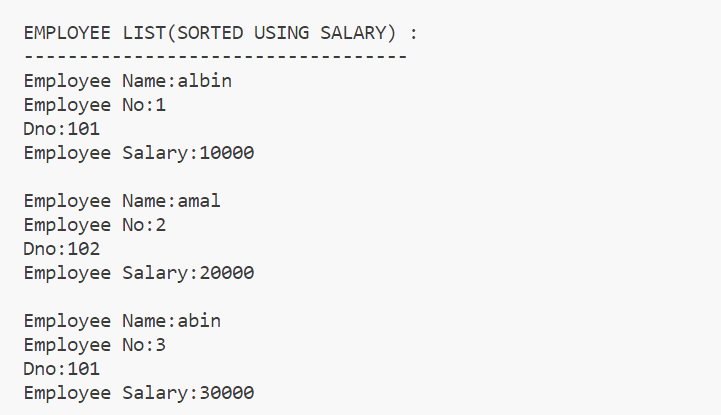
}

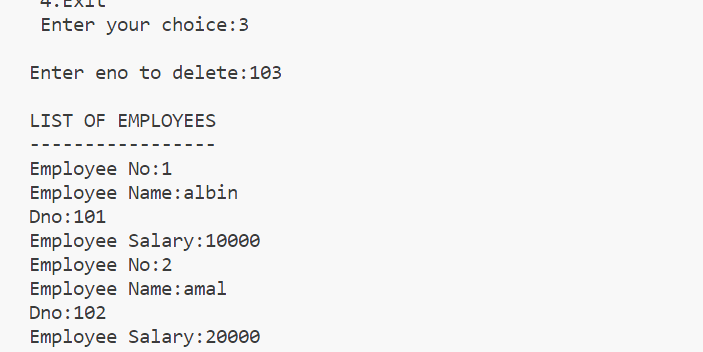
**Output:**











|  |
| --- |
| **Program 15** |
| **Read a polynomial and display –use array** |

**Source Code:**

#include <stdio.h>

int i;

void disp(int a[], int m)

{

  printf("\nThe polynomial is : ");

  for (i = m; i >= 0; i--)

  {

    if (a[i] != 0 && i != 0)

      printf("%dx^%d+", a[i], i);

    else if (i == 0)

      printf("%d", a[i]);

  }

}

void main()

{

  int a[20], b[30], m, n;

  printf("\nEnter degree of the polynomial : ");

  scanf("%d", &m);

  printf("\nEnter the coefficients(enter constant first): ");

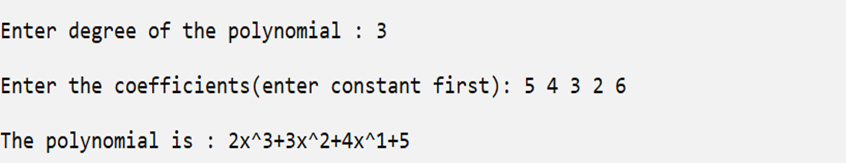
  for (i = 0; i <= m; i++)

    scanf("%d", &a[i]);

  disp(a, m);

}

**Output:**



|  |
| --- |
| **Program 16** |
| **Add two polynomials –use array itself.** |

**Source Code:**

#include <stdio.h>

int main()

{

    int c1[10], e1[10], c2[10], e2[10], c3[20], e3[20];

    int i, j, n1, n2, k = 0;

    printf("Enter the number of terms for polynomial 1: ");

    scanf("%d", &n1);

    printf("Enter the polynomial 1 terms:\n");

    for (i = 0; i < n1; i++)

    {

        printf("Enter the coefficient: ");

        scanf("%d", &c1[i]);

        printf("Enter the exponent: ");

        scanf("%d", &e1[i]);

    }

    printf("Enter the number of terms for polynomial 2: ");

    scanf("%d", &n2);

    printf("Enter the polynomial 2 terms:\n");

    for (i = 0; i < n2; i++)

    {

        printf("Enter the coefficient: ");

        scanf("%d", &c2[i]);

        printf("Enter the exponent: ");

        scanf("%d", &e2[i]);

    }

    i = j = 0;

    while (i < n1 && j < n2)

    {

        if (e1[i] == e2[j])

        {

            c3[k] = c1[i] + c2[j];

            e3[k] = e1[i];

            if (e3[k] != 0)

            { // Skip terms with an exponent of zero

                k++;

            }

            i++;

            j++;

        }

        else if (e1[i] > e2[j])

        {

            c3[k] = c1[i];

            e3[k] = e1[i];

            if (e3[k] != 0)

            { // Skip terms with an exponent of zero

                k++;

            }

            i++;

        }

        else

        {

            c3[k] = c2[j];

            e3[k] = e2[j];

            if (e3[k] != 0)

            { // Skip terms with an exponent of zero

                k++;

            }

            j++;

        }

    }

    while (i < n1)

    {

        c3[k] = c1[i];

        e3[k] = e1[i];

        if (e3[k] != 0)

        { // Skip terms with an exponent of zero

            k++;

        }

        i++;

    }

    while (j < n2)

    {

        c3[k] = c2[j];

        e3[k] = e2[j];

        if (e3[k] != 0)

        { // Skip terms with an exponent of zero

            k++;

        }

        j++;

    }

    printf("Resultant polynomial after addition:\n");

    for (i = 0; i < k; i++)

    {

        if (c3[i] != 0)

        {

            if (c3[i] > 0 && i != 0)

            {

                printf(" + ");

            }

            printf("%dx^%d", c3[i], e3[i]);

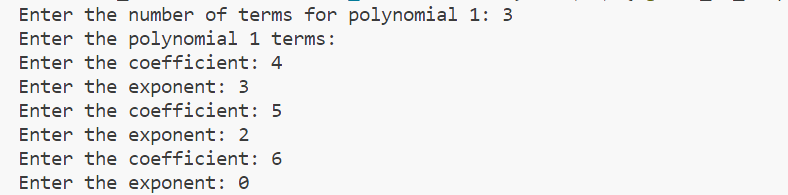
        }

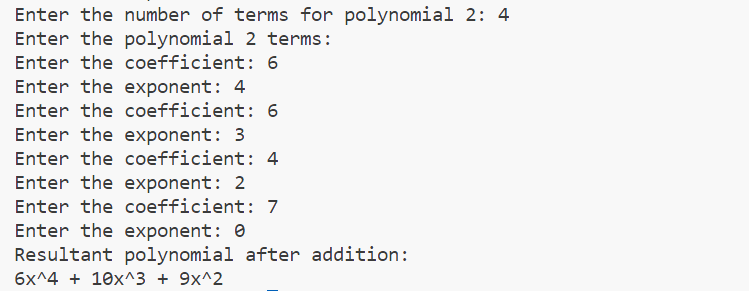
    }

    return 0;

}

**Output:**





|  |
| --- |
| **Program 17** |
| **Read a polynomial and display - use structure array.** |

**Source Code:**

#include <stdio.h>

struct poly

{

    int coeff;

    int exp;

};

int main()

{

    int i, num;

    struct poly p[10];

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

    scanf("%d", &num);

    for (i = 0; i < num; i++)

    {

        printf("Enter the coefficient: ");

        scanf("%d", &p[i].coeff);

        printf("Enter the degree: ");

        scanf("%d", &p[i].exp);

    }

    printf("The entered polynomial is: ");

    for (i = 0; i < num; i++)

    {

        printf("%dx^%d", p[i].coeff, p[i].exp);

        if (i < num - 1)

        {

            printf(" + ");

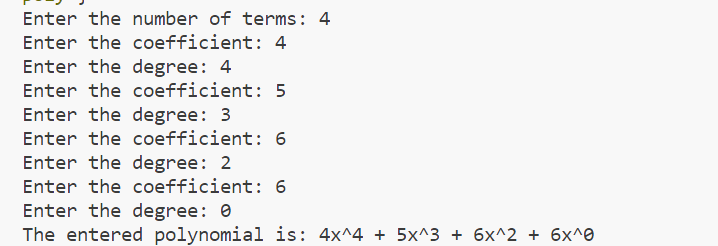
        }

    }

    return 0;

}

**Output**



|  |
| --- |
| **Program 18** |
| **Add two polynomials use structure array.** |

**Source Code:**

#include <stdio.h>

struct poly

{

  int coe;

  int ex;

};

int main()

{

  int i, j, n1, n2, k = 0;

  struct poly p1[10], p2[10], p3[10];

  printf("enter the no of terms: ");

  scanf("%d", &n1);

  printf("enter the polynomial 1\n");

  for (i = 0; i < n1; i++)

  {

    printf("enter the coeffient: ");

    scanf("%d", &p1[i].coe);

    printf("enter the exponent: ");

    scanf("%d", &p1[i].ex);

  }

  printf("enter the no of terms: ");

  scanf("%d", &n2);

  printf("enter the polynomial 2\n");

  for (i = 0; i < n2; i++)

  {

    printf("enter the coeffient: ");

    scanf("%d", &p2[i].coe);

    printf("enter the exponent: ");

    scanf("%d", &p2[i].ex);

  }

  i = j = 0;

  while (i < n1 && j < n2)

  {

    if (p1[i].ex == p2[j].ex)

    {

      p3[k].coe = p1[i].coe + p2[i].coe;

      p3[k].ex = p1[i].ex;

      i++;

      j++;

      k++;

    }

    else if (p1[i].ex > p2[j].ex)

    {

      p3[k].coe = p1[i].coe;

      p3[k].ex = p1[i].ex;

      i++;

      k++;

    }

    else

    {

      p3[k].coe = p2[j].coe;

      p3[k].ex = p2[j].ex;

      j++;

      k++;

    }

  }

  while (i < n1)

  {

    p3[k].coe = p1[i].coe;

    p3[k].ex = p1[i].ex;

    i++;

    k++;

  }

  while (j < n2)

  {

    p3[k].coe = p2[j].coe;

    p3[k].ex = p2[j].ex;

    j++;

    k++;

  }

  printf("\n POLYNOMIALS ADDITION");

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

  for (i = 0; i < k; i++)

  {

    printf("%dx^%d", p3[i].coe, p3[i].ex);

    if (i < k - 1)

    {

      printf(" + ");

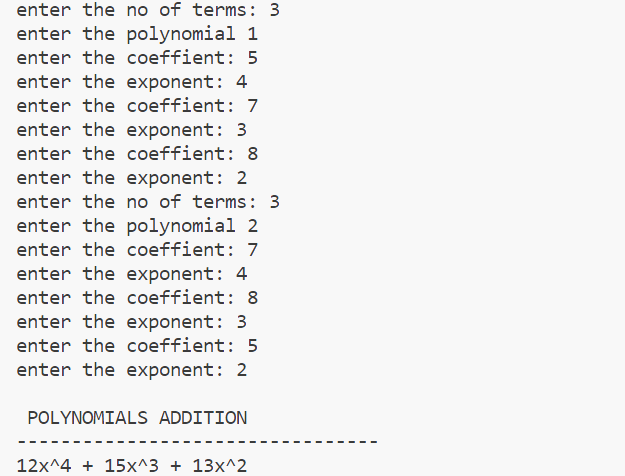
    }

  }

  return 0;

}

**Output:**



|  |
| --- |
| **Program 19** |
| **Subtract two polynomials.** |

**Source Code:**

#include<stdio.h>

struct poly{

int coe;

int ex;

};

int main()

{

int i,j,n1,n2,k=0;

struct poly p1[10],p2[10],p3[10];

printf("enter the no of terms\n");

scanf("%d",&n1);

printf("enter the polynomial 1\n");

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

printf("enter the coeffient\n");

scanf("%d",&p1[i].coe);

printf("enter the exponent\n");

scanf("%d",&p1[i].ex);

}

printf("enter the no of terms\n");

scanf("%d",&n2);

printf("enter the polynomial 2\n");

for(i=0;i<n2;i++){

printf("enter the coeffient\n");

scanf("%d",&p2[i].coe);

printf("enter the exponent\n");

scanf("%d",&p2[i].ex);

}

i=j=0;

while(i<n1 && j<n2){

if(p1[i].ex==p2[j].ex){

p3[k].coe=p1[i].coe-p2[i].coe;

p3[k].ex=p1[i].ex;

i++;j++;k++;

}

else if(p1[i].ex>p2[j].ex){

p3[k].coe=p1[i].coe;

p3[k].ex=p1[i].ex;

i++;k++;

}

else{

p3[k].coe=p2[j].coe;

p3[k].ex=p2[j].ex;

j++;k++;

}

}

while(i<n1){

p3[k].coe=p1[i].coe;

p3[k].ex=p1[i].ex;

i++;k++;

}

while(j<n2){

p3[k].coe=p2[j].coe;

p3[k].ex=p2[j].ex;

j++;k++;

}

printf("\n POLYNOMIALS SUBSTRATION");

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

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

printf("%dx^%d",p3[i].coe,p3[i].ex);

if (i < k - 1)

{

printf(" + ");

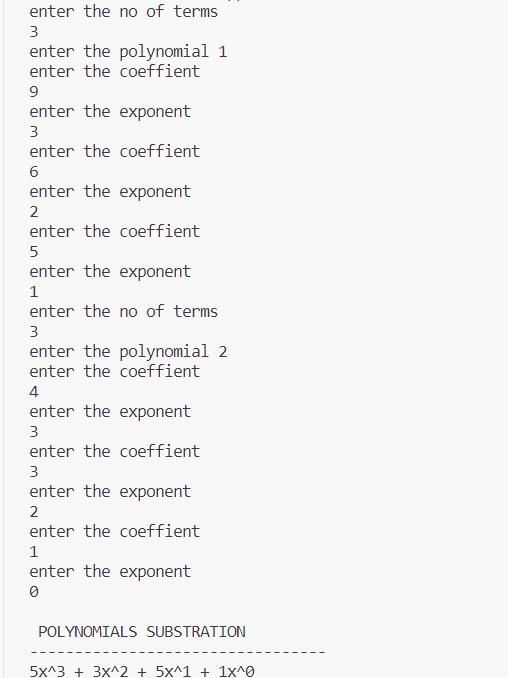
}

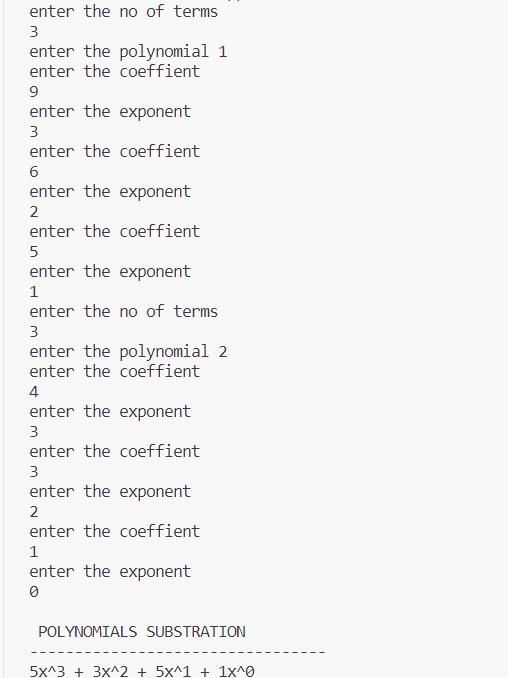
}

return 0;

}

**Output:**





|  |
| --- |
| **Program 20** |
| **Multiply two polynomials.** |

**Source Code:**

#include <stdio.h>

struct poly

{

  int coe;

  int exp;

};

int main()

{

  struct poly poly1[10], poly2[10], product[100];

  int noOfTerms1, noOfTerms2, count = -1;

  int i, j;

  printf("\nEnter Number Of Terms Of 1st Polynomial: ");

  scanf("%d", &noOfTerms1);

  for (i = 0; i < noOfTerms1; i++)

  {

    printf("\nEnter Coefficient: ");

    scanf("%d", &poly1[i].coe);

    printf("\nEnter Exponent: ");

    scanf("%d", &poly1[i].exp);

  }

  printf("\nEnter Number Of Terms Of 2nd Polynomial: ");

  scanf("%d", &noOfTerms2);

  for (i = 0; i < noOfTerms2; i++)

  {

    printf("\nEnter Coefficient: ");

    scanf("%d", &poly2[i].coe);

    printf("\nEnter Exponent: ");

    scanf("%d", &poly2[i].exp);

  }

  for (i = 0; i < noOfTerms1; i++)

  {

    for (j = 0; j < noOfTerms2; j++)

    {

      product[++count].exp = poly1[i].exp + poly2[j].exp;

      product[count].coe = poly1[i].coe \* poly2[j].coe;

    }

  }

  printf("\nThe Product Of Two Polynomials Is: \n");

  for (i = 0; i <= count; i++)

  {

    if (product[i].exp == 0)

      printf("%d ", product[i].coe);

    else if (product[i].exp == 1)

      printf("%dx ", product[i].coe);

    else

      printf("%dx^%d ", product[i].coe, product[i].exp);

    if (i != count)

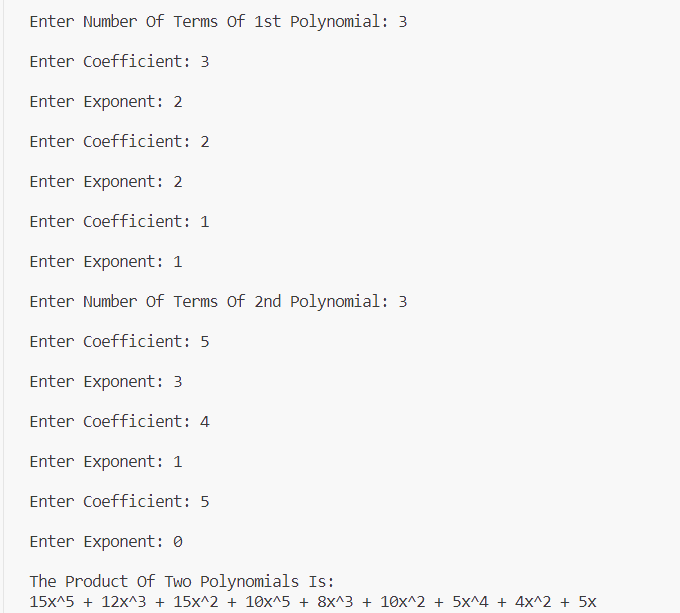
      printf("+ ");

  }

  return 0;

}

**Output:**



|  |
| --- |
| **Program 21** |
| **Implement a) malloc , b) calloc and c) free functions.** |

**Source Code:**

#include<stdio.h>

#include<process.h>

void main(){

int ch,n,p,i,\*a;

printf("\nEnter limit:");

scanf("%d",&n);

do{

printf("\n1.malloc\n2.calloc\n3.exit\nEnter your choice:");

scanf("%d",&ch);

switch(ch){

case 1:a=(int\*)malloc(n\*sizeof(int));

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

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

scanf("%d",&p);

a[i]=p;

}

printf("\n Elements are:");

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

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

free(a);

break;

case 2:a=(int\*)calloc(n,sizeof(int));

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

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

scanf("%d",&p);

a[i]=p;

}

printf("\n Elements are:");

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

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

free(a);

break;

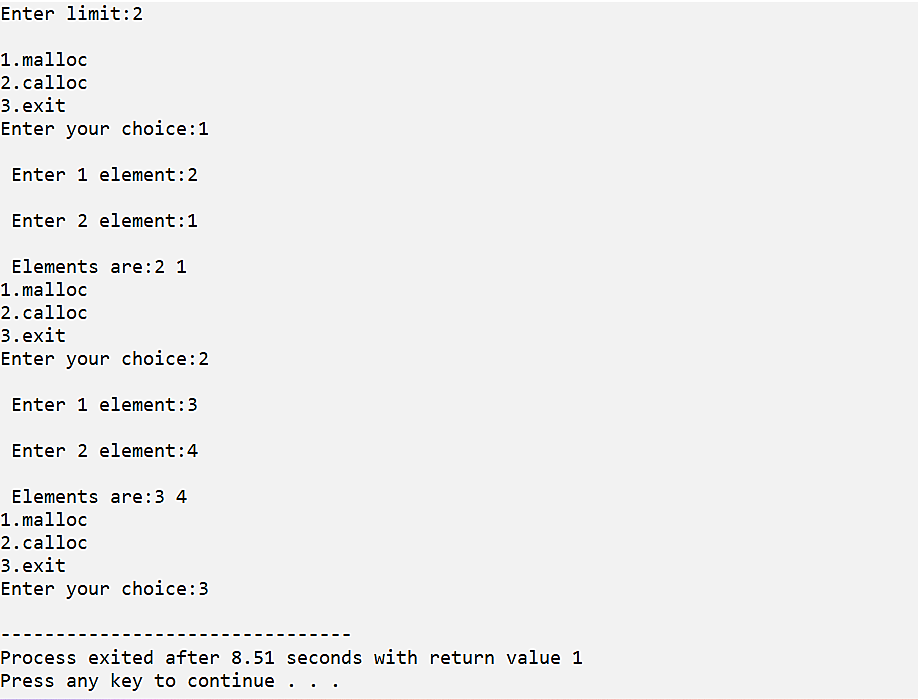
case 3:exit(1);

}

}while(1);

}

**Output:**



|  |
| --- |
| **Program 22** |
| **Use malloc to read n integers and find the mean.** |

**Source Code:**

#include<stdio.h>

void main(){

int n,p,sum,i,\*a;

printf("\n Enter limit:");

scanf("%d",&n);

a=(int\*)malloc(n\*sizeof(int));

printf("\n Enter elements:");

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

scanf("%d",&p);

a[i]=p;

sum+=a[i];

}

printf("\n Elements are:");

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

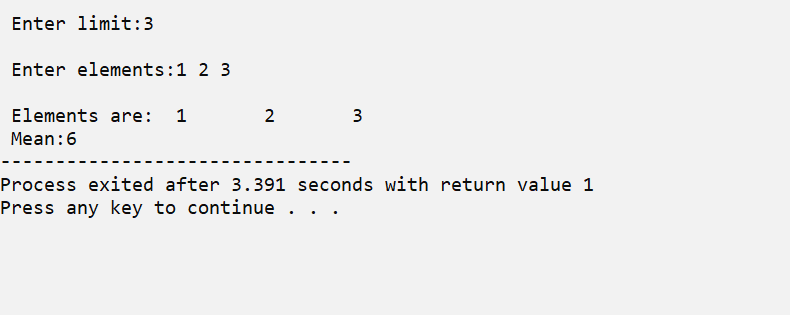
printf("\t%d",a[i]);

printf("\n Mean:%d",sum);

free(a);

}

**Output:**



|  |
| --- |
| **Program 23** |
| **Use calloc to read n numbers and find the mode.** |

**Source Code:**

#include <stdio.h>

#include <stdlib.h>

int max\_val[5];

void main()

{

  int n, \*ptr, i, y;

  int x;

  printf("Enter the limit: ");

  scanf("%d", &n);

  ptr = (int \*)calloc(n, sizeof(int));

  printf("Enter the values to calculate mode:\n");

  for (i = 0; i < n; i++)

  {

    scanf("%d", &ptr[i]);

  }

  printf("\n====\nMODE\n====\n");

  mode(ptr, n);

  free(ptr);

}

int mode(int ptr[], int n)

{

  int i, j, k = 0;

  int max\_count = 0;

  for (i = 0; i < n; i++)

  {

    int count = 0;

    for (j = i + 1; j < n; j++)

    {

      if (ptr[i] == ptr[j])

      {

        count = count + 1;

      }

    }

    if (count > max\_count)

    {

      max\_count = count;

    }

  }

  for (i = 0; i < n; i++)

  {

    int count = 0;

    for (j = i + 1; j < n; j++)

    {

      if (ptr[i] == ptr[j])

    {

        count = count + 1;

      }

    }

    if (count == max\_count)

    {

      printf("\n%d", ptr[i]);

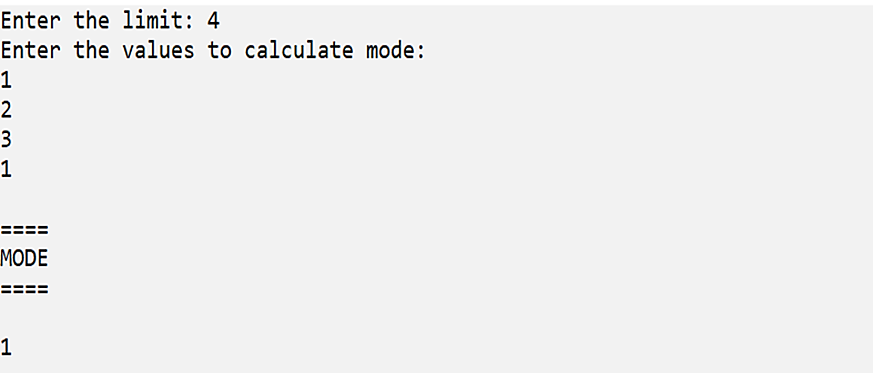
    }

  }

  return 0;

}

**Output:**



|  |
| --- |
| **Program 24** |
| **Declare a structure for Books having author\_name and book\_name. Create an array of books using a pointer variable. Provide functions for reading n books and displaying the same using pointers.** |

**Source Code:**

#include <stdio.h>

#include <stdlib.h>

struct book

{

    char author\_name[20];

    char book\_name[20];

};

struct book \*a;

int i, n;

void read()

{

    a = (struct book \*)malloc(sizeof(struct book));

    for (i = 0; i < n; i++)

    {

        printf("\n Enter details of book %d", i + 1);

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

        printf("\n Enter book name:");

        gets(a[i].book\_name);

        printf("\n Enter author name:");

        gets(a[i].author\_name);

    }

}

void disp()

{

    for (i = 0; i < n; i++)

    {

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

        printf("\nDetails of book %d", i + 1);

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

        printf("\n Book name:%s", a[i].book\_name);

        printf("\n Author name:%s", a[i].author\_name);

    }

}

void main()

{

    printf("\n Enter no of books you want to enter:");

    scanf("%d", &n);

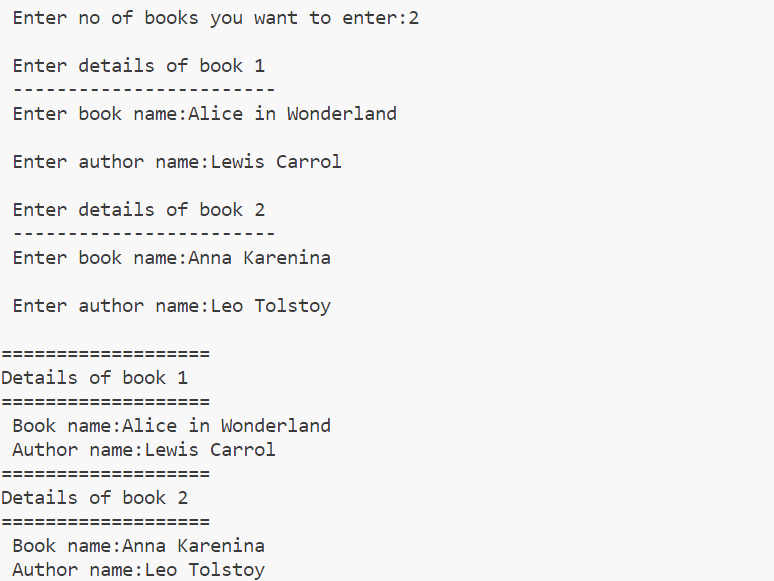
    getchar();

    read();

    disp();

}

**Output:**

****

|  |
| --- |
| **Program 25** |
| **Use realloc to implement varchar for any length.** |

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

int main()

{

    char \*ptr;

    char str[50];

    int len, n, i;

    printf("\nEnter the string : ");

    scanf("%s", &str);

    len = strlen(str);

    ptr = (char \*)malloc(len \* sizeof(char));

    strcpy(ptr, str);

    printf("\nThe string using malloc is : ");

    for (i = 0; i < len; i++)

    {

        printf("%c", \*(ptr + i));

    }

    printf("\n\nEnter the new size : ");

    scanf("%d", &n);

    ptr = (char \*)realloc(ptr, n);

    printf("\nThe string using realloc is : ");

    for (i = 0; i < n && ptr[i] != '\0'; i++)

    {

        printf("%c", \*(ptr + i));

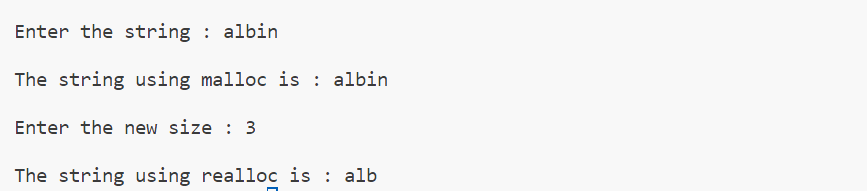
    }

    free(ptr);

    return 0;

}

**Output:**



|  |
| --- |
| **Program 26** |
| **Implement Queue using array.** |

**Source Code:**

#include <stdio.h>

int q[30], n;

int f = -1, r = -1;

void enqueue(int e)

{

    if ((f == -1) && (r == -1))

    {

        f = r = 0;

        q[r] = e;

    }

    else if (r == n - 1)

        printf("\nQueue Full\n");

    else

    {

        r++;

        q[r] = e;

    }

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

}

void dequeue()

{

    if ((f == -1) && (r == -1))

        printf("\nqueue empty!!\n");

    else if (f == r)

    {

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

        r = f = -1;

    }

    else

    {

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

        f++;

    }

}

int menu()

{

    int ch;

    printf("\n 1-ENQUEUE\n 2-DEQUEUE\n 3-EXIT\n Enter the choice: ");

    scanf("%d", &ch);

    return ch;

}

int main()

{

    int i;

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

    scanf("%d", &n);

    for (i = menu(); i != 3; i = menu())

    {

        switch (i)

        {

        case 1:

            printf("\nEnter the value to enqueue: ");

            scanf("%d", &i);

            enqueue(i);

            break;

        case 2:

            dequeue();

            break;

        default:

            printf("\nInvalid choice");

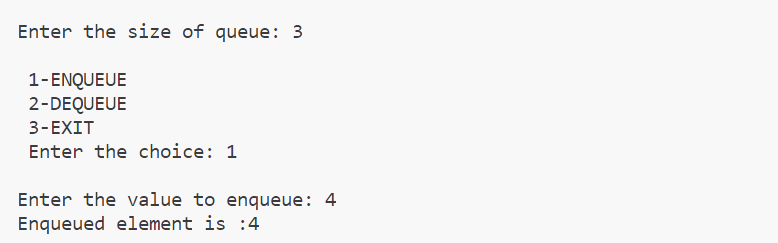
        }

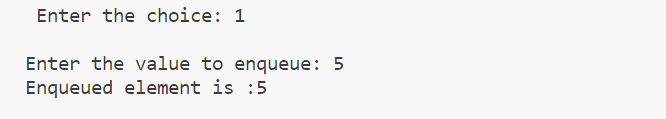
    }

    return 0;

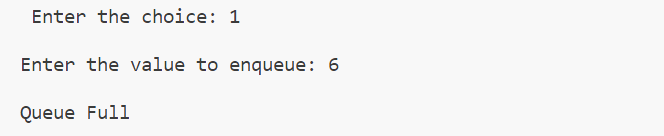
}

**Output:**











|  |
| --- |
| **Program 27** |
| **Implement priority queue.** |

**Source Code:**

#include <stdio.h>

#include <stdlib.h>

#define MAX 5

void insert\_by\_priority(int);

void delete\_by\_priority(int);

void create();

void check(int);

void display\_pqueue();

int pri\_que[MAX];

int front, rear;

int main()

{

    int n, ch;

    create();

    while (ch != 3)

    {

        printf("\nMENU\n=====\n1.Insert an element\n2.Display the queue\n3.Exit\n");

        printf("\nEnter your choice: ");

        scanf("%d", &ch);

        switch (ch)

        {

        case 1:

            printf("Enter the element to be inserted:");

            scanf("%d", &n);

            insert\_by\_priority(n);

            break;

        case 2:

            display\_pqueue();

            break;

        case 3:

            exit(0);

            break;

        default:

            printf("\nEnter valid choice:\n");

        }

    }

}

void create()

{

    front = rear = -1;

}

void insert\_by\_priority(int data)

{

    if (rear >= MAX - 1)

    {

        printf("\nQueue overflow no more elements can be inserted");

        return;

    }

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

    {

        front++;

        rear++;

        pri\_que[rear] = data;

        return;

    }

    else

        check(data);

    rear++;

}

void check(int data)

{

    int i, j;

    for (i = 0; i <= rear; i++)

    {

        if (data >= pri\_que[i])

        {

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

            {

                pri\_que[j] = pri\_que[j - 1];

            }

            pri\_que[i] = data;

            return;

        }

    }

    pri\_que[i] = data;

}

void delete\_by\_priority(int data)

{

    int i;

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

    {

        printf("\nQueue is empty no elements to delete");

        return;

    }

    for (i = 0; i <= rear; i++)

    {

        if (data == pri\_que[i])

        {

            for (; i < rear; i++)

            {

                pri\_que[i] = pri\_que[i + 1];

            }

            pri\_que[i] = -99;

            rear--;

            if (rear == -1)

                front = -1;

            return;

        }

    }

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

}

void display\_pqueue()

{

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

    {

        printf("\nQueue is empty");

        return;

    }

    for (; front <= rear; front++)

    {

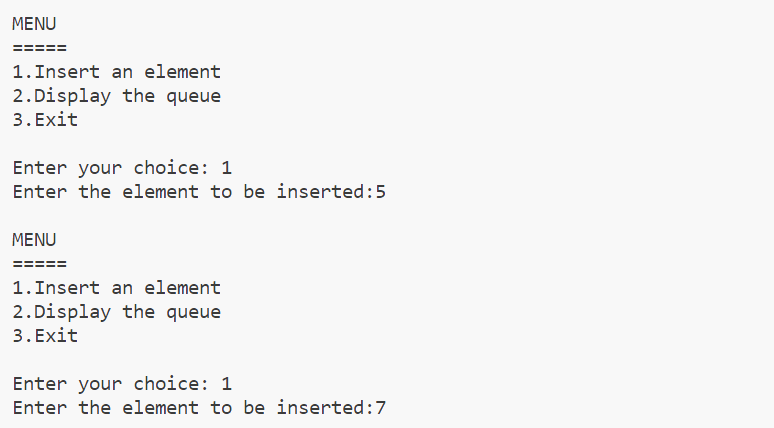
        printf(" %d ", pri\_que[front]);

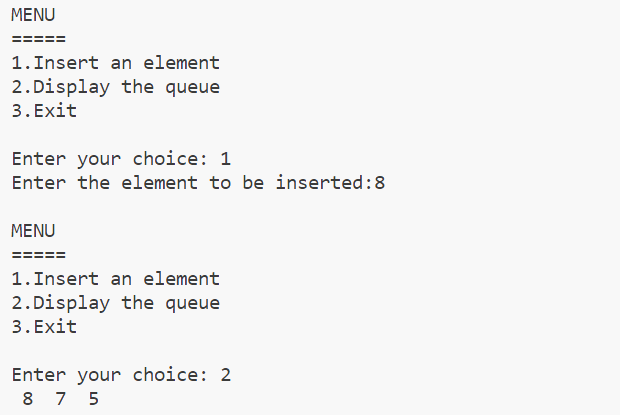
    }

    front = 0;

}

**Output:**

****

****

|  |
| --- |
| **Program 28** |
| **Demonstrate a linked list creation and display.** |

**Source Code:**

#include <stdio.h>

#include <malloc.h>

struct node

{

    int data;

    struct node \*next;

};

struct node \*head = NULL;

void insert(int e)

{

    struct node \*t;

    if (head == NULL)

    {

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

        head->data = e;

        head->next = NULL;

    }

    else

    {

        t = head;

        while (t->next != NULL)

        {

            t = t->next;

        }

        t->next = (struct node \*)malloc(sizeof(struct node));

        t->next->data = e;

        t->next->next = NULL;

    }

    printf("\n%d is inserted", e);

}

void disp()

{

    struct node \*t;

    if (head == NULL)

    {

        printf("Linked List Is Empty");

    }

    else

    {

        t = head;

        printf("\nLinked list elements are:\n");

        while (t != NULL)

        {

            printf("%d\t", t->data);

            t = t->next;

        }

        printf("\n");

    }

}

int main()

{

disp();

    insert(10);

    insert(20);

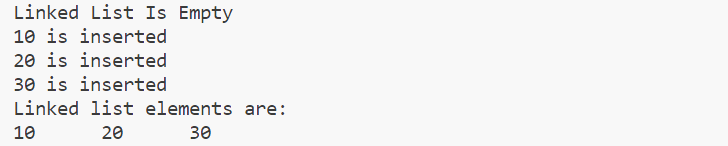
    insert(30);

    disp();

    return 0;

}

**Output:**



|  |
| --- |
| **Program 29** |
| **Write a program with functions  to insert a new node**   * 1. **At the beginning of a Singly Linked List.**   2. **At the end of the linked list**   3. **After a specified element in a linked list.** |

**Source Code:**

#include <stdio.h>

#include <stdlib.h>

struct node

{

    int data;

    struct node \*next;

};

struct node \*head = NULL;

void atend(int e)

{

    if (head == NULL)

    {

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

        head->data = e;

        head->next = NULL; // Initialize next to NULL

    }

    else

    {

        struct node \*t = head;

        while (t->next != NULL)

        {

            t = t->next;

        }

        t->next = (struct node \*)malloc(sizeof(struct node));

        t->next->data = e;

        t->next->next = NULL;

    }

}

void atbegin(int e)

{

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

    if (newnode == NULL)

    {

        printf("Memory allocation failed\n");

        exit(1);

    }

    newnode->data = e;

    newnode->next = head;

    head = newnode;

}

void afterelement(int e, int n)

{

    struct node \*t, \*a;

    t = head;

    while ((t->next != NULL) && (t->data != n))

    {

        t = t->next;

    }

    if ((t->next == NULL) && (t->data != n))

    {

        printf("element not found");

    }

    else

    {

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

        a->data = e;

        a->next = t->next;

        t->next = a;

    }

}

void disp()

{

    struct node \*t = head;

    if (head == NULL)

    {

        printf("Linked list is empty\n");

    }

    else

    {

        while (t != NULL)

        {

            printf("%d\t", t->data);

            t = t->next;

        }

        printf("\n");

    }

}

int menu()

{

    int ch;

    printf("\n 1-At Begining \n 2-At End \n 3-After Element \n 4-Display \n 5-Exit \n Enter your choice: ");

    scanf("%d", &ch);

    return ch;

}

int elemnt()

{

    int n;

    printf("Enter the element: ");

    scanf("%d", &n);

    return n;

}

int main()

{

    int ch;

    int n, m;

    for (ch = menu(); ch != 5; ch = menu())

    {

        switch (ch)

        {

        case 1:

            n = elemnt();

            atbegin(n);

            break;

        case 2:

            n = elemnt();

            atend(n);

            break;

        case 3:

            n = elemnt();

            printf("\nEnter the elemnt after which you want to enter new element: ");

            scanf("%d", &m);

            afterelement(n, m);

            break;

        case 4:

            disp();

            break;

        default:

            printf("invalid choice");

            break;

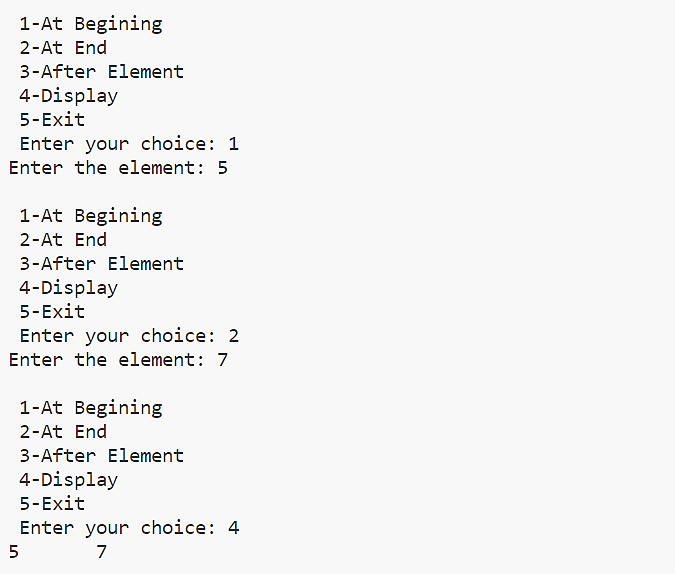
        }

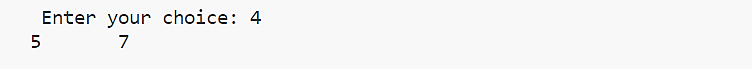
    }

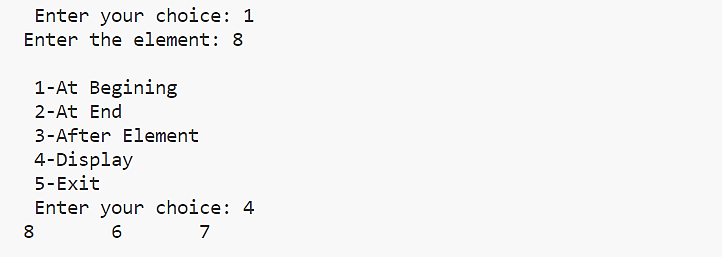
    return 0;

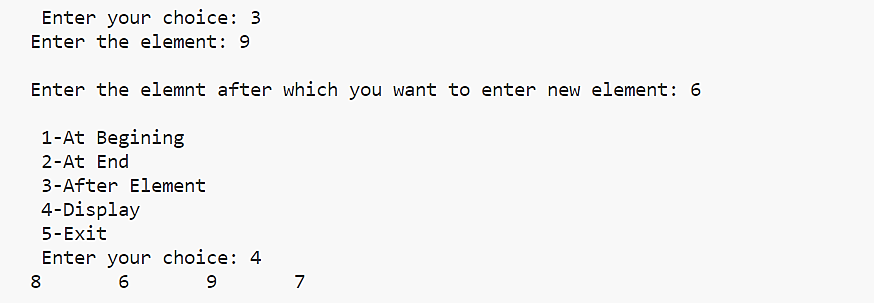
}

**Output:**









|  |
| --- |
| **Program 30** |
| **Write a program with functions to delete a node**   * 1. **From the beginning of the linked list**   2. **From the end of the linked list**   3. **The node with specified data element** |

**Source Code:**

#include<stdio.h>

#include<stdlib.h>

#include<process.h>

struct node{

int info;

struct node \*next;

};

struct node \*first=NULL,\*last=NULL;

void create(){

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

int n;

printf("\n Enter value:");

scanf("%d",&n);

temp->info=n;

temp->next=NULL;

if(first==NULL){

first=temp;

last=first;

}

else{

last->next=temp;

last=temp;

}

}

void delet(){

struct node \*prev=NULL,\*cur=NULL,\*t;

int count=1,pos,ch;

printf("\n 1.DELETE AT 1ST NODE\n 2.DELETE AT LAST NODE\n 3.DELETE AN

SPECIFIC ELEMENT\n CHOOSE YOUR OPTION:");

scanf("%d",&ch);

switch(ch){

case 1:if(first!=NULL){

printf("\n deleted element:%d",first->info);

first=first->next;

}

else

printf("\n not possible");

break;

case 2:t=first;

while(t->next->next!=NULL){

t=t->next;

}

t->next=NULL;

printf("\nNode Deleted ");

break;

case 3: t=first;

int n;

printf("\nEnter the data to be deleted:");

scanf("%d",&n);

while(t->next!=NULL && t->next->info!=n){

t=t->next;

}

if(t->next==NULL){

printf("\nelement not found not found!");

}

else{

t->next=t->next->next;

}

printf("\n Node Deleted");

break;

}

}

void display()

{

struct node \*t = first;

if(t == NULL)

{

printf("List is Empty\n");

}

else{

printf("\n Elements:\t");

while (t != NULL)

{

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

t = t->next;

}

printf("\n\n");

}

}

void main(){

int e,c;

do{

printf("\n 1.Create\n 2.Delete\n 3.Display\n 4.Exit\n Choose your option:");

scanf("%d",&c);

switch(c){

case 1: create();

break;

case 2: delet();

break;

case 3:display();

break;

case 4:exit(1);

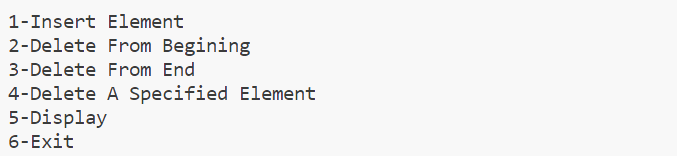
break;

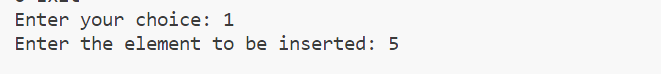
}

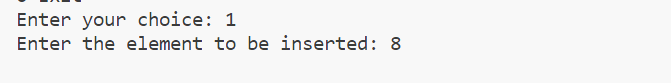
}while(1);

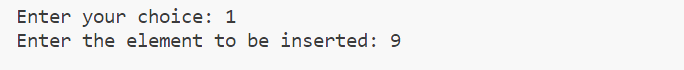
}

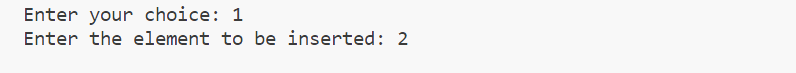
**Output:**



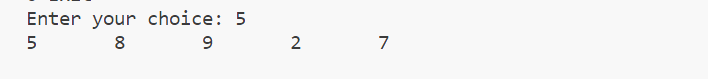




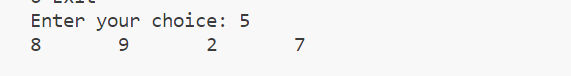


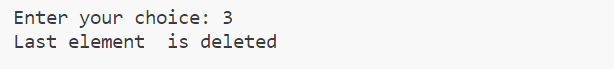


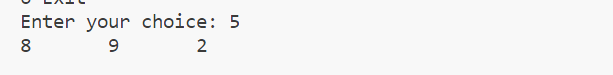


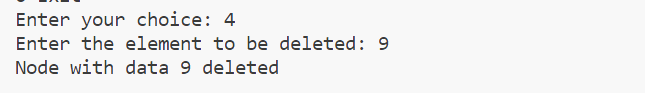














|  |
| --- |
| **Program 31** |
| **Write a program to create a singly linked list of n nodes and display it in reverse order.** |

**Source Code:**

#include <stdio.h>

#include <malloc.h>

struct node

{

    int data;

    struct node \*next;

};

struct node \*head = NULL;

void insert(int e)

{

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

    newnode->data = e;

    newnode->next = NULL;

    if (head == NULL)

    {

        head = newnode;

    }

    else

    {

        struct node \*t;

        t = head;

        while (t->next != NULL)

        {

            t = t->next;

        }

        t->next = newnode;

    }

    printf("\n%d is inserted", newnode->data);

}

void reverse()

{

    struct node \*t = head;

    struct node \*prev = NULL, \*next = NULL;

    while (t != NULL)

    {

        next = t->next;

        t->next = prev;

        prev = t;

        t = next;

    }

    head = prev;

    printf("\nElements Are Reversed");

}

void display()

{

    if (head == NULL)

    {

        printf("\nLinked List Is Empty");

    }

    else

    {

        struct node \*t = head;

        while (t != NULL)

        {

            printf("%d\t", t->data);

            t = t->next;

        }

        printf("\n");

    }

}

int menu()

{

    int ch;

    printf("\nEnter Your Choice \n1-Insert \n2-Display \n3-Reverse \n4-Exit\n");

    scanf("%d", &ch);

    return ch;

}

int main()

{

    int ch, e;

    for (ch = menu(); ch != 4; ch = menu())

    {

        switch (ch)

        {

        case 1:

            printf("\nEnter The Element To Insert: ");

            scanf("%d", &e);

            insert(e);

            break;

        case 2:

            display();

            break;

        case 3:

            reverse();

            break;

        default:

            printf("Invalid choice!");

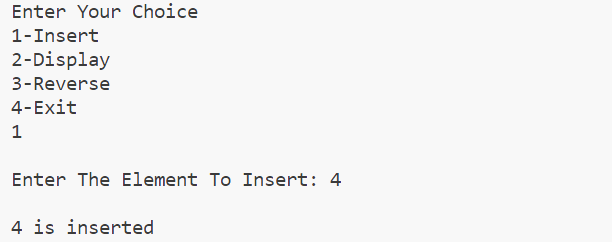
            break;

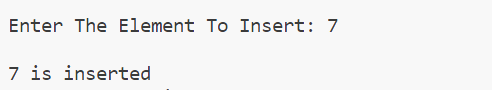
        }

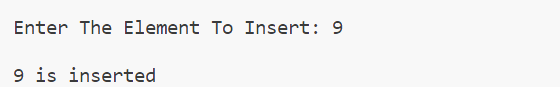
    }

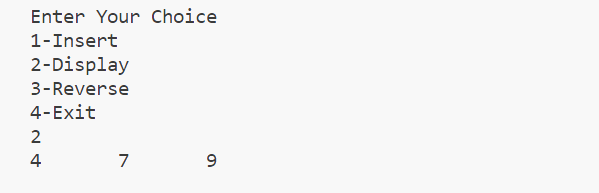
}

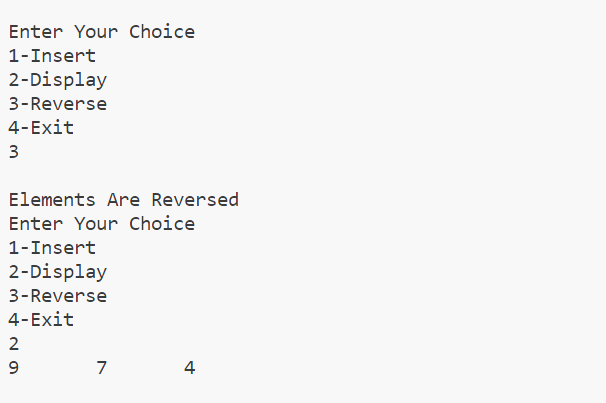
**Output:**

****









|  |
| --- |
| **Program 32** |
| **Sort the elements in a linked list using**   * 1. **changing the values (swapping the values)**   2. **Changing the address (Swapping the address).** |

**Source Code:**

#include <stdio.h>

#include <stdlib.h>

struct node

{

    int data;

    struct node \*next;

};

struct node \*ptr = NULL;

struct node \*head = NULL;

void insert(int val)

{

    if (head == NULL)

    {

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

        head->data = val;

        head->next = NULL;

    }

    else

    {

        ptr = head;

        while (ptr->next != NULL)

        {

            ptr = ptr->next;

        }

        ptr->next = (struct node \*)malloc(sizeof(struct node));

        ptr->next->data = val;

        ptr->next->next = NULL;

    }

}

void sortList()

{

    struct node \*current = head, \*index = NULL;

    int temp;

    if (head == NULL)

    {

        return;

    }

    else

    {

        while (current != NULL)

        {

            index = current->next;

            while (index != NULL)

            {

                if (current->data > index->data)

                {

                    temp = current->data;

                    current->data = index->data;

                    index->data = temp;

                }

                index = index->next;

            }

            current = current->next;

        }

    }

}

void Sortaddress()

{

    int swapp, i;

    struct node \*ptr1;

    struct node \*lptr = NULL;

    if (head == NULL)

        return;

    do

    {

        swapp = 0;

        ptr1 = head;

        while (ptr1->next != lptr)

        {

            if (ptr1->data > ptr1->next->data)

            {

                swap(ptr1, ptr1->next);

                swapp = 1;

            }

            ptr1 = ptr1->next;

        }

        lptr = ptr1;

    } while (swapp);

}

void swap(struct node \*a, struct node \*b)

{

    int temp = a->data;

    a->data = b->data;

    b->data = temp;

}

void display()

{

    struct node \*ptr = head;

    while (ptr != NULL)

    {

        printf("%d\t", ptr->data);

        ptr = ptr->next;

    }

    printf("\n");

}

void main()

{

    int ch, val;

    while (1)

    {

        printf("\n1.Insert.\n2.Display linked list\n3.Sort by swapping values\n4.Sort by swapping address\n5.Exit\nEnter your choice:");

        scanf("%d", &ch);

        switch (ch)

        {

        case 1:

            printf("Enter the number to insert :");

            scanf("%d", &val);

            insert(val);

            break;

        case 2:

            display();

            break;

        case 3:

            sortList();

            printf("\nSorted by swapping values");

            break;

        case 4:

            Sortaddress();

            printf("\nSorted by swapping address");

            break;

        case 5:

            exit(0);

            break;

        default:

        {

            printf("Wrong choice\n");

            break;

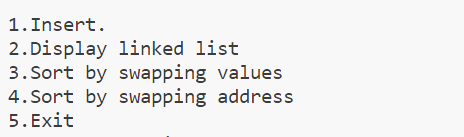
        }

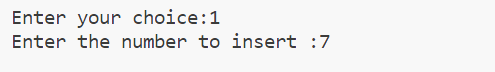
        }

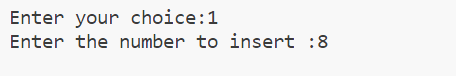
    }

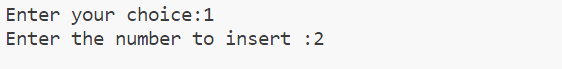
}

**Output:**

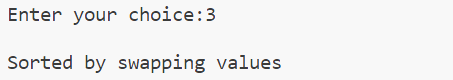


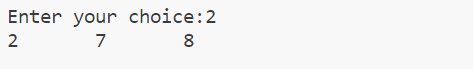


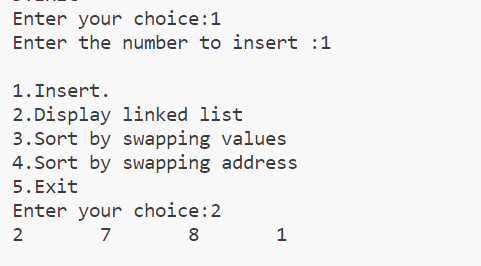


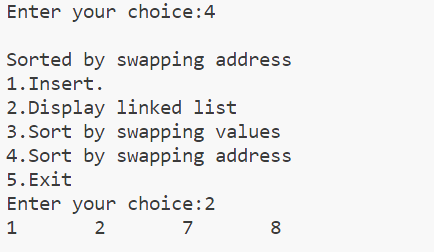












|  |
| --- |
| **Program 33** |
| **Polynomial using linked list - addition and multiplication.** |

**Source Code:**

#include <stdio.h>

#include <stdlib.h>

struct node

{

    float coeff;

    int expo;

    struct node \*next;

};

typedef struct node node;

struct node \*insert(struct node \*head, float co, int ex)

{

    struct node \*temp;

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

    newP->coeff = co;

    newP->expo = ex;

    newP->next = NULL;

    if (head == NULL || ex > head->expo)

    {

        newP->next = head;

        head = newP;

    }

    else

    {

        temp = head;

        while (temp->next != NULL && temp->next->expo >= ex)

            temp = temp->next;

        newP->next = temp->next;

        temp->next = newP;

    }

    return head;

}

struct node \*create(struct node \*head)

{

    int n, i;

    float coeff;

    int expo;

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

    scanf("%d", &n);

    for (i = 0; i < n; i++)

    {

        printf("Enter the coefficient for term %d: ", i + 1);

        scanf("%f", &coeff);

        printf("Enter the exponent for term %d: ", i + 1);

        scanf("%d", &expo);

        head = insert(head, coeff, expo);

    }

    return head;

}

void disp(struct node \*head)

{

    if (head == NULL)

        printf("No Polynomial.");

    else

    {

        struct node \*temp = head;

        while (temp != NULL)

        {

            printf("(%.0fx^%d)", temp->coeff, temp->expo);

            temp = temp->next;

            if (temp != NULL)

                printf(" + ");

            else

                printf("\n");

        }

    }

}

void polyAdd(struct node \*head1, struct node \*head2)

{

    struct node \*ptr1 = head1;

    struct node \*ptr2 = head2;

    struct node \*sum = NULL;

    while (ptr1 != NULL && ptr2 != NULL)

    {

        if (ptr1->expo == ptr2->expo)

        {

            sum = insert(sum, ptr1->coeff + ptr2->coeff, ptr1->expo);

            ptr1 = ptr1->next;

            ptr2 = ptr2->next;

        }

        else if (ptr1->expo > ptr2->expo)

        {

            sum = insert(sum, ptr1->coeff, ptr1->expo);

            ptr1 = ptr1->next;

        }

        else if (ptr1->expo < ptr2->expo)

        {

            sum = insert(sum, ptr2->coeff, ptr2->expo);

            ptr2 = ptr2->next;

        }

    }

    while (ptr1 != NULL)

    {

        sum = insert(sum, ptr1->coeff, ptr1->expo);

        ptr1 = ptr1->next;

    }

    while (ptr2 != NULL)

    {

        sum = insert(sum, ptr2->coeff, ptr2->expo);

        ptr2 = ptr2->next;

    }

    printf("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

    printf("\nAdded polynomial");

    printf("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

    disp(sum);

}

node \*polyMult(node \*head1, node \*head2, node \*pro)

{

    node \*ptr1 = head1;

    node \*ptr2 = head2;

    // Check if first or second polynomial is NULL

    if (head1 == NULL || head2 == NULL)

    {

        printf("\nNo polynomial\n");

        return;

    }

    // Multiplication of two polynomials

    while (ptr1 != NULL)

    {

        while (ptr2 != NULL)

        {

            float coeffPro = ptr1->coeff \* ptr2->coeff;

            int expoSum = ptr1->expo + ptr2->expo;

            pro = insert(pro, coeffPro, expoSum);

            ptr2 = ptr2->next;

        }

        ptr1 = ptr1->next;

        ptr2 = head2;

    }

    return pro;

}

node \*addLikeTerms(node \*pro, node \*res)

{

    node \*temp1, \*temp2;

    temp1 = pro;

    while (temp1->next != NULL)

    {

        temp2 = temp1->next;

        while (temp2 != NULL)

        {

            if (temp1->expo == temp2->expo)

            {

                float coeffSum = temp1->coeff + temp2->coeff;

                res = insert(res, coeffSum, temp1->expo);

                temp1 = temp1->next;

                break;

            }

            else

            {

                res = insert(res, temp1->coeff, temp1->expo);

                break;

            }

            temp2 = temp2->next;

        }

        temp1 = temp1->next;

    }

    res = insert(res, temp1->coeff, temp1->expo);

    return res;

}

void main()

{

    struct node \*head1 = NULL;

    struct node \*head2 = NULL;

    node \*pro = NULL;

    node \*res = NULL;

    printf("\nEnter the First polynomial");

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

    head1 = create(head1);

    printf("\nEnter the second polynomial");

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

    head2 = create(head2);

    printf("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

    printf("\n First polynomial");

    printf("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

    disp(head1);

    printf("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

    printf("\n Second polynomial");

    printf("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

    disp(head2);

    polyAdd(head1, head2);

    pro = polyMult(head1, head2, pro);

    res = addLikeTerms(pro, res);

    printf("\n\*\*\*\*\*\*\*\*");

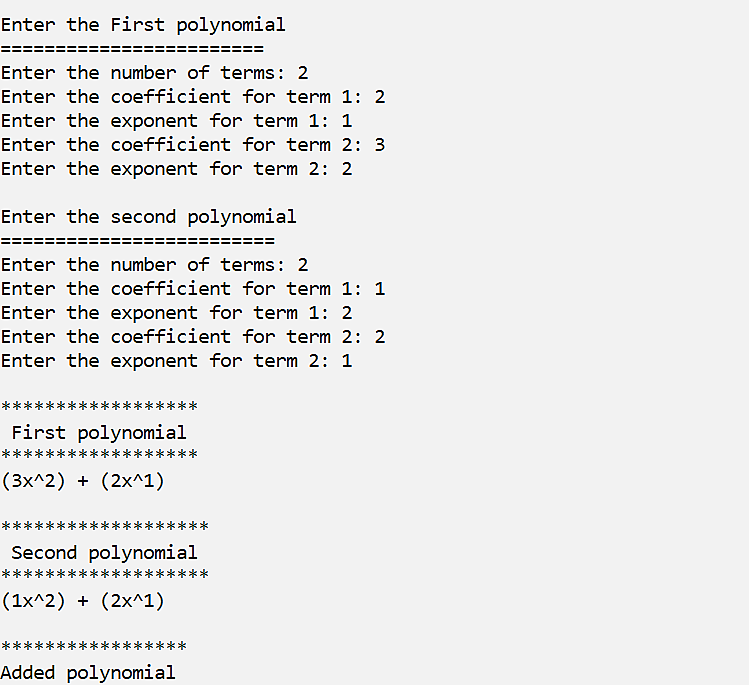
    printf("\nProduct");

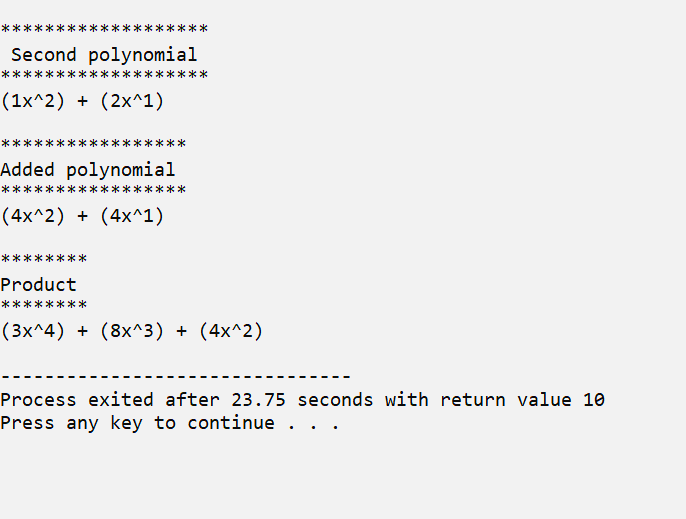
    printf("\n\*\*\*\*\*\*\*\*\n");

    disp(res);

}

**Output:**





|  |
| --- |
| **Program 34** |
| **Linked list using names - insert, delete, display, sort, reverse, count.** |

**Source Code:**

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

struct node

{

    char data[20];

    struct node \*next;

};

typedef struct node node;

node \*head = NULL;

void insert(char e[])

{

    if (head == NULL)

    {

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

        strcpy(head->data, e);

        head->next = NULL;

    }

    else

    {

        node \*t = head;

        while (t->next != NULL)

        {

            t = t->next;

        }

        t->next = (node \*)malloc(sizeof(node));

        strcpy(t->next->data, e);

        t->next->next = NULL;

    }

}

void delete\_elem(char e[])

{

    node \*t;

    int f = 0;

    if (head == NULL)

    {

        printf("\nList is Empty\n");

    }

    else if (strcmp(head->data, e) == 0)

    {

        head = head->next;

    }

    else

    {

        t = head;

        while (t->next != NULL)

        {

            if (strcmp(t->next->data, e) == 0)

            {

                t->next = t->next->next;

                f = 1;

                break;

            }

            t = t->next;

        }

        if (f == 0)

        {

            printf("\n%s not found\n", e);

        }

    }

}

void display()

{

    node \*t = head;

    if (t == NULL)

    {

        printf("\nList is Empty\n");

        return;

    }

    printf("\nNames in the linked list are:");

    while (t != NULL)

    {

        printf("\t%s", t->data);

        t = t->next;

    }

    printf("\n\n");

}

void sort()

{

    node \*temp1 = head, \*temp2;

    char elem[20];

    if (head == NULL)

    {

        printf("\nList is empty\n");

        return;

    }

    while (temp1 != NULL)

    {

        temp2 = temp1->next;

        while (temp2 != NULL)

        {

            if (strcmp(temp1->data, temp2->data) > 0)

            {

                strcpy(elem, temp1->data);

                strcpy(temp1->data, temp2->data);

                strcpy(temp2->data, elem);

            }

            temp2 = temp2->next;

        }

        temp1 = temp1->next;

    }

    display();

}

void reverse(node \*tmp)

{

    if (tmp == NULL)

    {

        return;

    }

    else

    {

        reverse(tmp->next);

    }

    printf("\t%s", tmp->data);

}

void count()

{

    node \*tmp = head;

    int count = 0;

    while (tmp != NULL)

    {

        count++;

        tmp = tmp->next;

    }

    printf("\nNumber of elements in the linked list: %d", count);

}

int main()

{

    int ch;

    char e[20];

    do

    {

        printf("\nLinked list\n============\n 1.Insertion\n 2.Deletion \n 3.Display\n 4.Sort\n 5.Reverse \n 6.Count\n 7.Exit\n Enter your choice: ");

        scanf("%d", &ch);

        switch (ch)

        {

        case 1:

            fflush(stdin);

            printf("\nEnter the name to insert: ");

            gets(e);

            insert(e);

            break;

        case 2:

            fflush(stdin);

            printf("\nEnter the name you want to delete: ");

            gets(e);

            delete\_elem(e);

            break;

        case 3:

            display();

            break;

        case 4:

            sort();

            break;

        case 5:

            if (head == NULL)

                printf("\nList is empty\n");

            else

            {

                printf("\nNames in the linked list in reverse order:");

                reverse(head);

            }

            break;

        case 6:

            count();

            break;

        case 7:

            exit(0);

            break;

        default:

            printf("\nInvalid choice!\n");

            break;

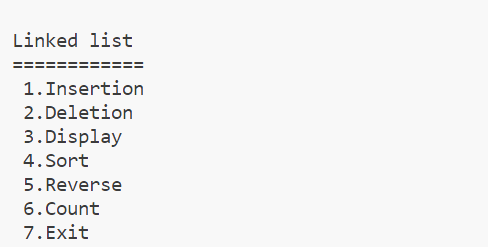
        }

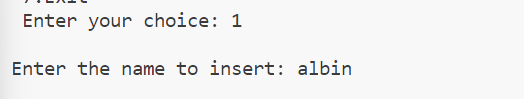
    } while (ch != 7);

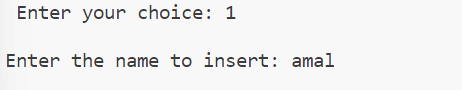
    return 0;

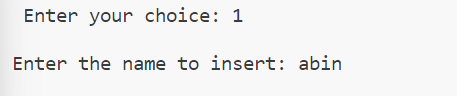
}

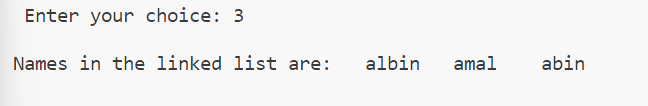
**Output:**

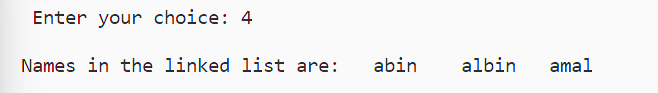
****

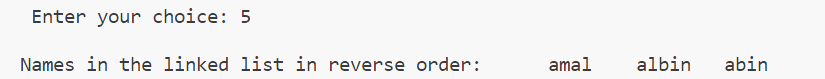
****

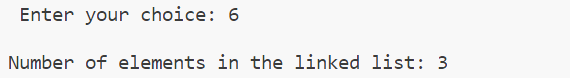
****

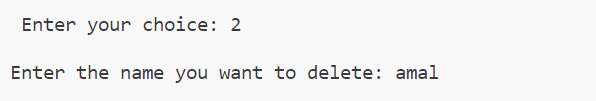
****

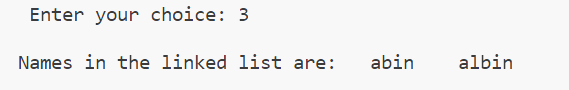
****











|  |
| --- |
| **Program 35** |
| **Linked Stack.** |

**Source Code:**

#include<stdio.h>

#include<malloc.h>

struct node{

    int data;

    struct node \*next;

}\*top=NULL;

void push(int e)

{

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

    if(new==NULL)

    {

        printf("stack underflow");

        return;

    }

    new->data=e;

    new->next=top;

    top=new;

    printf("%d is pushed onto the stack\n",e);

}

void pop()

{

    if(top==NULL)

    {

        printf("stack underflow");

    }

    else

    {

        struct node \*t = top;

        printf("%d is popped out",top->data);

        top=top->next;

    }

}

void peek()

{

    if(top==NULL)

    {

        printf("stack is empty");

    }

    printf("Element at top is %d\n",top->data);

}

int main()

{

    push(10);

    push(20);

    push(30);

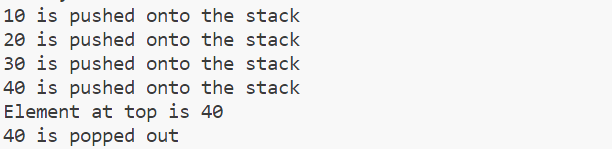
    push(40);

    peek();

    pop();

}

**Output:**



|  |
| --- |
| **Program 36** |
| **Linked Queue.** |

**Source Code:**

#include<stdio.h>

#include<malloc.h>

struct node{

    int data;

    struct node \*next;

};

typedef struct node node;

node \*front,\*rear;

void enqueue(int e)

{

    node \*newnode;

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

    newnode->data=e;

    newnode->next=NULL;

    if(rear==NULL)

    {

        front=rear=newnode;

    }

    else{

        rear->next=newnode;

        rear=newnode;

    }

}

void dequeue()

{

    if(front==NULL)

    {

        printf("Queue is empty!");

    }

    node \*t=front;

    front=front->next;

    printf("\nRemoved element is %d",t->data);

    if(front==NULL);

    {

        rear=NULL;

    }

    free(t);

}

void display()

{

    if(front==NULL)

    {

        printf("\nQueue is empty!");

    }

    else{

        struct node \*t=front;

        printf("Queue Elements Are: ");

        while(t!=NULL)

        {

            printf("%d ",t->data);

            t=t->next;

        }

        printf("\n");

    }

}

int menu()

{

    int ch;

    printf("\n 1-Enqueue\n 2-Deque\n 3-Display\n 4-Exit\n Enter Your choice: ");

    scanf("%d",&ch);

    return ch;

}

int main()

{

    int ch,e;

    for(ch=menu();ch!=4;ch=menu())

    {

        switch (ch)

        {

        case 1:

            printf("Enter The Element: ");

            scanf("%d",&e);

            enqueue(e);

            break;

        case 2:

            dequeue();

            break;

        case 3:

            display();

            break;

        default:

            printf("\ninvalid choice!");

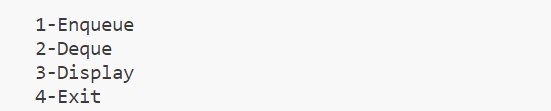
            break;

        }

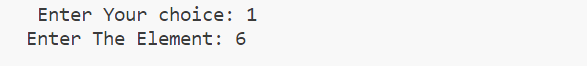
    }

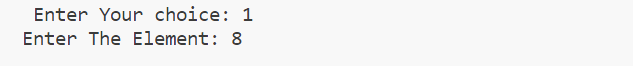
}

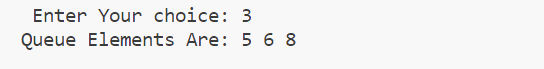
**Output:**

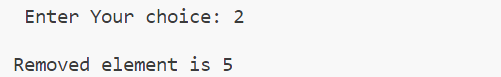
****

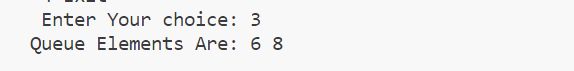
****

****

****

****

****

****

|  |
| --- |
| **Program 37** |
| **Circular Linked List** |

**Source Code:**

#include <stdio.h>

#include <stdlib.h>

struct node

{

  int data;

  struct node \*next;

};

typedef struct node clist;

clist \*head = NULL;

void insertion(int a)

{

  clist \*t;

  if (head == NULL)

  {

    head = (clist \*)malloc(sizeof(clist));

    head->data = a;

    head->next = head;

  }

  else

  {

    t = head;

    while (t->next != head)

    {

      t = t->next;

    }

    t->next = (clist \*)malloc(sizeof(clist));

    t->next->data = a;

    t->next->next = head;

  }

}

void disp()

{

  clist \*t;

  t = head;

  if (t == NULL)

  {

    printf("Empty C List");

  }

  else

  {

    do

    {

      printf("%d\t", t->data);

      t = t->next;

    } while (t != head);

  }

}

int main()

{

  int ch, e;

  printf("\n CIRCULAR LINKED LIST \n---------------------\n");

  do

  {

    printf("\n1.Insertion\n2.Display\n3.Exit\n Choose your option:");

    scanf("%d", &ch);

    switch (ch)

    {

    case 1:

      printf("\n enter element:");

      scanf("%d", &e);

      insertion(e);

      break;

    case 2:

      disp();

      break;

    case 3:

      exit(1);

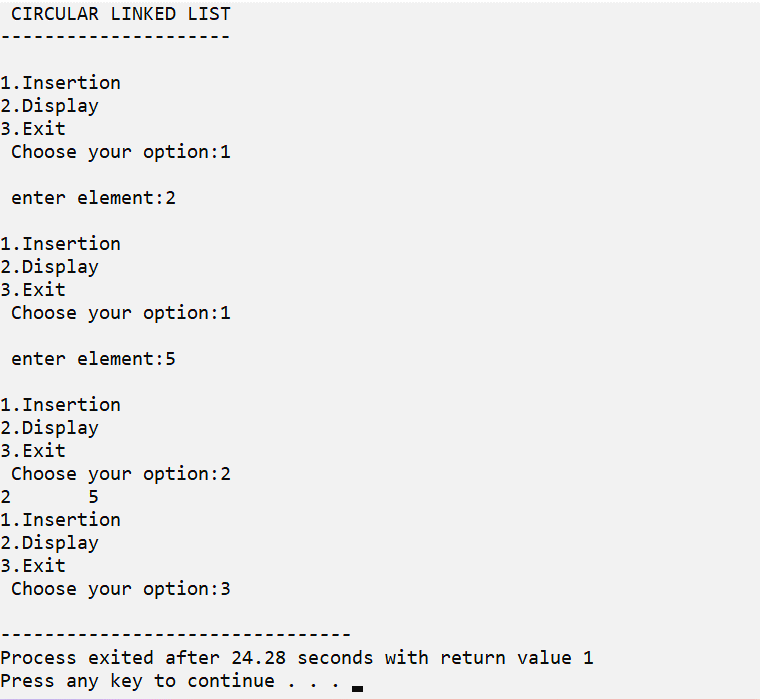
    }

  } while (1);

  return 0;

}

**Output:**



|  |
| --- |
| **Program 38** |
| **Circular Linked Queue.** |

**Source Code:**

#include <stdio.h>

#include <stdlib.h>

struct node

{

  int data;

  struct node \*next;

};

struct node \*f = NULL;

struct node \*r = NULL;

void enqueue(int d) // Insert elements in Queue

{

  struct node \*n;

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

  n->data = d;

  n->next = NULL;

  if ((r == NULL) && (f == NULL))

  {

    f = r = n;

    r->next = f;

  }

  else

  {

    r->next = n;

    r = n;

    n->next = f;

  }

}

void dequeue() // Delete an element from Queue

{

  struct node \*t;

  t = f;

  if ((f == NULL) && (r == NULL))

    printf("\nQueue is Empty");

  else if (f == r)

  {

    f = r = NULL;

    free(t);

  }

  else

  {

    f = f->next;

    r->next = f;

    free(t);

  }

}

void print()

{ // Print the elements of Queue

  struct node \*t;

  t = f;

  if ((f == NULL) && (r == NULL))

    printf("\nQueue is Empty");

  else

  {

    do

    {

      printf("\t%d", t->data);

      t = t->next;

    } while (t != f);

  }

}

int main()

{

  int opt, n, i, data;

  printf("CIRCULAR LINKED QUEUE\n----------------------\n");

  do

  {

    printf("\n1.Enqueue\n2.Display\n3.Deletion\n4.Exit\nChoose your option:");

    scanf("%d", &opt);

    switch (opt)

    {

    case 1:

      printf("\nEnter the number of data:");

      scanf("%d", &n);

      printf("\nEnter your data:");

      i = 0;

      while (i < n)

      {

        scanf("%d", &data);

        enqueue(data);

        i++;

      }

      break;

    case 2:

      print();

      break;

    case 3:

      dequeue();

      break;

    case 4:

      exit(1);

      break;

    default:

      printf("\nIncorrect Choice");

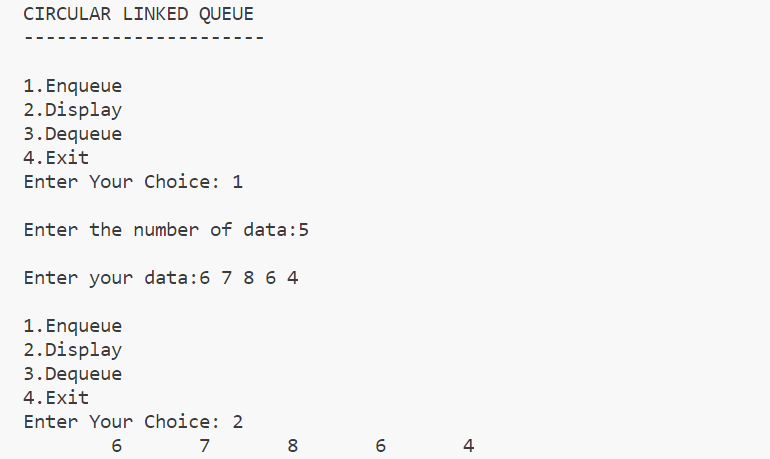
    }

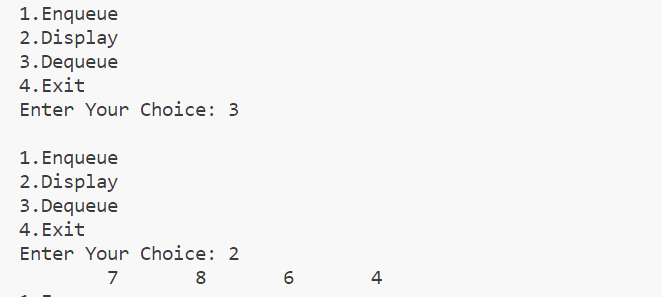
  } while (opt != 0);

  return 0;

}

**Output:**

****

****

|  |
| --- |
| **Program 39** |
| **Doubly Linked List** |

**Source Code:**

#include <stdio.h>

#include <stdlib.h>

struct node

{

    int data;

    struct node \*next;

    struct node \*prev;

};

typedef struct node dll;

dll \*head = NULL;

void insert(int a)

{

    dll \*t;

    if (head == NULL)

    {

        head = (dll \*)malloc(sizeof(dll));

        head->data = a;

        head->next = NULL;

        head->prev = NULL;

    }

    else

    {

        for (t = head; t->next != NULL; t = t->next)

            ;

        t->next = (dll \*)malloc(sizeof(dll));

        t->next->data = a;

        t->next->next = NULL;

        t->next->prev = t;

    }

    printf("\n%d is inserted\n",a);

}

void delete(int a)

{

    dll \*t;

    if (head == NULL)

        printf("D L L is empty");

    else if (head->data == a)

    {

        if (head->next == NULL)

            head = NULL;

        else

        {

            head = head->next;

            head->prev = NULL;

        }

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

    }

    else

    {

        for (t = head; t != NULL && t->data != a; t = t->next)

            ;

        if (t == NULL)

            printf("Element Not Found");

        else if (t->next == NULL)

            t->prev->next = NULL;

        else

        {

            t->next->prev = t->prev;

            t->prev->next = t->next;

        }

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

    }

}

void disp()

{

    dll \*t;

    for (t = head; t != NULL; t = t->next)

    {

        printf("%d\t", t->data);

    }

}

int main()

{

    int ch, e, n;

    printf("\n DOUBLY LINKED LIST\n-------------------\n");

    do

    {

        printf("\n1.Insertion\n2.Deletion\n3.Display\n4.exit\nchoose your option:");

        scanf("%d", &ch);

        switch (ch)

        {

        case 1:

            printf("\nEnter element:");

            scanf("%d", &e);

            insert(e);

            break;

        case 2:

            printf("\nEnter element to be deleted:");

            scanf("%d", &n);

            delete (n);

            break;

        case 3:

            disp();

            break;

        case 4:

            exit(1);

            break;

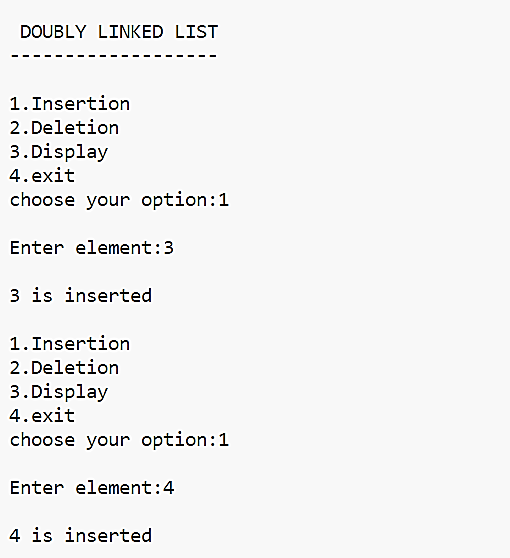
        }

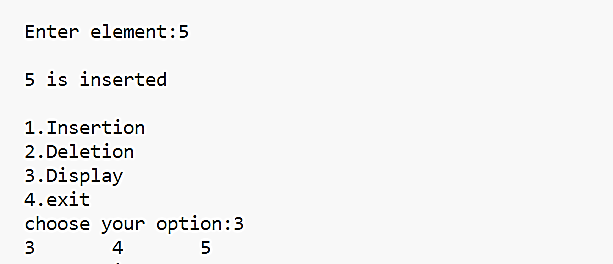
    } while (1);

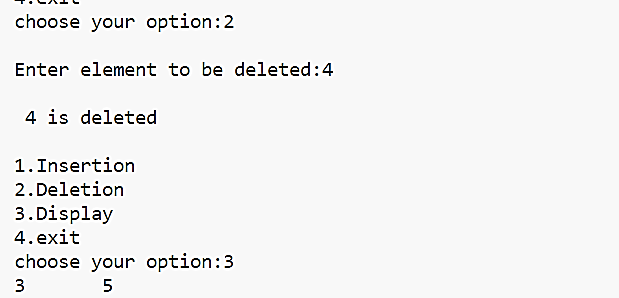
    return 0;

}

**Output:**

****

****

****

|  |
| --- |
| **Program 40** |
| **Circular Doubly Linked List.** |

**Source Code:**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

struct node {

    char data[100];

    struct node \*next, \*prev;

};

typedef struct node cdll;

cdll \*head = NULL;

void insert(char e[]) {

    cdll \*t;

    if (head == NULL) {

        head = (cdll \*)malloc(sizeof(cdll));

        strcpy(head->data, e);

        head->next = head;

        head->prev = head;

    } else {

        for (t = head; t->next != head; t = t->next);

        t->next = (cdll \*)malloc(sizeof(cdll));

        strcpy(t->next->data, e);

        t->next->next = head;

        t->next->prev = t;

        head->prev = t->next;

    }

}

void disp() {

    cdll \*t;

    if (head == NULL) {

        printf("Empty Linked List");

    } else {

        t = head;

        do {

            puts(t->data);

            t = t->next;

        } while (t != head);

    }

}

void delete(char e[]) {

    cdll \*t;

    if (head == NULL) {

        printf("Empty Linked List");

    } else if (strcmp(head->data, e) == 0 && head->next == head) {

        free(head);

        head = NULL;

    } else if (strcmp(head->data, e) == 0) {

        head->prev->next = head->next;

        head->next->prev = head->prev;

        cdll \*temp = head;

        head = head->next;

        free(temp);

    } else {

        t = head->next;

        while (t != head && strcmp(t->data, e) != 0) {

            t = t->next;

        }

        if (t == head) {

            printf("Not found\n");

        } else {

            t->next->prev = t->prev;

            t->prev->next = t->next;

            free(t);

        }

    }

}

int main() {

    char e[100];

    int ch;

    printf("\nCIRCULAR DOUBLY LINKED LIST");

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

    do {

        printf("\n1.Insert\n2.Display\n3.Delete\n4.Exit\nChoose your option:");

        scanf("%d", &ch);

        getchar();

        switch (ch) {

            case 1:

                printf("\nEnter name: ");

                fgets(e, sizeof(e), stdin);

                e[strcspn(e, "\n")] = '\0';

                insert(e);

                break;

            case 2:

                disp();

                break;

            case 3:

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

                fgets(e, sizeof(e), stdin);

                e[strcspn(e, "\n")] = '\0';

                delete(e);

                break;

            case 4:

                while (head != NULL) {

                    cdll \*temp = head->next;

                    free(head);

                    head = temp;

                }

                exit(0);

                break;

            default:

                printf("Invalid choice. Please choose a valid option.\n");

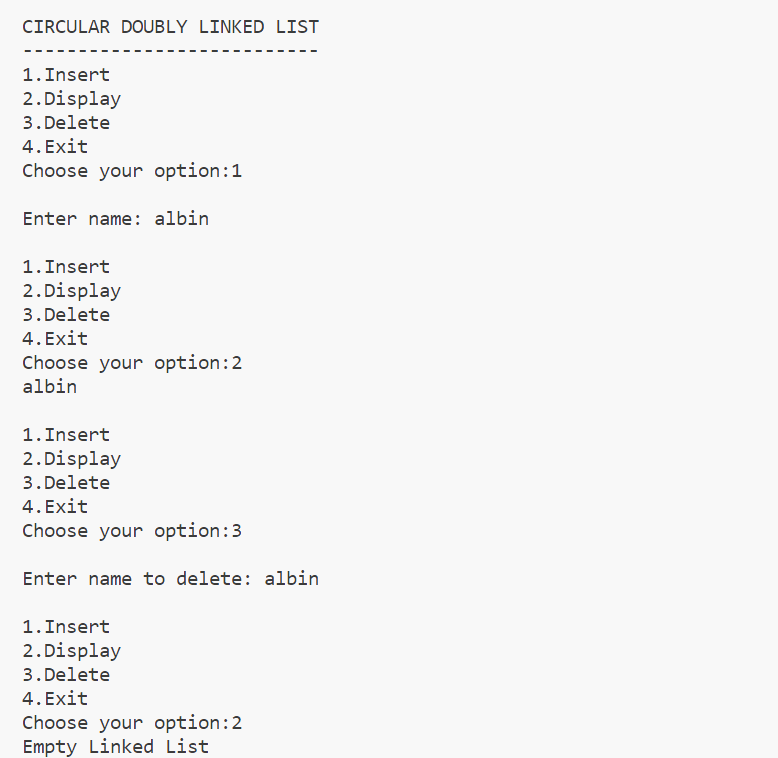
        }

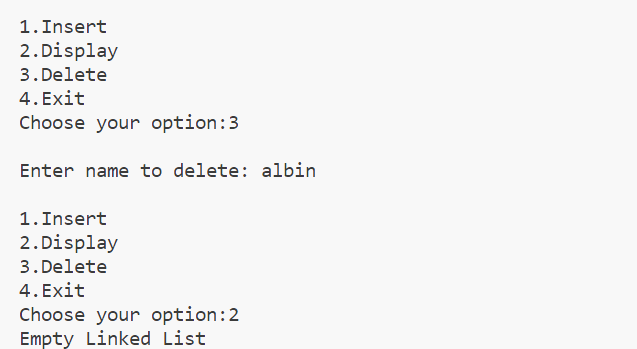
    } while (1);

    return 0;

}

**Output:**





|  |
| --- |
| **Program 41** |
| **Binary search tree insertion and display Traversal using inorder, preorder and postorder using recursion** |

**Source Code:**

#include <stdio.h>

#include <stdlib.h>

struct node

{

    int data;

    struct node \*left;

    struct node \*right;

};

typedef struct node tree;

tree \*root = NULL;

void insert(int e)

{

    tree \*p, \*x;

    if (root == NULL)

    {

        root = (tree \*)malloc(sizeof(tree));

        root->data = e;

        root->left = NULL;

        root->right = NULL;

    }

    else

    {

        p = root;

        while (p != NULL)

        {

            x = p;

            if (p->data > e)

                p = p->left;

            else

                p = p->right;

        }

        if (x->data > e)

        {

            x->left = (tree \*)malloc(sizeof(tree));

            x->left->data = e;

            x->left->left = NULL;

            x->left->right = NULL;

        }

        else

        {

            x->right = (tree \*)malloc(sizeof(tree));

            x->right->data = e;

            x->right->left = NULL;

            x->right->right = NULL;

        }

    }

}

void preorder(tree \*r)

{

    if (r != NULL)

    {

        printf("%d\t", r->data);

        preorder(r->left);

        preorder(r->right);

    }

}

void postorder(tree \*r)

{

    if (r != NULL)

    {

        postorder(r->left);

        postorder(r->right);

        printf("%d\t", r->data);

    }

}

void inorder(tree \*r)

{

    if (r != NULL)

    {

        inorder(r->left);

        printf("%d\t", r->data);

        inorder(r->right);

    }

}

int main()

{

    int ch, e;

    while (1)

    {

        printf("\n1-Insert\n2-Inorder\n3-preorder\n4-postorder\n5-exit\nEnter Your Choice: ");

        scanf("%d", &ch);

        {

            switch (ch)

            {

            case 1:

                printf("\nEnter the element:");

                scanf("%d", &e);

                insert(e);

                break;

            case 2:

                inorder(root);

                break;

            case 3:

                preorder(root);

                break;

            case 4:

                postorder(root);

                break;

            case 5:

                exit(0);

                break;

            default:

                printf("\nInvalid choice");

                break;

            }

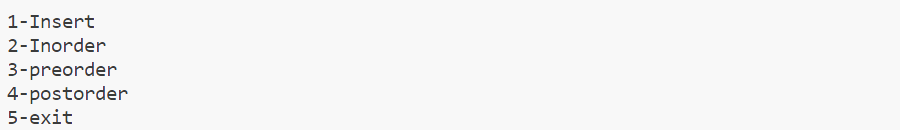
        }

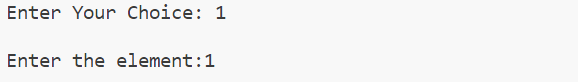
    }

    return 0;

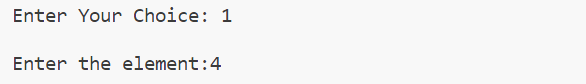
}

**Output:**

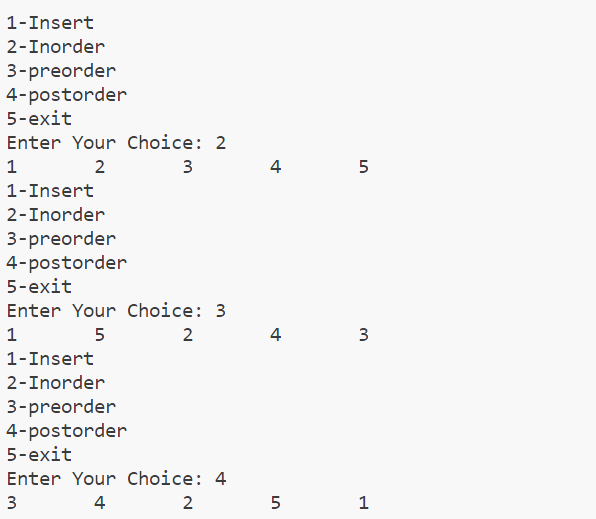
****

****

****

****

****

****

|  |
| --- |
| **Program 42** |
| **Binary search tree insertion and display in-order without using recursion.** |

**Source Code:**

#include <stdio.h>

#include <stdlib.h>

struct node

{

    int data;

    struct node \*left, \*right;

};

typedef struct node tree;

struct stack

{

    tree \*ptr;

    struct stack \*next;

};

typedef struct stack stack;

tree \*root = NULL;

stack \*top = NULL;

void push(tree \*t)

{

    stack \*temp = (stack \*)malloc(sizeof(stack)); // allocate new node

    temp->ptr = t;

    temp->next = top;

    top = temp;

}

tree \*pop()

{

    tree \*t = NULL;

    if (top != NULL)

    {

        t = top->ptr;

        top = top->next;

    }

    return t;

}

void inorderwor(tree \*r)

{

    tree \*t;

    for (t = r; t != NULL; t = t->left)

    {

        push(t);

    }

    t = pop();

    while (t != NULL)

    {

        printf("%d\t", t->data);

        if (t->right != NULL)

        {

            for (t = t->right; t != NULL; t = t->left)

            {

                push(t);

            }

        }

        t = pop();

    }

}

void insert(int e)

{

    tree \*p, \*x;

    if (root == NULL)

    {

        root = (tree \*)malloc(sizeof(tree));

        root->data = e;

        root->left = NULL;

        root->right = NULL;

    }

    else

    {

        x = root;

        while (x != NULL)

        {

            p = x;

            if (e < x->data)

            {

                x = x->left;

            }

            else if (e > x->data)

            {

                x = x->right;

            }

        }

        if (e < p->data)

        {

            p->left = (tree \*)malloc(sizeof(tree));

            p->left->data = e;

            p->left->left = NULL;

            p->left->right = NULL;

        }

        else if (e > p->data)

        {

            p->right = (tree \*)malloc(sizeof(tree));

            p->right->data = e;

            p->right->left = NULL;

            p->right->right = NULL;

        }

    }

}

int main()

{

    int ch, e, ch1;

    printf("\nBINARY SEARCH TREE\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

    do

    {

        printf("\n1.Insertion\n2.inorderdisplay\n3.Exit\nChoose your option:");

        scanf("%d", &ch);

        switch (ch)

        {

        case 1:

            printf("Enter no: ");

            scanf("%d", &e);

            insert(e);

            break;

        case 2:

            inorderwor(root);

            break;

        case 3:

            exit(1);

            break;

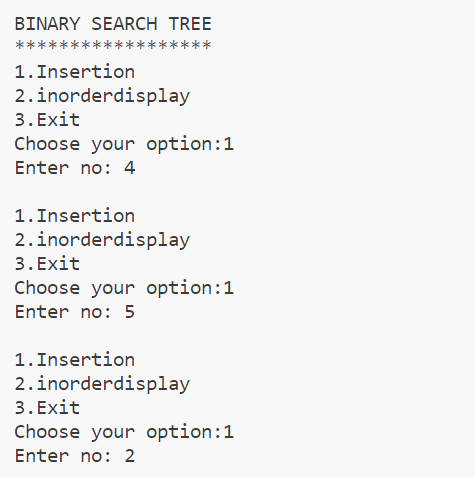
        }

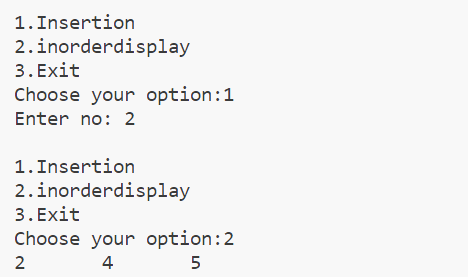
    } while (1);

    return 0;

}

**Output:**





|  |
| --- |
| **Program 43** |
| **Binary search tree insertion and display pre-order without using recursion.** |

**Source Code:**

#include <stdio.h>

#include <stdlib.h>

struct node

{

    int data;

    struct node \*left, \*right;

};

typedef struct node tree;

struct stack

{

    tree \*ptr;

    struct stack \*next;

};

typedef struct stack stack;

tree \*root = NULL;

stack \*top = NULL;

void push(tree \*t)

{

    stack \*temp = (stack \*)malloc(sizeof(stack));

    temp->ptr = t;

    temp->next = top;

    top = temp;

}

tree \*pop()

{

    tree \*t = NULL;

    if (top != NULL)

    {

        t = top->ptr;

        top = top->next;

    }

    return t;

}

void preorderwor(tree \*r)

{

    tree \*t;

    for (t = r; t != NULL; t = t->left)

    {

        printf("%d\t", t->data);

        push(t);

    }

    t = pop();

    while (t != NULL)

    {

        if (t->right != NULL)

        {

            for (t = t->right; t != NULL; t = t->left)

            {

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

                push(t);

            }

        }

        t = pop();

    }

}

void insert(int e)

{

    tree \*p, \*x;

    if (root == NULL)

    {

        root = (tree \*)malloc(sizeof(tree));

        root->data = e;

        root->left = NULL;

        root->right = NULL;

    }

    else

    {

        x = root;

        while (x != NULL)

        {

            p = x;

            if (e < x->data)

            {

                x = x->left;

            }

            else if (e > x->data)

            {

                x = x->right;

            }

        }

        if (e < p->data)

        {

            p->left = (tree \*)malloc(sizeof(tree));

            p->left->data = e;

            p->left->left = NULL;

            p->left->right = NULL;

        }

        else if (e > p->data)

        {

            p->right = (tree \*)malloc(sizeof(tree));

            p->right->data = e;

            p->right->left = NULL;

            p->right->right = NULL;

        }

    }

}

void main()

{

    int ch, e, ch1;

    printf("\nBINARY SEARCH TREE");

    do

    {

        printf("\n1.Insertion\n2.preorderdisplay\n3.Exit\nChoose your option:");

        scanf("%d", &ch);

        switch (ch)

        {

        case 1:

            printf("Enter no: ");

            scanf("%d", &e);

            insert(e);

            break;

        case 2:

            printf("\n");

            preorderwor(root);

            break;

        case 3:

            exit(1);

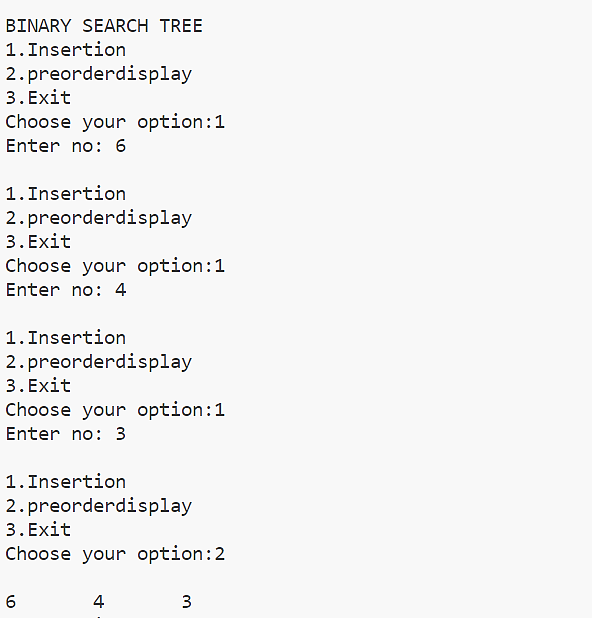
            break;

        }

    } while (1);

}

**Output:**

****

|  |
| --- |
| **Program 44** |
| **Binary search tree insertion and display post-order without using recursion.** |

**Source Code:**

#include <stdio.h>

#include <stdlib.h>

struct node {

    struct node \*left;

    int info;

    struct node \*right;

};

struct node\* insert(struct node\* root, int key) {

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

    newNode->info = key;

    newNode->left = newNode->right = NULL;

    if (root == NULL) {

        return newNode;

    }

    struct node\* current = root;

    struct node\* parent = NULL;

    while (current != NULL) {

        parent = current;

        if (key < current->info) {

            current = current->left;

        } else if (key > current->info) {

            current = current->right;

        } else {

            free(newNode);

            return root; // Duplicate key, no need to insert

        }

    }

    if (key < parent->info) {

        parent->left = newNode;

    } else {

        parent->right = newNode;

    }

    return root;

}

void postorder(struct node\* root) {

    if (root == NULL) {

        return;

    }

    struct node\* current = root;

    struct node\* temp = NULL;

    while (current != NULL) {

        if (current->right == NULL) {

            printf("%d ", current->info);

            current = current->left;

        } else {

            temp = current->right;

            while (temp->left != NULL && temp->left != current) {

                temp = temp->left;

            }

            if (temp->left == NULL) {

                temp->left = current;

                current = current->right;

            } else {

                temp->left = NULL;

                printf("%d ", current->info);

                current = current->left;

            }

        }

    }

}

int main() {

    struct node \*root = NULL;

    int choice, k;

    printf("\n BINARY SEARCH TREE\n--------------------");

    while (1) {

        printf("\n");

        printf("1.Insert\n2.Display Postorder\n3.Quit\n");

        printf("Enter your choice : ");

        scanf("%d", &choice);

        switch (choice) {

            case 1:

                printf("Enter value: ");

                scanf("%d", &k);

                root = insert(root, k);

                break;

            case 2:

                printf("Postorder traversal: ");

                postorder(root);

                printf("\n");

                break;

            case 3:

                exit(0);

            default:

                printf("Wrong choice\n");

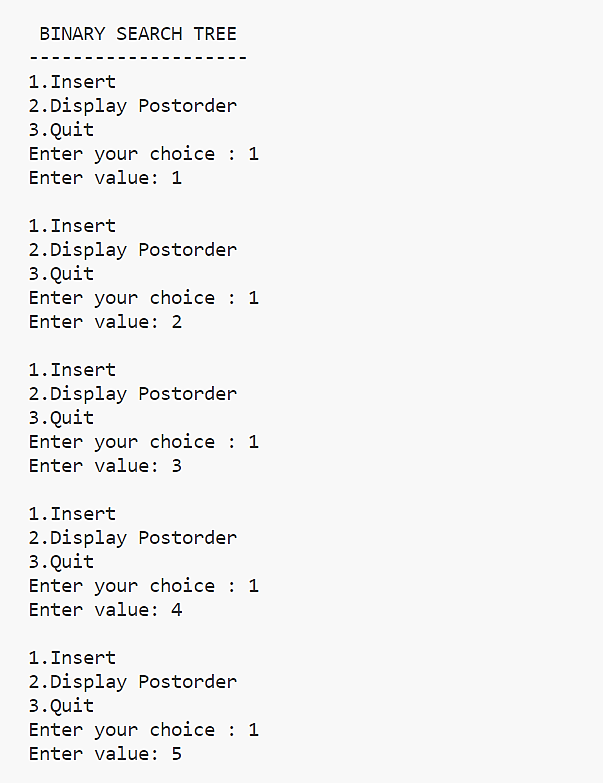
        }

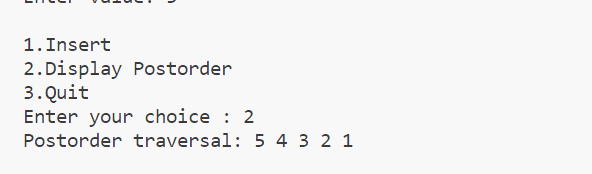
    }

    return 0;

}

**Output:**

****

****

|  |
| --- |
| **Program 45** |
| **Binary search tree insertion using names and display the names in ascending order using inorder traversal.** |

**Source Code:**

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include <process.h>

struct dictionary

{

    char name[20];

    struct dictionary \*left, \*right;

};

typedef struct dictionary dict;

dict \*root = NULL;

int check(char a[], char b[])

{

    int i, j, c;

    for (i = 0, j = 0; a[i] != '\0' && b[j] != '\0'; i++, j++)

    {

        if (a[i] > b[j])

        {

            c = 1;

            break;

        }

        else if (b[j] > a[i])

        {

            c = -1;

            break;

        }

        else

            c = 0;

    }

    if (c == 1)

        return 1;

    else if (c == -1)

        return -1;

    else

        return 0;

}

void insert(dict \*temp)

{

    int flag = 0;

    dict \*ptr, \*p;

    ptr = root;

    if (root == NULL)

        root = temp;

    else

    {

        while (ptr != NULL)

        {

            if (check(temp->name, ptr->name) > 0)

            {

                p = ptr;

                ptr = ptr->right;

            }

            else if (check(temp->name, ptr->name) < 0)

            {

                p = ptr;

                ptr = ptr->left;

            }

            else if (check(temp->name, ptr->name) == 0)

            {

                flag = 1;

                printf("\nName exists!!!!");

                break;

            }

        }

        if (flag == 0 && ptr == NULL)

        {

            if (check(p->name, temp->name) == 1)

                p->left = temp;

            else if (check(p->name, temp->name) == -1)

                p->right = temp;

        }

    }

}

void disp(dict \*root)

{

    if (root != NULL)

    {

        disp(root->left);

        printf("%s  ", root->name);

        disp(root->right);

    }

}

void main()

{

    dict \*t;

    int ch;

    char w1[20];

    printf("\nBINARY SEARCH TREE USING STRING");

    do

    {

        printf("\n1.Insert\n2.Display\n3.Exit\nEnter your choice: ");

        scanf("%d", &ch);

        switch (ch)

        {

        case 1:

            t = (dict \*)malloc(sizeof(dict));

            t->left = NULL;

            t->right = NULL;

            printf("Enter name: ");

            scanf("%s", t->name);

            insert(t);

            break;

        case 2:

            printf("\nNames:");

            disp(root);

            printf("\n");

            break;

        case 3:

            exit(1);

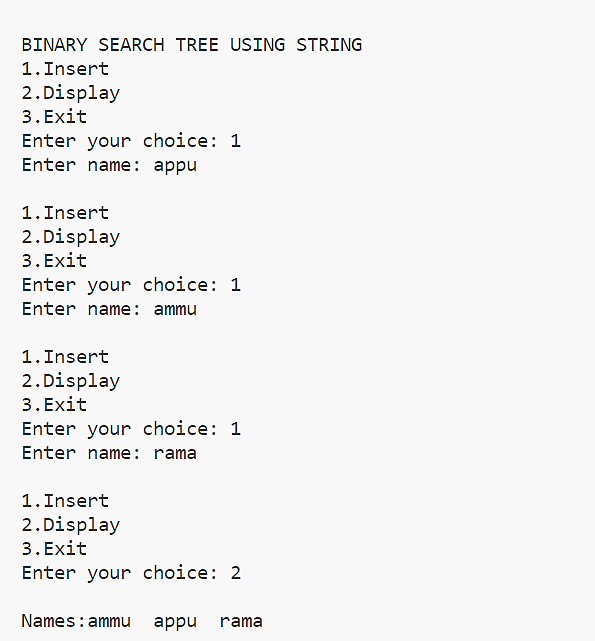
            break;

        }

    } while (1);

}

**Output:**

****



|  |
| --- |
| **Program 46** |
| **Demonstrate the data structure of adjacent matrix using arrays.** |

**Source Code:**

#include <stdio.h>

#define V 4

// Initialize the matrix to zero

void init(int arr[][V])

{

    int i, j;

    for (i = 0; i < V; i++)

        for (j = 0; j < V; j++)

            arr[i][j] = 0;

}

void addEdge(int arr[][V], int i, int j)

{

    arr[i][j] = 1;

    arr[j][i] = 1;

}

void printAdjMatrix(int arr[][V])

{

    int i, j;

    for (i = 0; i < V; i++)

    {

        printf("%d: ", i);

        for (j = 0; j < V; j++)

        {

            printf("%d ", arr[i][j]);

        }

        printf("\n");

    }

}

int main()

{

    int adjMatrix[V][V];

    init(adjMatrix);

    addEdge(adjMatrix, 0, 1);

    addEdge(adjMatrix, 0, 2);

    addEdge(adjMatrix, 1, 2);

    addEdge(adjMatrix, 2, 0);

    addEdge(adjMatrix, 2, 3);

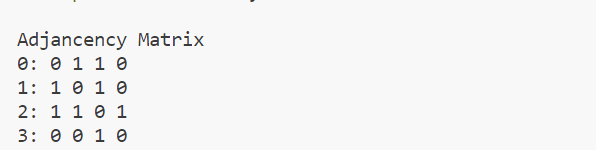
    printf("\nAdjancency Matrix\n");

    printAdjMatrix(adjMatrix);

    return 0;

}

**Output:**



|  |
| --- |
| **Program 47** |
| **Demonstrate the data structure of adjacent matrix  using linked lists.** |

**Source Code:**

#include <stdio.h>

#include <stdlib.h>

struct node

{

    int vertex;

    struct node \*next;

};

struct node \*createNode(int);

struct Graph

{

    int numVertices;

    struct node \*\*adjLists;

};

struct node \*createNode(int v)

{

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

    newNode->vertex = v;

    newNode->next = NULL;

    return newNode;

}

struct Graph \*createAGraph(int vertices)

{

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

    graph->numVertices = vertices;

    graph->adjLists = malloc(vertices \* sizeof(struct node \*));

    int i;

    for (i = 0; i < vertices; i++)

        graph->adjLists[i] = NULL;

    return graph;

}

void addEdge(struct Graph \*graph, int s, int d)

{

    struct node \*newNode = createNode(d);

    newNode->next = graph->adjLists[s];

    graph->adjLists[s] = newNode;

    newNode = createNode(s);

    newNode->next = graph->adjLists[d];

    graph->adjLists[d] = newNode;

}

void printGraph(struct Graph \*graph)

{

    int v;

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

    {

        struct node \*temp = graph->adjLists[v];

        printf("\n Vertex %d\n: ", v);

        while (temp)

        {

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

            if (temp->next)

            {

                printf(" -> ");

            }

            temp = temp->next;

        }

        printf("\n");

    }

}

int main()

{

    struct Graph \*graph = createAGraph(4);

    addEdge(graph, 0, 1);

    addEdge(graph, 0, 2);

    addEdge(graph, 0, 3);

    addEdge(graph, 1, 2);

    printf("\nAdjancency List\n");

    printGraph(graph);

    return 0;

}

**Output:**

