Assignment 1

1. Write a C program to print an array.

#include <stdio.h>

int main()

{

    int size;

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

    scanf("%d", &size);

    int arr[size];

    for (int i = 0; i < size; i++)

    {

        printf("Enter value for %d block: ", i);

        scanf("%d", &arr[i]);

    }

    for (int i = 0; i < size; i++)

    {

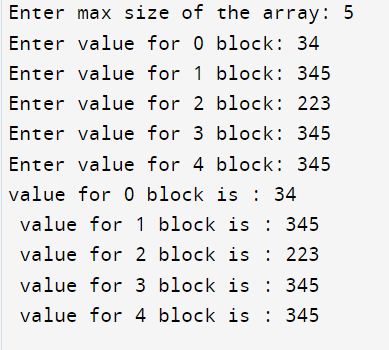
        printf(" value for %d block is : %d \n", i, arr[i]);

    }

    return 0;

}

**Output-**



2. Write a C program to check whether a given string is Palindrome or not.

#include <stdio.h>

#include<string.h>

int main()

{

char str1[1000];

int i, flag = 0;

printf("Enter the string to check for pallindrome: ");

gets(str1);

int length = strlen(str1);

// strrev(str2);

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

{

if (str1[i] != str1[length - i - 1])

{

flag = 1;

break;

}

}

if (flag == 0)

{

printf("It's a pallindrome!");

}

else

{

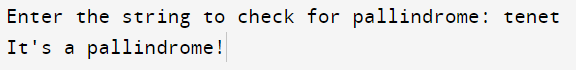
printf("It's not a pallindrome! ");

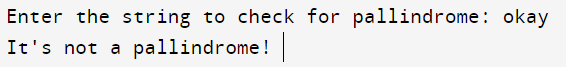
}

return 0;

}

**Output-**

****

****

3. Write a C program to convert temperature from degree Centigrade to Fahrenheit.

#include<stdio.h>

int main(){

float cel, faren;

printf("Enter the temperature in celsius: ");

scanf("%f",&cel);

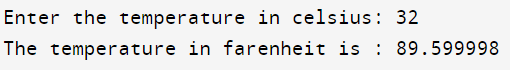
faren=(cel\*(9.0/5.0))+32.0;

printf("The temperature in farenheit is : %f",faren);

return 0;

}

**Output-**

****

4. Write a C program to sort an array.

#include <stdio.h>

void printarr(int \*a, int n)

{

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

{

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

}

}

void sortarr(int \*a, int n)

{

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

{

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

{

if (a[j] > a[j + 1])

{

int temp = a[j];

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

a[j + 1] = temp;

}

}

}

}

int main()

{

int size;

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

scanf("%d", &size);

int arr[size];

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

{

printf("Enter integer value for %d block: ", i);

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

}

printarr(arr, size);

printf("\n");

sortarr(arr, size);

printf("\n");

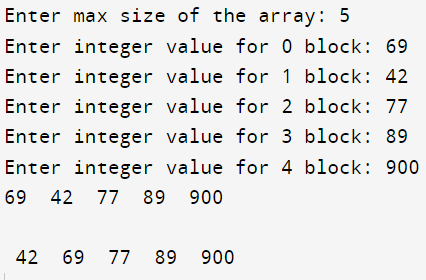
printarr(arr, size);

printf("\n");

return 0;

}

**Output-**

****

5. Write a C program to print the largest and second largest element of the array.

#include<stdio.h>

int main(){

int max=0,smax=0,length;

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

scanf("%d",&length);

int arr[length];

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

printf("Enter integer value for %d block: ",i);

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

}

//again

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

if(arr[i]>max){

smax=max;

max=arr[i];

}

else if (arr[i]>smax && arr[i]!=max)

{

smax=arr[i];

}

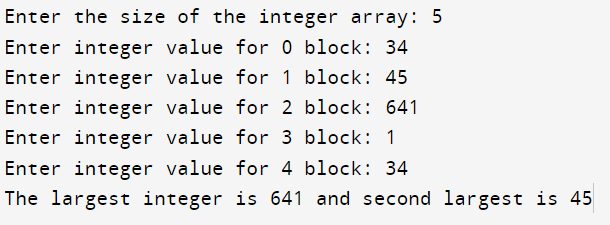
}

printf("The largest integer is %d and second largest is %d",max,smax);

return 0;

}

**Output-**

****

6. Write a C program to display Fibonacci series.

#include <stdio.h>

void main()

{

int a = 0, b = 1, c, d;

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

scanf("%d", &d);

for (int i = 1; i <= 5; i++)

{

printf(" %d ", a);

c = a + b;

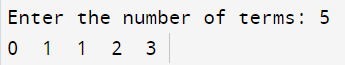
a = b;

b = c;

}

}

**Output-**

****

7. Write a program that reads two 2D metrices from the console, verifies if metrics multiplication is possible or not. Then multiplies the metrices and prints the 3rd metrics.

#include <stdio.h>

#include <math.h>

#include <string.h>

#include <stdlib.h>

#define PI 3.14159

int main()

{

int m, n, r, c;

printf("Enter Number of Rows of First Matrix: ");

scanf("%d", &m);

printf("Enter Number of Columns of First Matrix: ");

scanf("%d", &n);

printf("Enter Number of Rows of Second Matrix: ");

scanf("%d", &r);

printf("Enter Number of Columns of Second Matrix: ");

scanf("%d", &c);

if (n != r)

{

printf("This Matrix Multiplication is not possible");

exit(0);

}

int ar[m][n];

printf("Enter First Matrix Elements: ");

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

{

for (int j = 0; j < n; j++)

{

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

}

}

int arr[r][c];

printf("Enter Second Matrix Elements: ");

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

{

for (int j = 0; j < c; j++)

{

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

}

}

int a[m][c];

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

{

for (int j = 0; j < c; j++)

{

a[i][j] = 0;

}

}

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

{

for (int j = 0; j < c; j++)

{

for (int k = 0; k < n; k++)

{

a[i][j] += ar[i][k] \* arr[k][j];

}

}

}

printf("Final Matrix:\n");

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

{

for (int j = 0; j < c; j++)

{

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

}

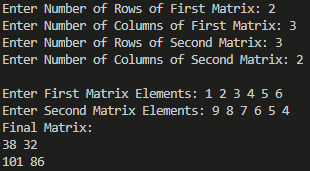
printf("\n");

}

return 0;

}

Output:



8. Write a program that reads a 2D metrics and checks if the metrics is a symmetric metrics or not.

#include <stdio.h>

int main()

{

int flag = 0, row, column;

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

scanf("%d", &row);

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

scanf("%d", &column);

int arr[row][column], trans[row][column];

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

{

for (int j = 0; j < column; j++)

{

printf("Enter the value for %d %d: ", i, j);

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

}

printf("\n");

}

printf("The value of the matrix is: \n ");

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

{

for (int j = 0; j < column; j++)

{

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

}

printf("\n");

}

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

{

for (int j = 0; j < column; j++)

{

trans[j][i] = arr[i][j];

}

}

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

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

{

for (int j = 0; j < column; j++)

{

printf("%d ", trans[i][j]);

}

printf("\n");

}

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

{

for (int j = 0; j < column; j++)

{

if (trans[i][j] != arr[i][j])

{

flag = 1;

break;

}

}

}

if (flag == 1)

{

printf("It's not symmetric");

}

else

{

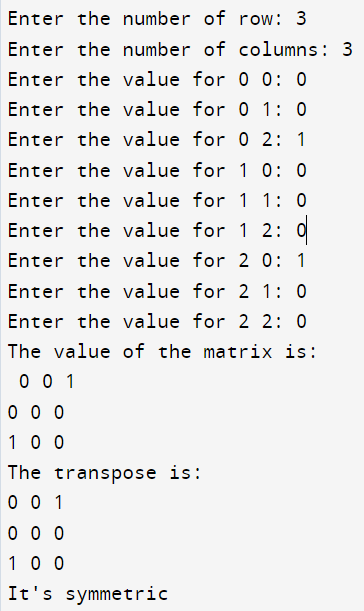
printf("It's symmetric");

}

return 0;

}

**Output-**

****

9. Write a C program to print reverse array

#include <stdio.h>

int main()

{

int size;

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

scanf("%d", &size);

int a[size];

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

{

printf("Enter value for block %d : ", i);

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

}

printf("The value you entered is : ");

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

{

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

}

printf("\n");

printf("The value in reverse are as follows: \n");

for (int i = size - 1; i >= 0; i--)

{

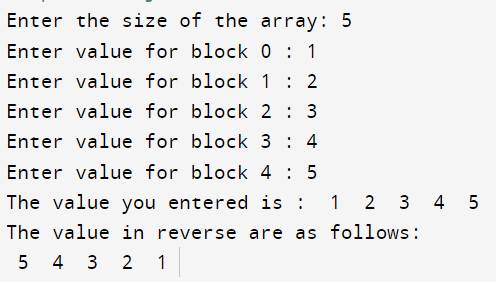
printf(" %d ", a[i]);

}

return 0;

}

**Output-**



10. Write a C program to check the sum of all elements of an array

#include <stdio.h>

int main()

{

int size, sum = 0;

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

scanf("%d", &size);

int a[size];

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

{

printf("Enter value for block %d : ", i);

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

}

printf("The value you entered is : ");

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

{

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

}

printf("\n The sum of elements is: ");

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

{

sum=sum+a[i];

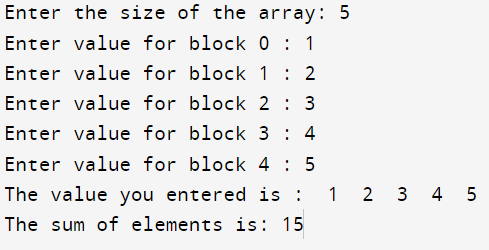
}

printf("%d",sum);

return 0;

}

**Output-**

****

11. Write a C program to check duplicate number in an array.

#include <stdio.h>

int main()

{

int size, flag = 0;

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

scanf("%d", &size);

int a[size];

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

{

printf("Enter value for block %d : ", i);

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

}

printf("The value you entered is : ");

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

{

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

}

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

{

for (int j = i + 1; j < size; j++)

{

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

{

flag = 1;

break;

}

}

}

printf("\n");

if (flag == 1)

{

printf("There is/are duplicates in the array");

}

else

{

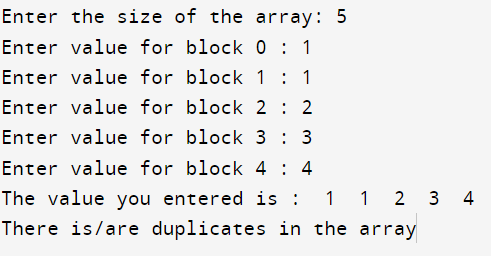
printf("No duplicates found");

}

return 0;

}

**Output-**

****

Assignment 2

1. Write a C program to read a 2D array (with most of the elements as 0s) and then represent

the same array as Sparse Metrics.

#include <stdio.h>

int r, c;

void main()

{

printf("Enter the number of rows and columns: ");

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

int a[r][c];

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

{

for (int j = 0; j < c; j++)

{

printf("Enter the values for %d %d: ", i, j);

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

}

}

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

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

{

for (int j = 0; j < c; j++)

{

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

}

printf("\n");

}

printf("Sparse Metrice representation: \n");

int nonzeroelements = 0;

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

{

for (int j = 0; j < c; j++)

{

if (a[i][j] != 0)

{

printf("row= %d column =%d value = %d \n", i, j, a[i][j]);

nonzeroelements++;

}

}

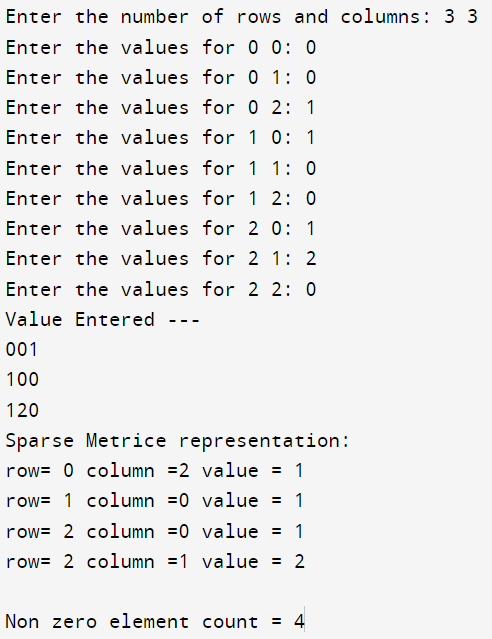
}

printf("\n");

printf("Non zero element count = %d", nonzeroelements);

}

**Output-**



2. Write a C program to pass an array to a function using Call by Value, update the array

values in the function, print the array elements both in the function and in the calling

function.

#include <stdio.h>

int main()

{

int size;

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

scanf("%d", &size);

int a[size];

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

{

printf("Enter the value for block %d: ", i);

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

}

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

{

printf(" The value for block %d before function call: %d\n", i, a[i]);

}

funct(a, size);

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

{

printf(" The value for block %d after function call: %d \n", i, a[i]);

}

return 0;

}

void funct(int a[], int size)

{

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

{

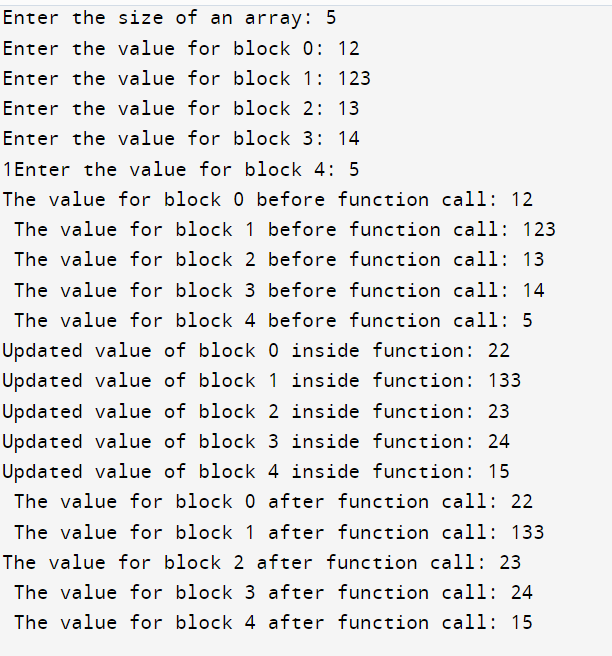
a[i] = a[i] + 10;

printf("Updated value of block %d inside function: %d \n", i,a[i]);

}

}

**Output-**

****

3. Write a C program to pass an array to a function using Call by Reference, update the array

values in the function, print the array elements both in the function and in the calling

function.

#include <stdio.h>

int main()

{

int size;

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

scanf("%d", &size);

int a[size];

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

{

printf("Enter the value for block %d: ", i);

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

}

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

{

printf(" The value for block %d before function call: %d\n", i, a[i]);

}

funct(a, size);

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

{

printf(" The value for block %d after function call: %d \n", i, a[i]);

}

return 0;

}

void funct(int \*a, int size)

{

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

{

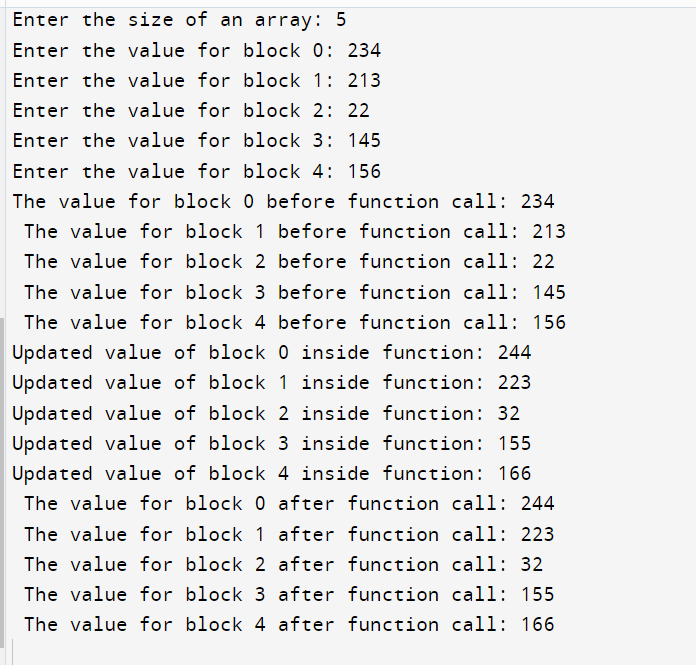
a[i] = a[i] + 10;

printf("Updated value of block %d inside function: %d \n", i, a[i]);

}

}

**Output-**



4. Write a program to display n number of elements. Memory should be allocated dynamically

// using malloc( ).

#include <stdio.h>

#include<stdlib.h>

#include <string.h>

int main()

{

int a, \*ptr;

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

scanf("%d", &a);

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

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

{

printf("Enter the value for block %d : ", i);

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

}

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

{

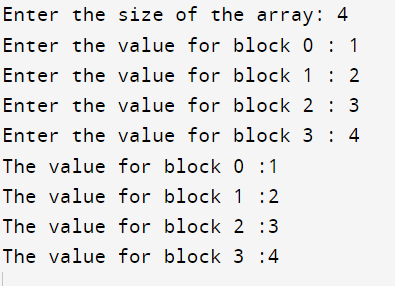
printf("The value for block %d :%d \n", i, ptr[i]);

}

return 0;

}

**Output-**



5. Write a program to display n number of elements. Memory should be allocated dynamically

using calloc( ).

#include <stdio.h>

#include<stdlib.h>

#include <string.h>

int main()

{

int a, \*ptr;

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

scanf("%d", &a);

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

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

{

printf("Enter the value for block %d : ", i);

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

}

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

{

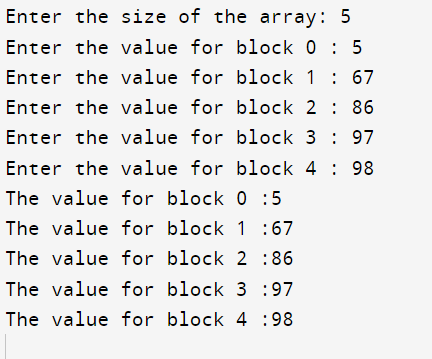
printf("The value for block %d :%d \n", i, ptr[i]);

}

return 0;

}

**Output-**



6. Write a program to allocate memory using malloc( ) and then reallocate the previously

allocated memory using realloc( ). Display the elements which have been taken after

reallocation.

#include <stdio.h>

#include <stdlib.h>

int main()

{

int is, fs, \*ptr;

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

scanf("%d", &is);

ptr = (int \*)malloc(is \* (sizeof(int)));

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

{ printf("Enter value for block %d: ",i);

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

}

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

{ printf("The value for block %d: %d \n",i,ptr[i]);

}

printf("Enter the value for reallocation:");

scanf("%d",&fs);

ptr=(int\*)realloc(ptr,fs\*sizeof(int));

printf("Enter the value of next %d elements \n",fs-is);

for(int i=is;i<fs;i++)

{ printf("Enter the value for block %d: ",i);

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

}

printf("All the elements are: \n");

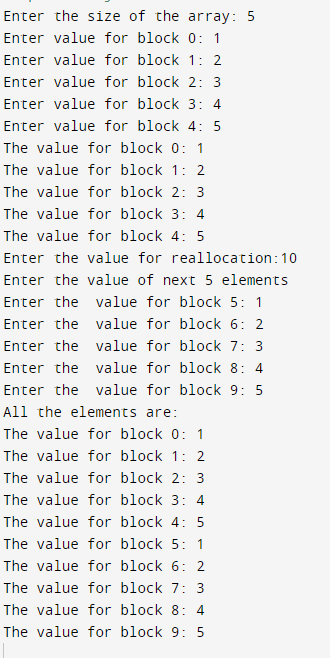
for (int i = 0; i < fs; i++)

{ printf("The value for block %d: %d \n",i,ptr[i]);

}

return 0;

**Output-**



7. Write a program to allocate memory using calloc( ) and then reallocate the previously

allocated memory using realloc( ). Display the elements which have been taken after

reallocation.

#include <stdio.h>

#include <stdlib.h>

int main()

{

int is, fs, \*ptr;

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

scanf("%d", &is);

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

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

{ printf("Enter value for block %d: ",i);

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

}

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

{ printf("The value for block %d: %d \n",i,ptr[i]);

}

printf("Enter the value for reallocation:");

scanf("%d",&fs);

ptr=(int\*)realloc(ptr,fs\*sizeof(int));

printf("Enter the value of next %d elements \n",fs-is);

for(int i=is;i<fs;i++)

{ printf("Enter the value for block %d: ",i);

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

}

printf("All the elements are: \n");

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

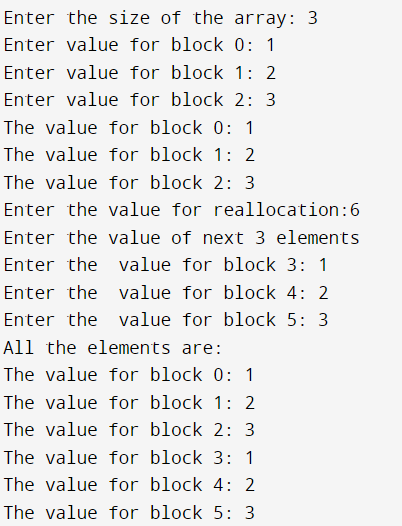
{ printf("The value for block %d: %d \n",i,ptr[i]);

}

return 0;

}

**Output-**



8. Write a C program to search an element in an Array using dynamic memory allocation.

#include <stdio.h>

#include <stdlib.h>

int main()

{

int i1, j, \*ptr, k = 0, b;

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

scanf("%d", &i1);

ptr = (int \*)malloc(i1 \* sizeof(int));

for (int i = 0; i < i1; i++)

{

printf("Enter value for block %d: ", i);

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

}

printf("Enter the element you want to find: ");

scanf("%d", &j);

for (int i = 0; i < i1; i++)

{

if (ptr[i] == j)

{

k = 1;

b = i;

break;

}

}

if (k == 0)

{

printf("Element not found!");

}

else

{

printf("Element found! \n");

printf("Its on block %d", b);

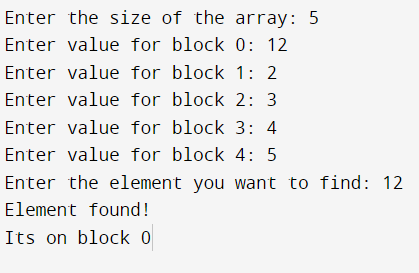
}

free(ptr);

return 0;

}

**Output-**



**ASSIGNMENT – 3**

1) Write a Menu driven C program to accomplish the following functionalities in single

linked list.

a) Create a single linked list.

b) Display the elements of a single linked list.

c) Insert a node at the beginning of a single linked list.

d) Insert a node at the end of a single linked list.

e) Insert a node before a given node of a single linked list.

f) Insert a node after a given node of a single linked list.

g) Delete a node from the beginning of a single linked list.

h) Delete a node from the end of a single linked list.

i) Delete a node after a given node of a single linked list.

j) Delete the entire single linked list.

#include <stdio.h>

#include <math.h>

#include <string.h>

#include <stdlib.h>

#define PI 3.14159

struct node

{

int data;

struct node \*link;

};

struct node \*insertEnd(struct node \*head, int var)

{

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

temp->data = var;

temp->link = NULL;

if (head == NULL)

{

head = temp;

}

else

{

struct node \*loop = head;

while (loop->link != NULL)

{

loop = loop->link;

}

loop->link = temp;

}

return head;

}

struct node \*insertBeginning(struct node \*head, int var)

{

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

temp->data = var;

temp->link = NULL;

if (head != NULL)

{

temp->link = head;

}

return temp;

}

struct node \*insertBeforeNode(struct node \*head, int var, int addbefore)

{

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

temp->data = var;

temp->link = NULL;

if (head == NULL)

{

head = temp;

}

else if (head->data == addbefore)

{

head = insertBeginning(head, var);

}

else

{

struct node \*loop1 = head;

struct node \*loop2 = head;

while (loop1->link->data != addbefore)

{

loop1 = loop1->link;

loop2 = loop2->link;

}

loop2 = loop2->link;

temp->link = loop2;

loop1->link = temp;

}

return head;

}

struct node \*insertAfterNode(struct node \*head, int var, int addafter)

{

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

temp->data = var;

temp->link = NULL;

if (head == NULL)

{

head = temp;

}

else

{

struct node \*loop1 = head;

struct node \*loop2 = head;

while (loop1->data != addafter)

{

loop1 = loop1->link;

loop2 = loop2->link;

}

loop2 = loop2->link;

temp->link = loop2;

loop1->link = temp;

}

return head;

}

struct node \*deleteBeginning(struct node \*head)

{

if (head == NULL)

{

printf("Linked List is Already Empty\n");

}

else if (head->link == NULL)

{

head = NULL;

}

else

{

head = head->link;

}

return head;

}

struct node \*deleteEnd(struct node \*head)

{

if (head == NULL)

{

printf("Linked List is Already Empty\n");

}

else if (head->link == NULL)

{

head = NULL;

}

else

{

struct node \*loop = head;

while (loop->link->link != NULL)

{

loop = loop->link;

}

loop->link = NULL;

}

return head;

}

struct node \*deleteAfterNode(struct node \*head, int deleteafter)

{

if (head == NULL)

{

printf("Linked List is Already Empty\n");

}

else

{

struct node \*loop = head;

while (loop->data != deleteafter)

{

loop = loop->link;

if (loop->link == NULL)

{

return head;

}

}

if (loop->link->link != NULL)

{

struct node \*temp = loop;

temp = temp->link->link;

loop->link = temp;

}

else

{

loop->link = NULL;

}

}

return head;

}

void display(struct node \*head)

{

printf("Your Linked List:\n");

struct node \*disp = head;

while (disp != NULL)

{

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

disp = disp->link;

}

}

int main()

{

struct node \*head = NULL;

while (1)

{

int ch, num, n;

printf("1. Insert at End\n2. Insert at Beginning\n");

printf("3. Insert Before a Node\n4. Insert After a Node\n");

printf("5. Delete from Beginning\n6. Delete from End\n");

printf("7. Delete After a Node\n8. Delete All:\n");

scanf("%d", &ch);

switch (ch)

{

case 1:

printf("Enter Data: ");

scanf("%d", &n);

head = insertEnd(head, n);

break;

case 2:

printf("Enter Data: ");

scanf("%d", &n);

head = insertBeginning(head, n);

break;

case 3:

printf("Enter Data: ");

scanf("%d", &n);

printf("Before which Number? ");

scanf("%d", &num);

head = insertBeforeNode(head, n, num);

break;

case 4:

printf("Enter Data: ");

scanf("%d", &n);

printf("After which Number? ");

scanf("%d", &num);

head = insertAfterNode(head, n, num);

break;

case 5:

head = deleteBeginning(head);

break;

case 6:

head = deleteEnd(head);

break;

case 7:

printf("After which Number? ");

scanf("%d", &num);

head = deleteAfterNode(head, num);

break;

case 8:

head = NULL;

}

display(head);

int x;

printf("\nDo you want to insert/delete more values?\n1. YES\n2. NO:\n");

scanf("%d", &x);

if (x == 2)

{

exit(0);

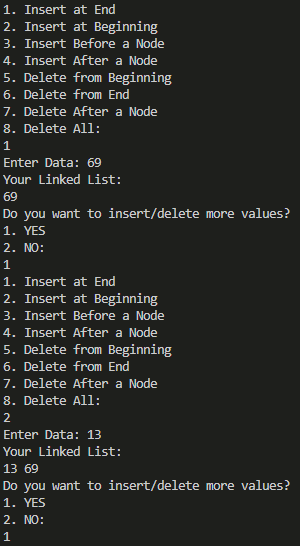
}

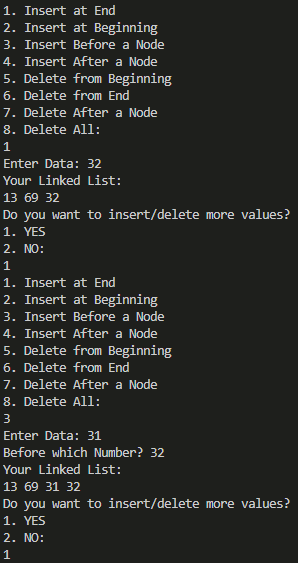
}

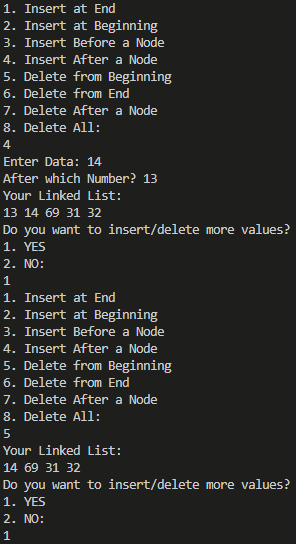
return 0;

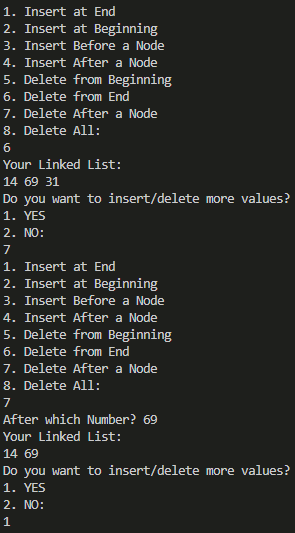
}

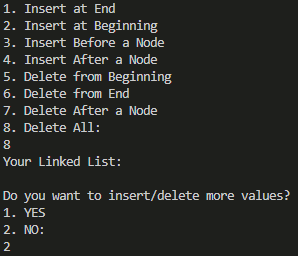
Output:











2) Write a Menu driven C program to accomplish the following functionalities in circular

linked list.

a) Create a circular linked list.

b) Display the elements of a circular linked list.

c) Insert a node at the beginning of a circular linked list.

d) Insert a node at the end of a circular linked list.

e) Delete a node from the beginning of a circular linked list.

f) Delete a node from the end of a circular linked list.

g) Delete a node after a given node of a circular linked list.

h) Delete the entire circular linked list.

#include <stdio.h>

#include <math.h>

#include <string.h>

#include <stdlib.h>

#define PI 3.14159

struct node

{

int data;

struct node \*link;

};

struct node \*insertBeginning(struct node \*head, int var)

{

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

temp->data = var;

temp->link = temp;

if (head != NULL)

{

struct node \*loop = head;

struct node \*flag = head;

while (loop->link != flag)

{

loop = loop->link;

}

loop->link = temp;

temp->link = head;

}

return temp;

}

struct node \*insertEnd(struct node \*head, int var)

{

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

temp->data = var;

temp->link = temp;

if (head == NULL)

{

head = temp;

}

else

{

struct node \*loop = head;

struct node \*flag = head;

while (loop->link != flag)

{

loop = loop->link;

}

loop->link = temp;

temp->link = flag;

}

return head;

}

struct node \*deleteBeginning(struct node \*head)

{

if (head == NULL)

{

printf("Linked List is Already Empty\n");

}

else if (head->link == head)

{

head = NULL;

}

else

{

struct node \*loop = head;

struct node \*flag = head;

while (loop->link != head)

{

loop = loop->link;

}

flag = flag->link;

loop->link = flag;

head = flag;

}

return head;

}

struct node \*deleteEnd(struct node \*head)

{

if (head == NULL)

{

printf("Linked List is Already Empty\n");

}

else if (head->link == NULL)

{

head = NULL;

}

else

{

struct node \*loop = head;

struct node \*flag = head;

while (loop->link->link != flag)

{

loop = loop->link;

}

loop->link = flag;

}

return head;

}

struct node \*deleteAfterNode(struct node \*head, int deleteafter)

{

if (head == NULL)

{

printf("Linked List is Already Empty");

}

else

{

struct node \*loop = head;

struct node \*flag = head;

while (loop->data != deleteafter)

{

if (loop->link == flag)

{

printf("Invalid Number");

return head;

}

loop = loop->link;

}

if (loop->link == head)

{

head = head->link;

loop->link = head;

}

else

{

struct node \*temp = loop;

temp = temp->link->link;

loop->link = temp;

}

}

return head;

}

void display(struct node \*head)

{

printf("Your Linked List:\n");

if (head != NULL)

{

struct node \*disp = head;

do

{

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

disp = disp->link;

} while (disp != head);

}

}

int main()

{

struct node \*head = NULL;

while (1)

{

int ch, num, n;

printf("1. Insert at Beginning\n2. Insert at End\n");

printf("3. Delete from Beginning\n4. Delete from End\n");

printf("5. Delete After a Node\n6. Delete All:\n");

scanf("%d", &ch);

switch (ch)

{

case 1:

printf("Enter Data: ");

scanf("%d", &n);

head = insertBeginning(head, n);

break;

case 2:

printf("Enter Data: ");

scanf("%d", &n);

head = insertEnd(head, n);

break;

case 3:

head = deleteBeginning(head);

break;

case 4:

head = deleteEnd(head);

break;

case 5:

printf("After which Number? ");

scanf("%d", &num);

head = deleteAfterNode(head, num);

break;

case 6:

head = NULL;

}

display(head);

int x;

printf("\nDo you want to insert/delete more values?\n1. YES\n2. NO:\n");

scanf("%d", &x);

if (x == 2)

{

exit(0);

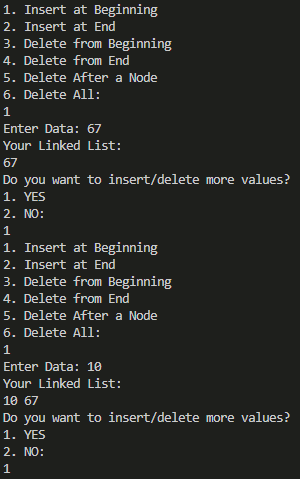
}

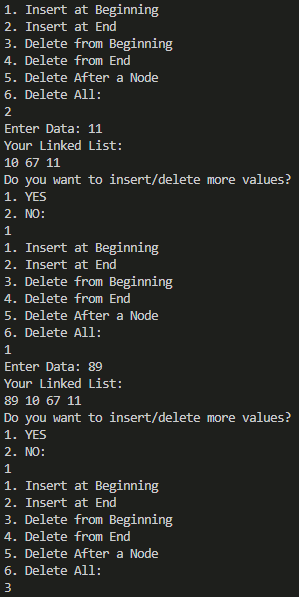
}

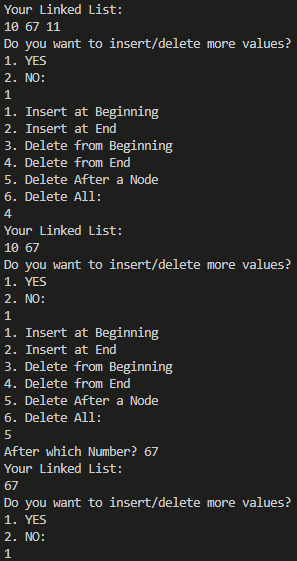
return 0;

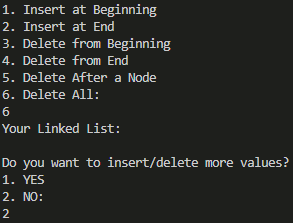
}

Output:









**Assignment – 4**

1. Write a Menu driven C program to accomplish the following functionalities in doubly linked list.

a) Create a doubly linked list.

b) Display the elements of a doubly linked list.

c) Insert a node at the beginning of a doubly linked list.

d) Insert a node at the end of a doubly linked list.

e) Insert a node before a given node of a doubly linked list.

f) Insert a node after a given node of a doubly linked list.

g) Delete a node from the beginning of a doubly linked list.

h) Delete a node from the end of a doubly linked list.

i) Delete a node after a given node of a doubly linked list.

j) Delete the entire doubly linked list.

#include <stdio.h>

#include <math.h>

#include <string.h>

#include <stdlib.h>

#define PI 3.14159

struct node

{

int data;

struct node \*prev;

struct node \*next;

};

struct node \*insertBeginning(struct node \*head, int var)

{

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

temp->data = var;

temp->prev = NULL;

temp->next = NULL;

if (head == NULL)

{

head = temp;

}

else

{

temp->next = head;

head->prev = temp;

head = temp;

}

return head;

}

struct node \*insertEnd(struct node \*head, int var)

{

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

temp->data = var;

temp->next = NULL;

temp->prev = NULL;

if (head == NULL)

{

head = temp;

}

else

{

struct node \*loop = head;

while (loop->next != NULL)

{

loop = loop->next;

}

loop->next = temp;

temp->prev = loop;

}

return head;

}

struct node \*insertBeforeNode(struct node \*head, int var, int addbefore)

{

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

temp->data = var;

temp->next = NULL;

temp->prev = NULL;

if (head == NULL)

{

head = temp;

}

else if (head->data == addbefore)

{

head = insertBeginning(head, var);

}

else

{

struct node \*loop = head;

struct node \*loop1 = head;

while (loop->next->data != addbefore)

{

loop = loop->next;

loop1 = loop1->next;

}

loop1 = loop1->next;

loop->next = temp;

temp->prev = loop;

temp->next = loop1;

loop1->prev = temp;

}

return head;

}

struct node \*insertAfterNode(struct node \*head, int var, int addafter)

{

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

temp->data = var;

temp->next = NULL;

temp->prev = NULL;

if (head == NULL)

{

head = temp;

}

else

{

struct node \*loop = head;

struct node \*loop1 = head;

while (loop->data != addafter)

{

loop = loop->next;

loop1 = loop1->next;

}

loop1 = loop1->next;

if (loop->next == NULL)

{

loop->next = temp;

temp->prev = loop;

return head;

}

loop->next = temp;

temp->prev = loop;

temp->next = loop1;

loop1->prev = temp;

}

return head;

}

struct node \*deleteBeginning(struct node \*head)

{

if (head == NULL)

{

printf("Linked List is Already Empty\n");

}

else if (head->next == NULL)

{

head = NULL;

}

else

{

head = head->next;

head->prev = NULL;

}

return head;

}

struct node \*deleteEnd(struct node \*head)

{

if (head == NULL)

{

printf("Linked List is Already Empty\n");

}

else if (head->next == NULL)

{

head = NULL;

}

else

{

struct node \*loop = head;

while (loop->next->next != NULL)

{

loop = loop->next;

}

loop->next = NULL;

}

return head;

}

struct node \*deleteAfterNode(struct node \*head, int deleteafter)

{

if (head == NULL)

{

printf("Linked List is Already Empty");

}

else

{

struct node \*loop = head;

while (loop->data != deleteafter)

{

loop = loop->next;

}

if (loop->next == NULL)

{

return head;

}

else if (loop->next->next != NULL)

{

struct node \*temp = loop;

temp = temp->next->next;

loop->next = temp;

temp->prev = loop;

}

else

{

loop->next = NULL;

}

}

return head;

}

void display(struct node \*head)

{

printf("Your Linked List:\n");

struct node \*disp = head;

while (disp != NULL)

{

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

disp = disp->next;

}

}

int main()

{

struct node \*head = NULL;

while (1)

{

int ch, num, n;

printf("1. Insert at Beginning\n2. Insert at End\n");

printf("3. Insert Before a Node\n4. Insert After a Node\n");

printf("5. Delete from Beginning\n6. Delete from End\n");

printf("7. Delete After a Node\n8. Delete All:\n");

scanf("%d", &ch);

switch (ch)

{

case 1:

printf("Enter Data: ");

scanf("%d", &n);

head = insertBeginning(head, n);

break;

case 2:

printf("Enter Data: ");

scanf("%d", &n);

head = insertEnd(head, n);

break;

case 3:

printf("Enter Data: ");

scanf("%d", &n);

printf("Before which Number? ");

scanf("%d", &num);

head = insertBeforeNode(head, n, num);

break;

case 4:

printf("Enter Data: ");

scanf("%d", &n);

printf("After which Number? ");

scanf("%d", &num);

head = insertAfterNode(head, n, num);

break;

case 5:

head = deleteBeginning(head);

break;

case 6:

head = deleteEnd(head);

break;

case 7:

printf("After which Number? ");

scanf("%d", &num);

head = deleteAfterNode(head, num);

break;

case 8:

head = NULL;

}

display(head);

int x;

printf("\nDo you want to insert/delete more values?\n1. YES\n2. NO:\n");

scanf("%d", &x);

if (x == 2)

{

exit(0);

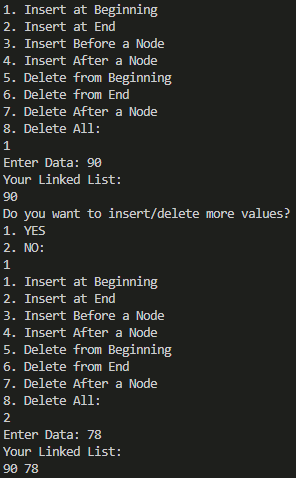
}

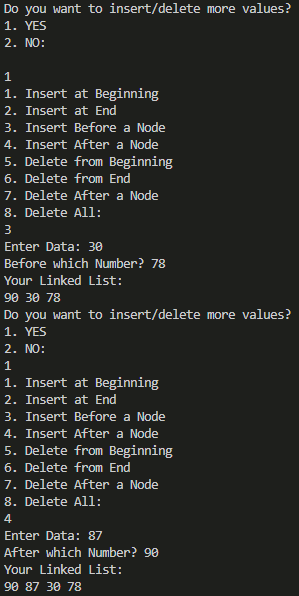
}

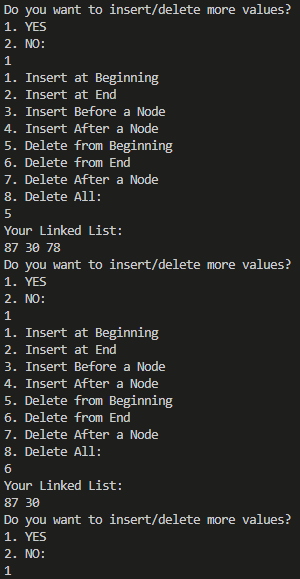
return 0;

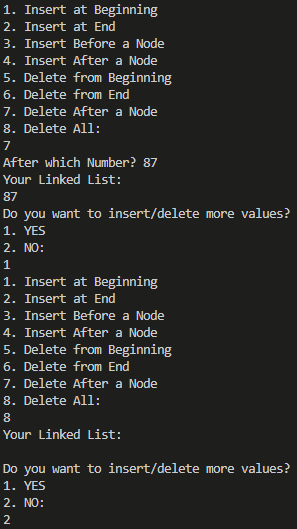
}

Output:









1. Write a Menu driven C program to accomplish the following functionalities in circular doubly linked list.

a) Create a circular doubly linked list.

b) Display the elements of a circular doubly linked list.

c) Insert a node at the beginning of a circular doubly linked list.

d) Insert a node at the end of a circular doubly linked list.

e) Delete a node from the beginning of a circular doubly linked list.

f) Delete a node from the end of a circular doubly linked list.

g) Delete a node after a given node of a circular doubly linked list.

h) Delete the entire circular doubly linked list.

#include <stdio.h>

#include <math.h>

#include <string.h>

#include <stdlib.h>

#define PI 3.14159

struct node

{

int data;

struct node \*next;

struct node \*prev;

};

struct node \*insertBeginning(struct node \*head, int var)

{

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

temp->data = var;

temp->next = temp;

temp->prev = temp;

if (head == NULL)

{

head = temp;

}

else

{

struct node \*t = head;

t = t->prev;

temp->next = head;

temp->prev = t;

head->prev = temp;

t->next = temp;

head = temp;

}

return head;

}

struct node \*insertEnd(struct node \*head, int var)

{

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

temp->data = var;

temp->next = temp;

temp->prev = temp;

if (head == NULL)

{

head = temp;

}

else

{

struct node \*t = head;

t = t->prev;

t->next = temp;

temp->prev = t;

temp->next = head;

head->prev = temp;

}

return head;

}

struct node \*deleteBeginning(struct node \*head)

{

if (head == NULL)

{

printf("Linked List is Already Empty\n");

}

else if (head->next == head)

{

head = NULL;

}

else

{

struct node \*temp = head;

temp = temp->prev;

head = head->next;

temp->next = head;

head->prev = temp;

}

return head;

}

struct node \*deleteEnd(struct node \*head)

{

if (head == NULL)

{

printf("Linked List is Already Empty\n");

}

else if (head->next == head)

{

head = NULL;

}

else

{

struct node \*temp = head;

temp = temp->prev->prev;

temp->next = head;

head->prev = temp;

}

return head;

}

struct node \*deleteAfterNode(struct node \*head, int deleteafter)

{

if (head == NULL)

{

printf("Linked List is Already Empty\n");

}

else

{

struct node \*loop = head;

struct node \*flag = head;

while (loop->data != deleteafter)

{

if (loop->next == flag)

{

printf("Invalid Number");

return head;

}

loop = loop->next;

}

if (loop->next == head)

{

head = deleteBeginning(head);

}

else

{

struct node \*temp = loop;

temp = temp->next->next;

loop->next = temp;

temp->prev = loop;

}

}

return head;

}

void display(struct node \*head)

{

printf("Your Linked List:\n");

if (head != NULL)

{

struct node \*loop = head;

do

{

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

loop = loop->next;

} while (loop != head);

}

}

int main()

{

struct node \*head = NULL;

while (1)

{

int ch, num, n;

printf("1. Insert at Beginning\n2. Insert at End\n");

printf("3. Delete from Beginning\n4. Delete from End\n");

printf("5. Delete After a Node\n6. Delete All:\n");

scanf("%d", &ch);

switch (ch)

{

case 1:

printf("Enter Data: ");

scanf("%d", &n);

head = insertBeginning(head, n);

break;

case 2:

printf("Enter Data: ");

scanf("%d", &n);

head = insertEnd(head, n);

break;

case 3:

head = deleteBeginning(head);

break;

case 4:

head = deleteEnd(head);

break;

case 5:

printf("After which Number? ");

scanf("%d", &num);

head = deleteAfterNode(head, num);

break;

case 6:

head = NULL;

}

display(head);

int x;

printf("\nDo you want to insert/delete more values?\n1. YES\n2. NO:\n");

scanf("%d", &x);

if (x == 2)

{

exit(0);

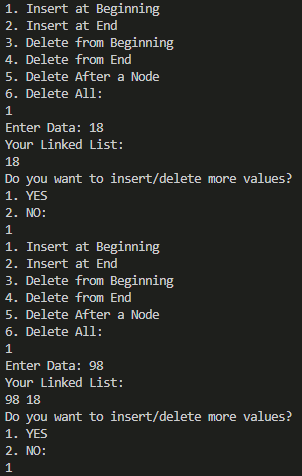
}

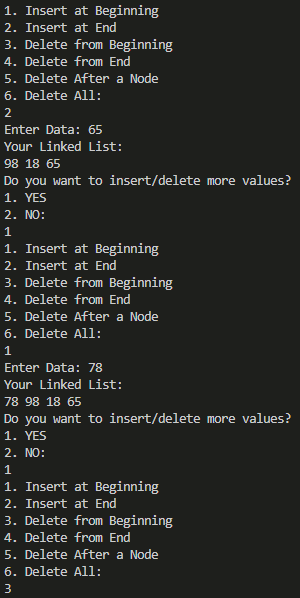
}

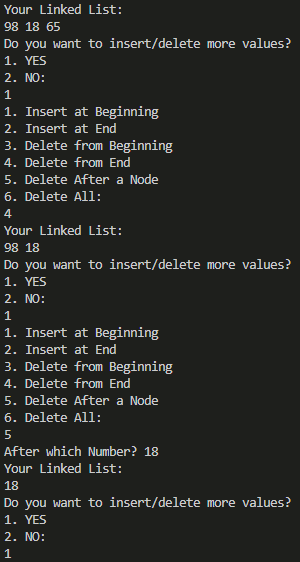
return 0;

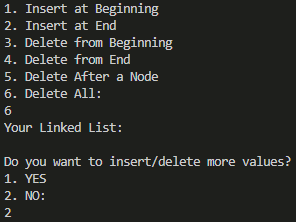
}

Output:









**ASSIGNMENT – 5**

1. Write a Menu driven C program to accomplish the following functionalities in Queue using an Array:

a. Insert an element into the queue using an array (Enqueue Operation).

b. Delete an element from the queue using an array (Dequeue Operation).

c. Return the value of the FRONT element of the queue (without deleting it from

the queue) using an array (Peep operation).

d. Display the elements of a queue using an array.

#include <stdio.h>

#include <math.h>

#include <string.h>

#include <stdlib.h>

#define PI 3.14159

int main()

{

int n;

printf("Enter Number of Elements: ");

scanf("%d", &n);

int ar[n];

int ch;

int front = -1, rear = -1;

while (1)

{

printf("1. Enqueue\n2. Dequeue\n3. Peek:\n");

scanf("%d", &ch);

switch (ch)

{

case 1:

if (rear == n - 1)

{

printf("Queue is Full\n");

}

else if (rear == -1)

{

front = 0;

rear = 0;

printf("Enter Data: ");

scanf("%d", &ar[rear]);

}

else

{

rear++;

printf("Enter Data: ");

scanf("%d", &ar[rear]);

}

break;

case 2:

if (front == -1)

{

break;

}

else if (front == rear)

{

front = -1;

rear = -1;

}

else

{

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

{

ar[i] = ar[i + 1];

}

rear--;

}

break;

case 3:

if (front != -1)

{

printf("The First Value of the Queue is: %d\n", ar[front]);

}

else

{

printf("Queue is Empty\n");

}

break;

}

if (front == -1)

{

printf("Queue is Empty\n");

}

else

{

printf("Your Queue:\n");

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

{

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

}

printf("\n");

}

int x;

printf("Do you want to modify the Queue?\n1. Yes\n2. No:\n");

scanf("%d", &x);

if (x == 2)

{

exit(0);

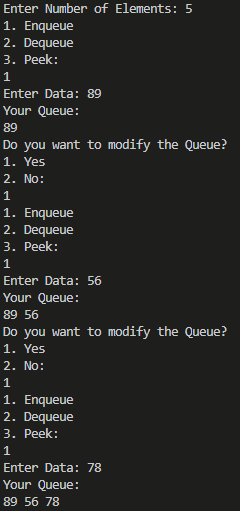
}

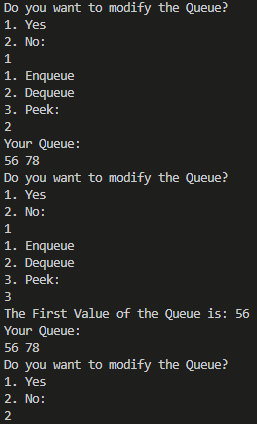
}

return 0;

}

Output:





1. Write a Menu driven C program to accomplish the following functionalities in Queue using Linked List:

e. Insert an element into the queue using a Linked List (Enqueue Operation).

f. Delete an element from the queue using a Linked List (Dequeue Operation).

g. Return the value of the FRONT element of the queue (without deleting it from

the queue) using a Linked List (Peep operation).

h. Display the elements of a queue using a Linked List.

#include <stdio.h>

#include <math.h>

#include <string.h>

#include <stdlib.h>

#define PI 3.14159

struct node

{

int data;

struct node \*link;

};

struct node \*enqueue(struct node \*head, int n, int var)

{

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

temp->data = var;

temp->link = NULL;

if (head == NULL)

{

head = temp;

}

else

{

struct node \*loop = head;

int count = 0;

while (loop->link != NULL)

{

count++;

loop = loop->link;

}

count++;

if (count >= n)

{

printf("Queue is Full\n");

}

else

{

loop->link = temp;

}

}

return head;

}

struct node \*dequeue(struct node \*head)

{

if (head == NULL)

{

printf("Queue is Empty\n");

}

else if (head->link == NULL)

{

head = NULL;

}

else

{

head = head->link;

}

return head;

}

void peek(struct node \*head)

{

if (head == NULL)

{

printf("Queue is Empty\n");

}

else

{

printf("The First Value of the Queue is: %d\n", head->data);

}

}

void display(struct node \*head)

{

if (head == NULL)

{

printf("Queue is Empty\n");

}

else

{

printf("Your Queue is:\n");

struct node \*loop = head;

while (loop != NULL)

{

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

loop = loop->link;

}

printf("\n");

}

}

int main()

{

int n;

printf("Enter Number of Elements: ");

scanf("%d", &n);

struct node \*head = NULL;

int ch, num;

while (1)

{

printf("1. Enqueue\n2. Dequeue\n3. Peek:\n");

scanf("%d", &ch);

switch (ch)

{

case 1:

printf("Enter Data: ");

scanf("%d", &num);

head = enqueue(head, n, num);

break;

case 2:

head = dequeue(head);

break;

case 3:

peek(head);

break;

}

display(head);

int x;

printf("Do you want to modify the Queue?\n1. Yes\n2. No:\n");

scanf("%d", &x);

if (x == 2)

{

exit(0);

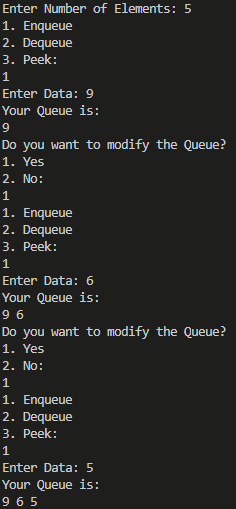
}

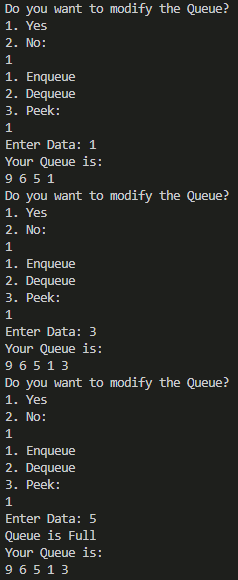
}

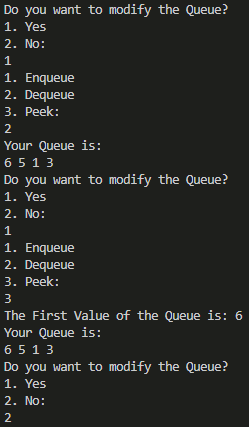
return 0;

}

Output:







1. Write a Menu driven C program to accomplish the following functionalities in Circular Queue using Array:

i. Insert an element into the circular queue.

j. Delete an element from the circular queue.

k. Return the value of the FRONT element of the circular queue (without deleting it

from the queue).

l. Display the elements of a circular queue using the circular queue.

#include <stdio.h>

#include <math.h>

#include <string.h>

#include <stdlib.h>

#define PI 3.14159

struct node

{

int data;

struct node \*link;

};

struct node \*enqueue(struct node \*head, int n, int var)

{

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

temp->data = var;

temp->link = temp;

if (head == NULL)

{

head = temp;

}

else

{

struct node \*loop = head;

struct node \*flag = head;

int count = 0;

while (loop->link != flag)

{

loop = loop->link;

count++;

}

count++;

if (count >= n)

{

printf("Queue is Full\n");

}

else

{

loop->link = temp;

temp->link = flag;

}

}

return head;

}

struct node \*dequeue(struct node \*head)

{

if (head == NULL)

{

printf("Queue is Empty\n");

}

else if (head->link == head)

{

head = NULL;

}

else

{

struct node \*loop = head;

while (loop->link != head)

{

loop = loop->link;

}

head = head->link;

loop->link = head;

}

return head;

}

void peek(struct node \*head)

{

if (head == NULL)

{

printf("Queue is Empty\n");

}

else

{

printf("The First Element of the Queue is: %d\n", head->data);

}

}

void display(struct node \*head)

{

if (head == NULL)

{

printf("Queue is Empty\n");

}

else

{

printf("Your Queue is: \n");

struct node \*loop = head;

do

{

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

loop = loop->link;

} while (loop != head);

printf("\n");

}

}

int main()

{

int n;

printf("Enter Number of Elements: ");

scanf("%d", &n);

struct node \*head = NULL;

int ch, num;

while (1)

{

printf("1. Enqueue\n2. Dequeue\n3. Peek:\n");

scanf("%d", &ch);

switch (ch)

{

case 1:

printf("Enter Data: ");

scanf("%d", &num);

head = enqueue(head, n, num);

break;

case 2:

head = dequeue(head);

break;

case 3:

peek(head);

break;

}

display(head);

int x;

printf("Do you want to modify the Queue?\n1. Yes\n2. No:\n");

scanf("%d", &x);

if (x == 2)

{

exit(0);

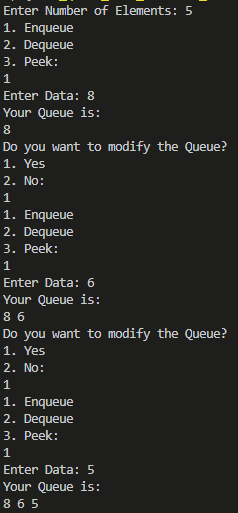
}

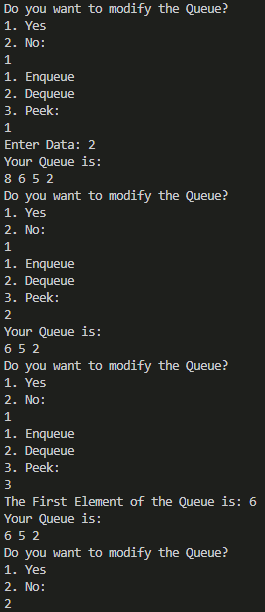
}

return 0;

}

Output:





**ASSIGNMENT – 6**

1. Write a Menu driven C program to accomplish the following functionalities in Stack using an Array:

a. Insert an element into the stack using an array (Push Operation).

b. Delete an element from the stack using an array (Pop Operation).

c. Return the value of the topmost element of the stack (without deleting it from

the stack) using an array.

d. Display the elements of a stack using an array.

#include <stdio.h>

#include <math.h>

#include <string.h>

#include <stdlib.h>

#define PI 3.14159

int main()

{

int n;

printf("Enter Number of Elements: ");

scanf("%d", &n);

int ar[n];

int front = -1, rear = -1;

while (1)

{

int ch;

printf("1. Push\n2. Pop\n3. Peek:\n");

scanf("%d", &ch);

switch (ch)

{

case 1:

if (front == -1)

{

front = 0;

rear = 0;

printf("Enter Data: ");

scanf("%d", &ar[front]);

}

else if (rear == (n - 1))

{

printf("Stack is Full\n");

}

else

{

rear++;

printf("Enter Data: ");

scanf("%d", &ar[rear]);

}

break;

case 2:

if (front == -1)

{

break;

}

else if (front == rear)

{

front = -1;

rear = -1;

}

else

{

rear--;

}

break;

case 3:

if (front == -1)

{

break;

}

else

{

printf("The Topmost Value in the Stack is: %d", ar[rear]);

}

break;

}

if (front == -1)

{

printf("Stack is Empty\n");

}

else

{

printf("Your Stack: \n");

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

{

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

}

printf("\n");

}

int x;

printf("Do you want to modify the Stack?\n1. Yes\n2. No:\n");

scanf("%d", &x);

if (x == 2)

{

exit(0);

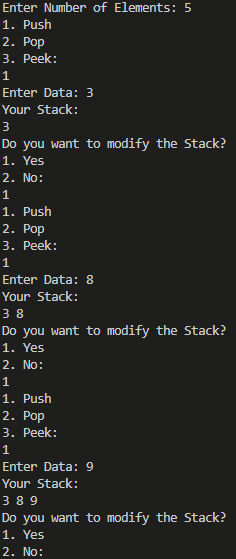
}

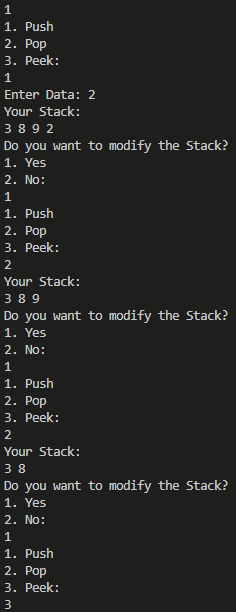
}

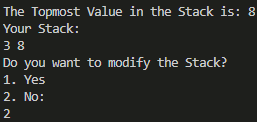
return 0;

}

Output:







1. Write a Menu driven C program to accomplish the following functionalities in Stack using Linked List:

a. Insert an element into the stack using a Linked List (Push Operation).

b. Delete an element from the stack using a Linked List (Pop Operation).

c. Return the value of the topmost element of the stack (without deleting it from

the stack) using a Linked List.

d. Display the elements of the stack using a Linked List.

#include <stdio.h>

#include <math.h>

#include <string.h>

#include <stdlib.h>

#define PI 3.14159

struct node

{

int data;

struct node \*link;

};

struct node \*push(struct node \*head, int n)

{

if (head == NULL)

{

int var;

printf("Enter Data: ");

scanf("%d", &var);

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

temp->data = var;

temp->link = NULL;

head = temp;

}

else

{

struct node \*loop = head;

int count = 0;

while (loop->link != NULL)

{

loop = loop->link;

count++;

}

count++;

if (count >= n)

{

printf("Stack is Full\n");

}

else

{

int var;

printf("Enter Data: ");

scanf("%d", &var);

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

temp->data = var;

temp->link = NULL;

loop->link = temp;

}

}

return head;

}

struct node \*pop(struct node \*head)

{

if (head == NULL)

{

return head;

}

else if (head->link == NULL)

{

head = NULL;

}

else

{

struct node \*loop = head;

while (loop->link->link != NULL)

{

loop = loop->link;

}

loop->link = NULL;

}

return head;

}

void peek(struct node \*head)

{

if (head != NULL)

{

struct node \*loop = head;

while (loop->link != NULL)

{

loop = loop->link;

}

printf("The Topmost Element in this Stack is: %d\n", loop->data);

}

}

void display(struct node \*head)

{

if (head == NULL)

{

printf("Stack is Empty\n");

}

else

{

struct node \*loop = head;

while (loop != NULL)

{

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

loop = loop->link;

}

}

}

int main()

{

struct node \*head = NULL;

int n;

printf("Enter Number of Elements: ");

scanf("%d", &n);

while (1)

{

int ch;

printf("1. Push\n2. Pop\n3. Peek:\n");

scanf("%d", &ch);

switch (ch)

{

case 1:

head = push(head, n);

break;

case 2:

head = pop(head);

break;

case 3:

peek(head);

break;

}

display(head);

int x;

printf("\nDo you want to modify the Stack?\n1. YES\n2. NO:\n");

scanf("%d", &x);

if (x == 2)

{

exit(0);

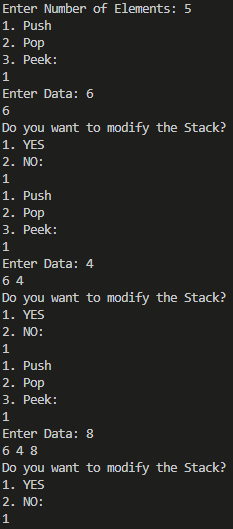
}

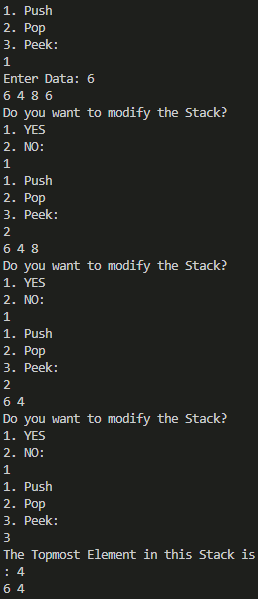
}

return 0;

}

Output:





1. Write a program to convert an infix expression into its equivalent postfix notation.

#include <stdio.h>

#include <string.h>

#include <math.h>

#include <stdlib.h>

typedef struct stack

{

char \*ch;

struct stack \*next;

} stack;

stack \*head = NULL;

int top = 1;

int max;

void push(char \*x, int max)

{

if (top <= max)

{

stack \*newnode;

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

newnode->ch = (char \*)malloc(max \* (sizeof(char)));

strcpy(newnode->ch, x);

newnode->next = NULL;

if (head == NULL)

head = newnode;

else

{

stack \*tmp = head;

newnode->next = tmp;

head = newnode;

top++;

}

}

else

printf("Stack OVERFLOW!\n");

}

char \*pop()

{

if (head == NULL)

{

printf("Stack UNDERFLOW!\n");

return " ";

}

else

{

char \*s;

stack \*tmp;

tmp = head->next;

head->next = NULL;

s = head->ch;

free(head);

head = tmp;

top--;

return s;

}

}

void popall()

{

int i = 1;

if (head == NULL)

i = 0;

else

{

stack \*tmp;

tmp = head->next;

printf("%s", head->ch);

head->next = NULL;

free(head);

head = tmp;

top--;

}

if (i)

popall();

}

int isOperator(char x)

{

switch (x)

{

case '+':

case '-':

case '/':

case '\*':

case '^':

return 1;

}

return 0;

}

char \*operate(char qs)

{

char \*a, \*b, c[2];

c[0] = qs;

c[1] = '\0';

a = pop();

b = pop();

strcat(a, b);

strcat(a, c);

return a;

}

int main()

{

char qs[100];

printf(“Enter Infix: ”);

scanf("%[^\n]s", qs);

int length = strlen(qs);

char \*s = (char \*)malloc(max \* sizeof(char));

char \*q = (char \*)malloc(sizeof(char));

max = length;

for (int i = length - 1; i >= 0; i--)

{

q[0] = qs[i];

q[1] = '\0';

if (isOperator(qs[i]))

{

s = operate(qs[i]);

push(s, max);

}

else

{

push(q, max);

}

}

popall();

return 0;

}

Output:



4. Write a program to convert an infix expression into its equivalent prefix notation.

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <ctype.h>

#define MAX\_EXPRESSION\_SIZE 100

typedef struct

{

char data[MAX\_EXPRESSION\_SIZE];

int top;

} Stack;

void initialize(Stack \*stack)

{

stack->top = -1;

}

void push(Stack \*stack, char item)

{

if (stack->top == MAX\_EXPRESSION\_SIZE - 1)

{

printf("Stack Overflow\n");

exit(EXIT\_FAILURE);

}

stack->data[++stack->top] = item;

}

char pop(Stack \*stack)

{

if (stack->top == -1)

{

printf("Stack Underflow\n");

exit(EXIT\_FAILURE);

}

return stack->data[stack->top--];

}

int isOperator(char c)

{

return c == '+' || c == '-' || c == '\*' || c == '/';

}

int getPrecedence(char operator)

{

switch (operator)

{

case '+':

case '-':

return 1;

case '\*':

case '/':

return 2;

default:

return 0;

}

}

void reverseString(char \*str)

{

int length = strlen(str);

int i, j;

char temp;

for (i = 0, j = length - 1; i < j; i++, j--)

{

temp = str[i];

str[i] = str[j];

str[j] = temp;

}

}

void infixToPrefix(char \*infixExpression, char \*prefixExpression)

{

Stack operatorStack;

initialize(&operatorStack);

reverseString(infixExpression);

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

{

if (infixExpression[i] == '(')

{

infixExpression[i] = ')';

}

else if (infixExpression[i] == ')')

{

infixExpression[i] = '(';

}

}

int j = 0;

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

{

char currentChar = infixExpression[i];

if (isalnum(currentChar))

{

prefixExpression[j++] = currentChar;

}

else if (currentChar == '(')

{

push(&operatorStack, currentChar);

}

else if (currentChar == ')')

{

while (operatorStack.top != -1 && operatorStack.data[operatorStack.top] != '(')

{

prefixExpression[j++] = pop(&operatorStack);

}

pop(&operatorStack);

}

else if (isOperator(currentChar))

{

while (operatorStack.top != -1 && operatorStack.data[operatorStack.top] != '(' &&

getPrecedence(operatorStack.data[operatorStack.top]) >= getPrecedence(currentChar))

{

prefixExpression[j++] = pop(&operatorStack);

}

push(&operatorStack, currentChar);

}

}

while (operatorStack.top != -1)

{

prefixExpression[j++] = pop(&operatorStack);

}

prefixExpression[j] = '\0';

reverseString(prefixExpression);

}

int main()

{

char infixExpression[MAX\_EXPRESSION\_SIZE];

printf("Enter Infix: ");

scanf("%s", infixExpression);

char prefixExpression[MAX\_EXPRESSION\_SIZE];

infixToPrefix(infixExpression, prefixExpression);

printf("Prefix Expression: %s\n", prefixExpression);

return 0;

}

Output:



5. Write a program to evaluate a postfix expression.

#include <stdio.h>

#include <stdlib.h>

#include <ctype.h>

#define MAX\_EXPRESSION\_SIZE 100

typedef struct

{

int data[MAX\_EXPRESSION\_SIZE];

int top;

} Stack;

void initialize(Stack \*stack)

{

stack->top = -1;

}

void push(Stack \*stack, int item)

{

if (stack->top == MAX\_EXPRESSION\_SIZE - 1)

{

printf("Stack Overflow\n");

exit(EXIT\_FAILURE);

}

stack->data[++stack->top] = item;

}

int pop(Stack \*stack)

{

if (stack->top == -1)

{

printf("Stack Underflow\n");

exit(EXIT\_FAILURE);

}

return stack->data[stack->top--];

}

int evaluatePostfix(char \*postfixExpression)

{

Stack operandStack;

initialize(&operandStack);

for (int i = 0; postfixExpression[i] != '\0'; i++)

{

char currentChar = postfixExpression[i];

if (isdigit(currentChar))

{

push(&operandStack, currentChar - '0');

}

else if (currentChar == ' ')

{

continue;

}

else

{

int operand2 = pop(&operandStack);

int operand1 = pop(&operandStack);

switch (currentChar)

{

case '+':

push(&operandStack, operand1 + operand2);

break;

case '-':

push(&operandStack, operand1 - operand2);

break;

case '\*':

push(&operandStack, operand1 \* operand2);

break;

case '/':

push(&operandStack, operand1 / operand2);

break;

default:

printf("Invalid operator: %c\n", currentChar);

exit(EXIT\_FAILURE);

}

}

}

return pop(&operandStack);

}

int main()

{

char postfixExpression[MAX\_EXPRESSION\_SIZE];

printf("Enter Postfix Expression:");

scanf("%s", postfixExpression);

int result = evaluatePostfix(postfixExpression);

printf("Result: %d\n", result);

return 0;

}

Output:



6. Write a program to evaluate a prefix expression.

#include <stdio.h>

#include <stdlib.h>

#include <ctype.h>

#include <string.h>

#define MAX\_SIZE 50

int stack[MAX\_SIZE];

int top = -1;

void initialize()

{

top = -1;

}

int isEmpty()

{

return top == -1;

}

int isFull()

{

return top == MAX\_SIZE - 1;

}

void push(int value)

{

if (isFull())

{

printf("Stack overflow. Cannot push %d.\n", value);

}

else

{

stack[++top] = value;

}

}

int pop()

{

if (isEmpty())

{

printf("Stack underflow. Cannot pop from an empty stack.\n");

return -1; // Return an invalid value

}

else

{

return stack[top--];

}

}

int evaluatePrefix(char \*expression)

{

int len = strlen(expression);

for (int i = len - 1; i >= 0; i--)

{

if (isdigit(expression[i]))

{

push(expression[i] - '0');

}

else if (expression[i] == ' ')

{

continue;

}

else

{

int operand1 = pop();

int operand2 = pop();

switch (expression[i])

{

case '+':

push(operand1 + operand2);

break;

case '-':

push(operand1 - operand2);

break;

case '\*':

push(operand1 \* operand2);

break;

case '/':

push(operand1 / operand2);

break;

default:

printf("Invalid operator: %c\n", expression[i]);

return -1; // Return an error code

}

}

}

return pop();

}

int main()

{

char expression[MAX\_SIZE];

printf("Enter a prefix expression: ");

scanf("%[^\n]%\*c", expression);

int result = evaluatePrefix(expression);

if (result != -1)

{

printf("Result: %d\n", result);

}

else

{

printf("Error in evaluating the expression.\n");

}

return 0;

}

Output:



7. Write a program to print the Fibonacci series using recursion.

#include <stdio.h>

int fibonacci(int n)

{

if (n <= 1)

{

return n;

}

else

{

return fibonacci(n - 1) + fibonacci(n - 2);

}

}

void printFibonacciSeries(int n)

{

printf("Fibonacci Series up to %d terms:\n", n);

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

{

printf("%d ", fibonacci(i));

}

printf("\n");

}

int main()

{

int terms;

printf("Enter the number of terms in the Fibonacci series: ");

scanf("%d", &terms);

printFibonacciSeries(terms);

return 0;

}

Output:



8. Write a program to solve the tower of Hanoi problem using recursion.

#include <stdio.h>

void towerOfHanoi(int n, char source, char auxiliary, char destination)

{

if (n == 1)

{

printf("Move disk 1 from %c to %c\n", source, destination);

return;

}

towerOfHanoi(n - 1, source, destination, auxiliary);

printf("Move disk %d from %c to %c\n", n, source, destination);

towerOfHanoi(n - 1, auxiliary, source, destination);

}

int main()

{

int n;

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

scanf("%d", &n);

printf("Steps to solve the Tower of Hanoi problem with %d disks:\n", n);

towerOfHanoi(n, 'A', 'B', 'C');

return 0;

}

Output:

