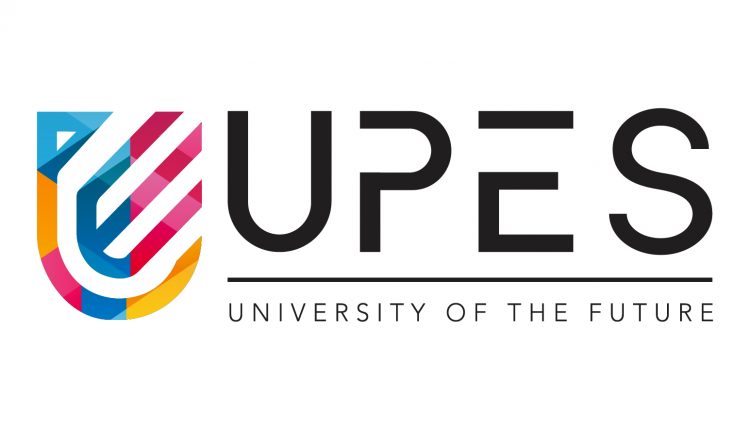
**SCHOOL OF COMPUTER SCIENCE**

**UNIVERSITY OF PETROLEUM AND ENERGY STUDIES**

**DEHRADUN, UTTRAKHAND**



**DATA STRUCTURE LAB**

**(CSEG 1111)**

**LAB FILE**

**(2021-2022)**

**FOR**

**2ND SEMESTER**

**SUBMITTED TO**: **SUBMITTED BY:**

MR. PUSHPENDRA K RAJPUT MUSKAN SINGH

ASSISTANT PROFESSOR-SS B.Tech CSE[2ND SEMESTER]

CYBERNETICS CLUSTER, R214220509

SCHOOL OF COMPUTER SCIENCE [BATCH 1]

|  |  |  |  |
| --- | --- | --- | --- |
| S. NO. | EXP.NO. | DESCRIPTION | PAGE NO. |
| 1 | 1.1L | FIND THE SUM OF ALL ARRAY ELEMENTS USING RECURSION | 1 to 3 |
| 2 | 1.2L | CREATE AN ARRAY ‘a1’ WITH ‘n’ ELEMENTS . INSERT AN ELEMENT IN THE ith POSITION OF ‘a1’ AND ALSO DELETE AN ELEMENT FROM jth POSITION OF ‘a1’. | 3 to 5 |
| 3 | 1.3L | CONVERT UPPERCASE STRING TO LOWERCASE USING FOR LOOP. | 5 to 6 |
| 4 | 1.4L | FIND THE SUM OF ROWS AND COLUMNS OF MATRIX OF GIVEN ORDER(ROW X COLUMN). | 6 to 9 |
| 5 | 1.5L | FIND THE PRODUCT OF TWO MATRICES USING POINTERS. | 9 to 14 |
| 6 | 1.6L | STORE ‘n’ NUMBERS (INTEGERS OR REAL) IN AN ARRAY. CONDUCT A LINEAR SEARCH FOR A GIVEN NUMBER AND REPORT SUCCESS OR FAILURE IN THE FORM OF A SUITABLE MESSAGE. | 14 to 15 |
| 7 | 1.1P | FIND THE TRANSPOSE AND INVERSE OF A MATRIX. | 16 to 23 |
| 8 | 1.2P | FIND IF THE GIVEN MATRIX OF ORDER (m x n) IS A SPARSE MATRIX OR NOT. | 23 to 25 |
| 9 | 1.3P | STORE ‘n’ NUMBERS (INTEGERS OR REAL) IN AN ARRAY IN ASCENDING OR DESCENDING ORDER. CONDUCT A BINARY SEARCH FOR A GIVEN NUMBER AND REPORT SUCCESS OR FAILURE IN THE FORM OF SUITABLE MESSAGE. | 25 to 27 |
| 10 | 1.4P | FIND OUT THE LARGEST AND SMALLEST NUMBER IN A GIVEN ARRAY. | 27 to 29 |
| 11 | 1.5P | CREATE AN ARRAY ‘a1’ WITH n ELEMENTS.   1. COPY ALL ELEMENTS OF ‘a1’ INTO ANOTHER ARRAY ‘a2’ USING POINTERS. DISPLAY THE CONTENTS OF BOTH THE ARRAY USING POINTERS. 2. MERGE THE CONTENTS OF ‘a1’ AND ‘a2’ INTO A NEW ARRAY ‘a3’ | 29 to 34 |
| 12 | 1.6P | FIND THE NUMBER OF ALPHABETS, DIGITS OR SPECIAL CHARACTERS IN A STRING. | 34 to 36 |
| 13 | 1.7P | FIND WHETHER THE ENTERED STRING IS PALINDROME OR NOT. | 36 to 38 |
| 14 | 1.8P | COUNT THE NUMBER OF WORDS IN A STRING. | 38 to 40 |
| 17 | 1.11P | ADD TWO COMPLEX NUMBERS BY PASSING STRUCTURE TO A FUNCTION AS ARGUMENT. | 40 to 41 |
| 18 | 2.1L | DESIGN A STRUCTURE ‘SUBJECT’ TO STORE THE DETAILS OF THE SUBJECTS LIKE SUBJECT NAME AND SUBJECT CODE. USING STRUCTURE POINTERS ALLOCATE MEMORY FOR THE STRUCTURE DYNAMICALLY SO AS TO OBTAIN DETAILS OF ‘N’ SUBJECTS USING FOR LOOP. | 42 to 43 |
| 19 | 2.2L | CREATE AN INTEGER ARRAY OF USER DEFINED SIZE N1 WITH DYNAMIC MEMORY ALLOCATION. STORE DATA AFTER READING FROM KEYBOARD. EXPAND THE SIZE OF ARRAY WITH N2. READ NEW VALUES (N2 VALUES FROM KEYBOARD). PRINT STATE OF ARRAY WITH ALL (NI+N2) VALUES**.** | 43 to 45 |
| 20 | 3.1L | IMPLEMENT SINGLE LINKED LIST DATA STRUCTURE AND ITS OPERATIONS LIKE INSERT AND DELETE IN THE BEGINNING, END AND NTH POSITION OF THE LIST AND DISPLAY THE ITEMS STORED IN THE LINKED LIST. | 46 to 54 |
| 21 | 3.2L | USING SINGLE LINKED LIST AND FUNCTIONS IMPLEMENT STACK AND ITS OPERATIONS LIKE INSERT, DELETE AND DISPLAY. | 55 to 60 |
| 22 | 3.1P | ADD TWO POLYNOMIALS USING LINKED LIST | 60 to 66 |
| 23 | 3.2P | CREATE A SORTED LINKED LIST. IMPLEMENT INSERTION AND SEARCHING OPERATION ON THE LIST. ALSO WRITE A FUNCTION DISPLAY THAT PRINTS THE CURRENT STATE OF THE LINKED LIST. | 66 to 70 |
| 24 | 3.3P | IMPLEMENT CIRCULAR LINKED LIST AND ITS OPERATIONS | 70 to 79 |
| 25 | 3.4P | IMPLEMENT DOUBLY LINKED LIST AND ITS OPERATIONS | 79 to 89 |
| 26 | 4.1L | USING ARRAY AND FUNCTIONS IMPLEMENT STACK AND ITS OPERATIONS LIKE PUSH, POP, PEEK. | 90-93 |
| 27 | 4.2L | USE THE STACK OPERATIONS DEVELOPED IN PROBLEM 1.4L AND REVERSE A STRING USING STACK. | 93-96 |
| 28 | 4.1P | USING ARRAY AND FUNCTIONS IMPLEMENT TWO STACKS AND ITS OPERATIONS. | 96-98 |
| 29 | 4.2P | CONVERT INFIX TO POSTFIX EXPRESSION USING STACK AND ARRAY. | 98-100 |
| 30 | 4.3P | EVALUATE POSTFIX EXPRESSION USING STACK AND ARRAY. | 100-102 |
| 31 | 4.4P | IMPLEMENT QUEUE OPERATIONS USING STACK. | 102-105 |
| 32 | 5.1L | USING CIRCULAR ARRAY AND FUNCTIONS IMPLEMENT QUEUE AND ITS OPERATIONS. | 106-109 |
| 33 | 5.2L | CHECK WHETHER THE STRING IS PALINDROME OR NOT USING QUEUE AND STACK. | 109-111 |
| 34 | 5.1P | IMPLEMENT QUEUE DATA STRUCTURE USING LINKED LIST AND ITS OPERATIONS. | 111-114 |
| 35 | 5.2P | USING LINKED LIST AND FUNCTIONS IMPLEMENT PRIORITY QUEUE AND ITS OPERATIONS. | 114-118 |
| 36 | 5.3P | USING ARRAY AND FUNCTIONS IMPLEMENT DOUBLE ENDED QUEUE (INPUT RESTRICTED AND OUTPUT RESTRICTED AND ITS OPERATIONS). | 118-126 |
| 37 | 6.1L | WAP TO IMPLEMENT LINEAR SEARCH ON A TWO WAY HEADER LINKED LIST. | 126-128 |
| 38 | 6.1P | MERGE THE CONTENTS OF TWO SORTED LINEAR LISTS TO A SINGLE SORTED LINEAR LIST. | 128-132 |
| 39 | 7.1L | READ THE NUMBERS FROM A TEXT FILE SORT THEM INTO AN ARRAY USING INSERTION SORT ALGORITHM AND WRITE BACK IN ANOTHER TEXT FILE. | 132-134 |
| 40 | 7.2L | READ THE NUMBERS FROM A TEXT FILE SORT THEM INTO AN ARRAY USING QUICK SORT ALGORITHM AND WRITE BACK IN ANOTHER TEXT FILE. | 134-136 |
| 41 | 7.1P | READ THE NUMBERS FROM A TEXT FILE SORT THEM INTO AN ARRAY USING SELECTION SORT ALGORITHM AND WRITE BACK IN ANOTHER TEXT FILE. | 137-139 |
| 42 | 7.2P | READ THE NUMBERS FROM A TEXT FILE SORT THEM INTO AN ARRAY USING MERGE SORT ALGORITHM AND WRITE BACK IN ANOTHER TEXT FILE. | 139-142 |
| 43 | 8.1L | IMPLEMENTING HASH TABLES WITH AND WITHOUT COLLISIO AVOIDANCE ALGORITHMS USING ARRAY/LINKED LIST. | 142-146 |
| 44 | 8.2L | IMPLEMENT A HASH TABLE USING ARRAYS.PERFORM INSERT, DELETE, SEARCH OPERATIONS ON THE HASH TABLE USING THE ABOVE HASH FUNCTION AND WITH LINEAR PROBING AS COLLISION AVOIDANCE. | 146-150 |

**EXPERIMENT 1**

**TITLE**: 1- ARRAY AND STRUCTURE

**OBJECTIVE:** TO APPLY THE CONCEPT OF ARRAY, STRUCTURE AND EXPERIMENT ON NESTED ARRAY AND ARRAY OF STRUCTURES

**PROBLEM – 1.1L:** FIND SUM OF ALL ARRAY ELEMENTS USING RECURSION

**SOLUTION APPROACH:** LET inputArray IA AN INTEGER ARRAY CONTAINING N ELEMENTS FROM INDEX 0 TO N-1 AND lastIndex IS AN INTEGER VARIABLE.

* INITIALIZE lastIndex WITH THE INDEX OF LAST ELEMENT IN ARRAY(lastIndex=N-1).
* WE CAN CALCULATE THE SUM OF inputArray ELEMENTS FROM INDEX BY ADDING SUM OF ELEMENTS FROM 0 TO N-2 AND inputarray[N-1].
* LET getSum(inputArray, lastIndex) function calculates sum of all elements of inputArray from index 0 to lastIndex.
* inputArray[0] + inputArray[1]…………inputArray[lastIndex-1]+inputArray[lastIndex]= (inputArray[0] + inputArray[1]……inputArray[lastIndex-1] +inputArray[lastIndex]
* getSum(inputArray, N-1) = getSum(inputArray, lastIndex-1) + inputArray[lastIndex]
* RECURSION WILL TERMINATE WHEN lastIndex < 0

**SOURCE CODE:**

/\*

\* C Program to find sum of N numbers using divide and conquer

\*/

#include <stdio.h>

#include <conio.h>

int main(){

int inputArray[100], counter, numberCount;

printf("Enter number of elements in Array: ");

scanf("%d", &numberCount);

printf("Enter %d numbers \n ", numberCount);

for(counter = 0; counter < numberCount; counter++){

scanf("%d", &inputArray[counter]);

}

printf("Sum of all numbers are : %d",

getSum(inputArray, 0 ,numberCount - 1));

getch();

return 0;

}

/\*

\* getSum function divides the input array into two equal half

\* and tries to find the sum of elements in both half recursively.

\* Finally, it adds the sum of left and right sub Array and return.

\* @Algorithm Divide and Conquer

\*/

int getSum(int \*inputArray, int leftIndex, int rightIndex){

int mid;

if(NULL == inputArray || leftIndex > rightIndex)

return 0;

if(leftIndex == rightIndex)

return inputArray[leftIndex];

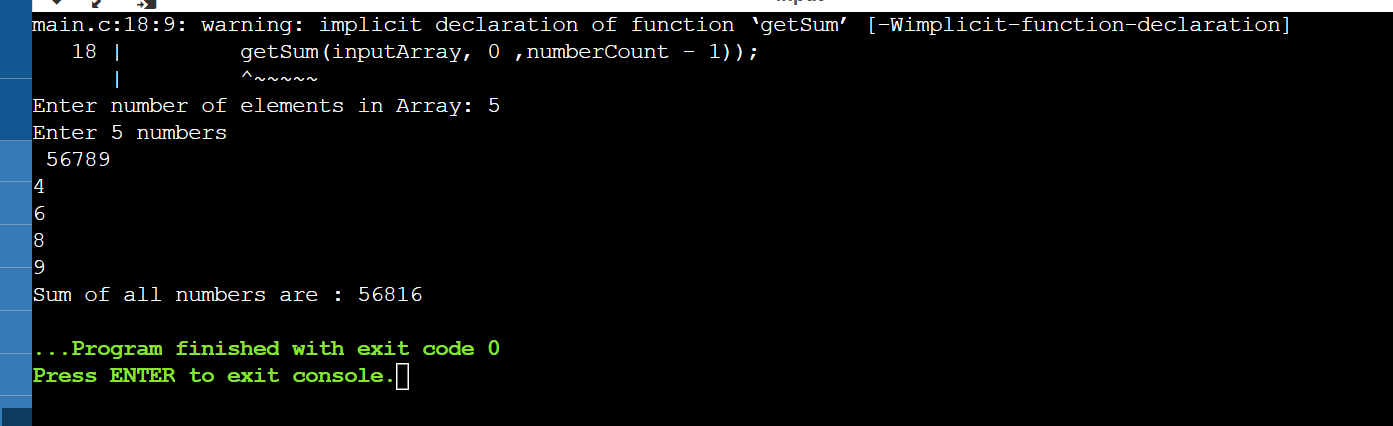
mid = (leftIndex + rightIndex) / 2;

return getSum(inputArray, leftIndex, mid) +

getSum(inputArray, mid+1, rightIndex);

}

**SAMPLE OUTPUT:**

****

**PROBLEM 1.2L:** CREATE AN ARRAY A1 WITH N ELEMENTS. INSERT AN ELEMENT IN ith POSITION OF A1 AND ALSO DELETE AN ELEMENT FROM jth POSITION OF A1.

#### SOLUTION APPROACH:

#### Insertion

* Input the array elements, the position of the new element to be inserted and the new element.
* Insert the new element at that position and shift the rest of the elements to right by one position.

**Deletion**

* Input the array elements, the position of the new element to be inserted and the new element.
* Delete the element and shift the rest of the elements to left by one position.

**SOURCE CODE:**

// C program to implement insert

// operation in an unsorted array.

#include<stdio.h>

// Inserts a key in arr[] of given capacity.

// n is current size of arr[]. This

// function returns n + 1 if insertion

// is successful, else n.

int insertSorted(int arr[], int n,

int key,

int capacity)

{

// Cannot insert more elements if n is

// already more than or equal to capacity

if (n >= capacity)

return n;

arr[n] = key;

return (n + 1);

}

// Driver Code

int main()

{

int arr[20] = {12, 16, 20, 40, 50, 70};

int capacity = sizeof(arr) / sizeof(arr[0]);

int n = 6;

int i, key = 26;

printf("\n Before Insertion: ");

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

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

// Inserting key

n = insertSorted(arr, n, key, capacity);

printf("\n After Insertion: ");

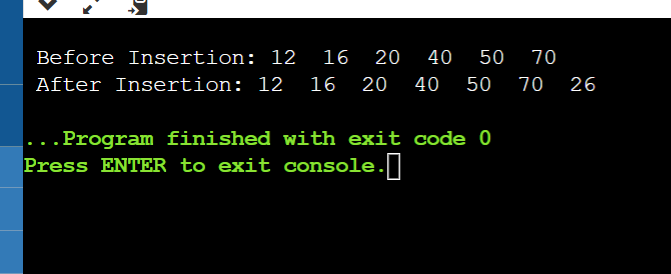
for (i = 0; i < n; i++)

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

return 0;

}

**SAMPLE OUTPUT:**

****

**PROBLEM 1.3L:** CONVERT UPPERCASE STRING TO LOWERCASE USING FOR LOOP.

**SOLUTION APPROACH:**

In this pseudocode, we declare an array of string str and a variable i. We read the string and then we run for loop to get each character of the string. We will check if each ASCII value of each character is between the value of a and z. If yes, then subtract 32 from each character to convert them into capital letters. For example, let us take string "tall" it will enter the loop as well as if statement now the 't'(ASCII value=116) will be subtracted with 32 to get 106 which is "T" now 'a'(97) will go inside the loop, it will convert to 65 which is "A". The same will happen with 'l' and at last, we will get "TALL".

**SOURCE CODE:**

//MUSKAN SINGH

//DATE 20 JAN 2022

#include <stdio.h>

#include <conio.h>

int main ()

{

char str[30];

int i;

printf (" Enter the string: ");

scanf (" %s", &str); // take a string

// use for loop to change string from upper case to lower case

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

{

// The ASCII value of A is 65 and Z is 90

if (str[i] >= 65 && str[i] <= 90)

str[i] = str[i] + 32; /\* add 32 to string character to convert into lower case. \*/

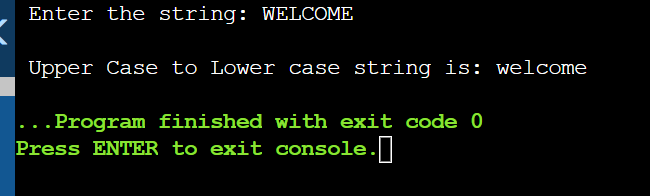
}

printf (" \n Upper Case to Lower case string is: %s", str);

return 0;

}

**SAMPLE OUTPUT**



**PROBLEM 1.4L:** FIND THE SUM OF ROWS AND COLUMNS OF MATRIX OF GIVEN ORDER (ROW X COLUMN)

1. **SOLUTION APPROACH:** Declare and initialize a two-dimensional array a.
2. Calculate the number of rows and columns present in the array a and store it in variables rows and cols respectively.
3. Maintain two variables sumRow and sumCol to store the sum of elements in the specific row and the sum of elements in specific column respectively.
4. To calculate the sum of elements in each row:
   1. Two loops will be used to traverse the array where the outer loop selects a row, and the inner loop represents the columns present in the matrix a.
   2. Calculate the sum by adding elements present in a row.
   3. Display sumRow.
   4. Repeat this for each row.
5. To calculate the sum of elements in each column:
   1. Two loops will be used to traverse the array where the outer loop select a column, and the inner loop represents the rows present in the matrix a.
   2. Calculate the sum by adding elements present in a column.
   3. Display sumCol.
   4. Repeat this for each column.

**SOURCE CODE:**

//MUSKAN SINGH

//20 JAN 2022

#include <stdio.h>

int main()

{

int rows, cols, sumRow, sumCol;

//Initialize matrix a

int a[][3] = {

{1, 2, 3},

{4, 5, 6},

{7, 8, 9}

};

//Calculates number of rows and columns present in given matrix

rows = (sizeof(a)/sizeof(a[0]));

cols = (sizeof(a)/sizeof(a[0][0]))/rows;

//Calculates sum of each row of given matrix

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

sumRow = 0;

for(int j = 0; j < cols; j++){

sumRow = sumRow + a[i][j];

}

printf("Sum of %d row: %d\n", (i+1), sumRow);

}

//Calculates sum of each column of given matrix

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

sumCol = 0;

for(int j = 0; j < rows; j++){

sumCol = sumCol + a[j][i];

}

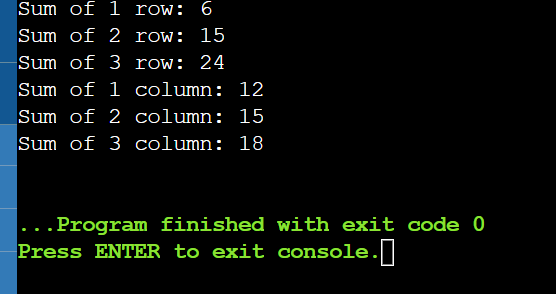
printf("Sum of %d column: %d\n", (i+1), sumCol);

}

return 0;

}

**SAMPLE OUTPUT:**

****

**PROBLEM 1.5L:**FIND THE PRODUCT OF TWO MATRICES USING POINTERS.

**SOLUTION APPROACH**: In array notation to multiply two matrix we use sum += A[row][i] \* B[i][col]; which in pointer notation is equivalent to sum += (\*(\*(A + row) + i)) \* (\*(\*(B + i) + col));

**SOURCE CODE:**

/\*\*

\* C program to multiply two matrix using pointers

\*/

//MUSKAN SINGH

//DATE 20 JAN 2022

#include <stdio.h>

#define ROW 3

#define COL 3

/\* Function declarations \*/

void matrixInput(int mat[][COL]);

void matrixPrint(int mat[][COL]);

void matrixMultiply(int mat1[][COL], int mat2[][COL], int res[][COL]);

int main()

{

int mat1[ROW][COL];

int mat2[ROW][COL];

int product[ROW][COL];

/\*

\* Input elements in matrices.

\*/

printf("Enter elements in first matrix of size %dx%d\n", ROW, COL);

matrixInput(mat1);

printf("Enter elements in second matrix of size %dx%d\n", ROW, COL);

matrixInput(mat2);

// Call function to multiply both matrices

matrixMultiply(mat1, mat2, product);

// Print product of both matrix

printf("Product of both matrices is : \n");

matrixPrint(product);

return 0;

}

/\*\*

\* Function to input elements in matrix from user.

\*

\* @mat Two-dimensional array to store user input.

\*/

void matrixInput(int mat[][COL])

{

int row, col;

for (row = 0; row < ROW; row++)

{

for (col = 0; col < COL; col++)

{

scanf("%d", (\*(mat + row) + col));

}

}

}

/\*\*

\* Function to print elements in a two-dimensional array.

\*

\* @mat Two-dimensional array to print.

\*/

void matrixPrint(int mat[][COL])

{

int row, col;

for (row = 0; row < ROW; row++)

{

for (col = 0; col < COL; col++)

{

printf("%d ", \*(\*(mat + row) + col));

}

printf("\n");

}

}

/\*\*

\* Function to multiply two matrices.

\*

\* @mat1 First matrix

\* @mat2 Second matrix

\* @res Resultant matrix to store product of both matrices.

\*/

void matrixMultiply(int mat1[][COL], int mat2[][COL], int res[][COL])

{

int row, col, i;

int sum;

for (row = 0; row < ROW; row++)

{

for (col = 0; col < COL; col++)

{

sum = 0;

/\*

\* Find sum of product of each elements of

\* rows of first matrix and columns of second

\* matrix.

\*/

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

{

sum += (\*(\*(mat1 + row) + i)) \* (\*(\*(mat2 + i) + col));

}

/\*

\* Store sum of product of row of first matrix

\* and column of second matrix to resultant matrix.

\*/

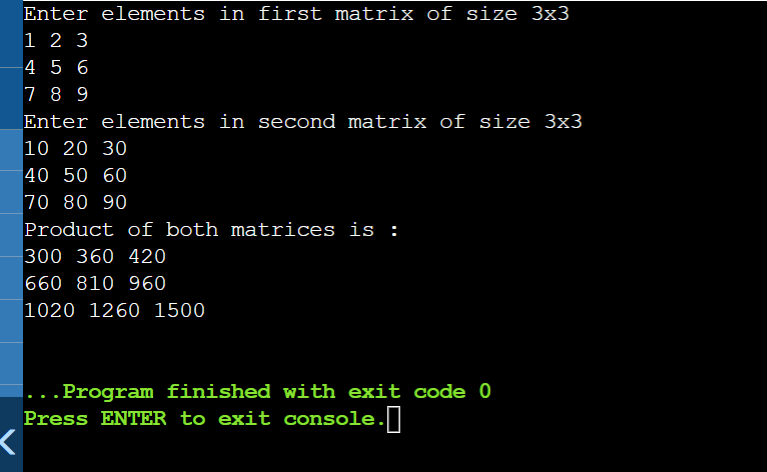
\*(\*(res + row) + col) = sum;

}

}

}

**SAMPLE OUTPUT**

****

**PROBLEM 1.6L:** STORE N NUMBERS(INTEGERS OR REAL) IN AN ARRAY. CONDUCT A LINEAR SEARCH FOR A GIVEN NUMBER AND REPORT SUCCESS OR FAILURE IN THE FORM OF A SUITABLE MESSAGE.

**SOLUTION APPROACH:** Start from the leftmost element of arr[] and one by one compare x with each element of arr[]. If x matches with an element, return the index.If x doesn’t match with any of elements, return -1.

**SOURCE CODE:**

// C code to linearly search x in arr[]. If x

// is present then return its location, otherwise

// return -1

#include <stdio.h>

int search(int arr[], int n, int x)

{

int i;

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

if (arr[i] == x)

return i;

return -1;

}

// Driver code

int main(void)

{

int arr[] = { 2, 3, 4, 10, 40 };

int x = 10;

int n = sizeof(arr) / sizeof(arr[0]);

// Function call

int result = search(arr, n, x);

(result == -1)

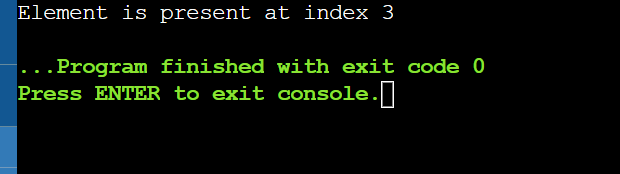
? printf("Element is not present in array")

: printf("Element is present at index %d", result);

return 0;

}

**SAMPLE OUTPUT:**

****

**PRACTICE ACTIVITIES OF EXPERIMENT 1**

**PROBLEM 1.1P:** FIND THE TRANSPOSE AND INVERSE OF A MATRIX.

**SOLUTION APPROACH:** The transpose of a matrix is a new matrix that is obtained by exchanging the rows and columns.

In this program, the user is asked to enter the number of rows r and columns c. Their values should be less than 10 in this program.

Then, the user is asked to enter the elements of the matrix (of order r\*c).

The program below then computes the transpose of the matrix and prints it on the screen.

**SOURCE CODE:**

#include <stdio.h>

int main() {

int a[10][10], transpose[10][10], r, c;

printf("Enter rows and columns: ");

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

// asssigning elements to the matrix

printf("\nEnter matrix elements:\n");

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

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

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

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

}

// printing the matrix a[][]

printf("\nEntered matrix: \n");

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

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

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

if (j == c - 1)

printf("\n");

}

// computing the transpose

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

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

transpose[j][i] = a[i][j];

}

// printing the transpose

printf("\nTranspose of the matrix:\n");

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

for (int j = 0; j < r; ++j) {

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

if (j == r - 1)

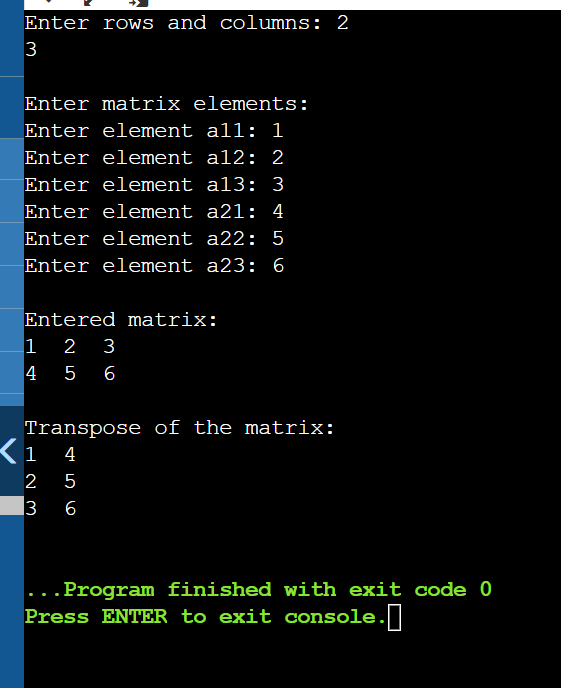
printf("\n");

}

return 0;

}

**SAMPLE OUTPUT:**

****

**INVERSE OF A MATRIX:**

**SOLUTION APPROACH:** In order to find the inverse of a matrix,

* The matrix must be a ***square matrix***.
* Determinant needs to be calculated and should not equal to zero (0).
* Then find the adjoint of a matrix and
* Lastly, multiply ***1/determinant by adjoint*** to get the inverse of a matrix.

**SOURCE CODE**

#include<stdio.h>

#include<math.h>

//function prototype that are being created

void cofactor(float [][25], float);

float determinant(float [][25], float);

void transpose(float [][25], float [][25], float);

int main()

{

float a[25][25], n, d;

int i, j;

printf("Enter the order of the Matrix: ");

scanf("%f", &n);

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

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

{

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

{

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

}

}

d = determinant(a, n);

if (d == 0)

printf("Since the determinant is zerp (0), therefor inverse is not possible.");

else

cofactor(a, n);

}

// function for the calculation of determinant

float determinant(float a[25][25], float k)

{

float s = 1, det = 0, b[25][25];

int i, j, m, n, c;

if (k == 1)

{

return (a[0][0]);

}

else

{

det = 0;

for (c = 0; c < k; c++)

{

m = 0;

n = 0;

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

{

for (j = 0 ;j < k; j++)

{

b[i][j] = 0;

if (i != 0 && j != c)

{

b[m][n] = a[i][j];

if (n < (k - 2))

n++;

else

{

n = 0;

m++;

}

}

}

}

det = det + s \* (a[0][c] \* determinant(b, k - 1));

s = -1 \* s;

}

}

return (det);

}

// function for cofactor calculation

void cofactor(float num[25][25], float f)

{

float b[25][25], fac[25][25];

int p, q, m, n, i, j;

for (q = 0;q < f; q++)

{

for (p = 0;p < f; p++)

{

m = 0;

n = 0;

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

{

for (j = 0;j < f; j++)

{

if (i != q && j != p)

{

b[m][n] = num[i][j];

if (n < (f - 2))

n++;

else

{

n = 0;

m++;

}

}

}

}

fac[q][p] = pow(-1, q + p) \* determinant(b, f - 1);

}

}

transpose(num, fac, f);

}

///function to find the transpose of a matrix

void transpose(float num[25][25], float fac[25][25], float r)

{

int i, j;

float b[25][25], inverse[25][25], d;

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

{

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

{

b[i][j] = fac[j][i];

}

}

d = determinant(num, r);

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

{

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

{

inverse[i][j] = b[i][j] / d;

}

}

printf("\nThe inverse of matrix: \n");

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

{

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

{

printf("\t%f", inverse[i][j]);

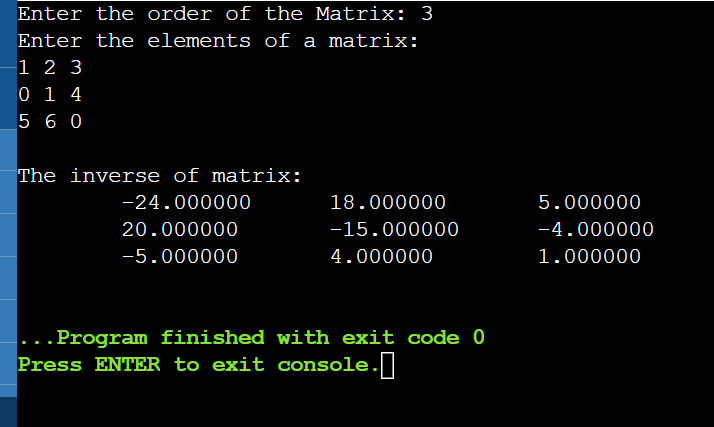
}

printf("\n");

}

}

**SAMPLE OUTPUT**

****

**PROBLEM 1.2P:** FIND IF THE GIVEN MATRIX OF ORDER (m x n) IS A SPARSE MATRIX OR NOT. [ ASSUME THAT A MATRIX CAN BECOME A SPARSE MATRIX IF MORE THAN HALF THE TOTAL NUMBER OF ITS ELEMENTS HAVE THE VALUE ZERO.]

**SOLUTION APPROACH:** To check whether a matrix is a sparse matrix, we only need to check the total number of elements that are equal to zero. If this count is more than (m \* n)/2, we return true.

**SOURCE CODE:**

//MUSKAN SINGH

//DATE 20 JAN 2022

#include <stdio.h>

int main()

{

int rows, cols, size, count = 0;

//Initialize matrix a

int a[][3] = {

{4, 0, 0},

{0, 5, 0},

{0, 0, 6}

};

//Calculates number of rows and columns present in given matrix

rows = (sizeof(a)/sizeof(a[0]));

cols = (sizeof(a)/sizeof(a[0][0]))/rows;

//Calculates the size of array

size = rows \* cols;

//Count all zero element present in matrix

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

for(int j = 0; j < cols; j++){

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

count++;

}

}

if(count > (size/2))

printf("Given matrix is a sparse matrix");

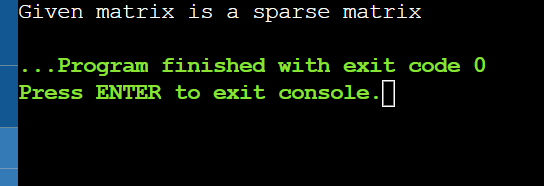
else

printf("Given matrix is not a sparse matrix");

return 0;

}

**SAMPLE OUTPUT**

****

**PROBLEM 1.3P:** STORE N NUMBERS IN AN ARRAY IN ASCENDING OR DESCENDING ORDER. CONDUCT A BINARY SEARCH FOR A GIVEN NUMBER AND REPORT SUCCESS OR FAILURE IN THE FORM OF SUITABLE MESSAGE.

**SOLUTION APPROACH:** We basically ignore half of the elements just after one comparison.

1. Compare x with the middle element.
2. If x matches with the middle element, we return the mid index.
3. Else If x is greater than the mid element, then x can only lie in the right half subarray after the mid element. So we recur for the right half.
4. Else (x is smaller) recur for the left half.

**SOURCE CODE**

// C program to implement recursive Binary Search

//MUSKAN SINGH

//DATE 20 JAN 2022

#include <stdio.h>

// A recursive binary search function. It returns

// location of x in given array arr[l..r] is present,

// otherwise -1

int binarySearch(int arr[], int l, int r, int x)

{

if (r >= l) {

int mid = l + (r - l) / 2;

// If the element is present at the middle

// itself

if (arr[mid] == x)

return mid;

// If element is smaller than mid, then

// it can only be present in left subarray

if (arr[mid] > x)

return binarySearch(arr, l, mid - 1, x);

// Else the element can only be present

// in right subarray

return binarySearch(arr, mid + 1, r, x);

}

// We reach here when element is not

// present in array

return -1;

}

int main(void)

{

int arr[] = { 2, 3, 4, 10, 40 };

int n = sizeof(arr) / sizeof(arr[0]);

int x = 10;

int result = binarySearch(arr, 0, n - 1, x);

(result == -1)

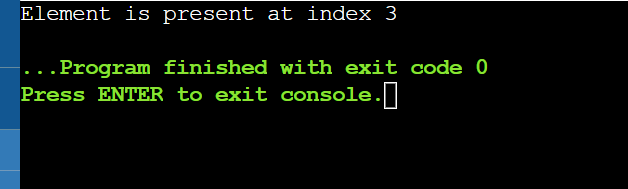
? printf("Element is not present in array")

: printf("Element is present at index %d", result);

return 0;

}

**SAMPLE OUTPUT:**

****

**PROBLEM 1.4P:** FIND OUT THE LARGEST AND SMALLEST NUMBER IN A GIVEN ARRAY**.**

**SOLUTION APPROACH:**

1. Input the array elements.
2. Initialize small = large = arr[0]
3. Repeat from i = 2 to n
4. if(arr[i] > large)
5. large = arr[i]
6. if(arr[i] < small)
7. small = arr[i]
8. Print small and large.

**SOURCE CODE:**

//MUSKAN SINGH

//DATE 20 JAN 2022

#include<stdio.h>

int main()

{

int a[50],i,n,large,small;

printf("How many elements:");

scanf("%d",&n);

printf("Enter the Array:");

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

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

large=small=a[0];

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

{

if(a[i]>large)

large=a[i];

if(a[i]<small)

small=a[i];

}

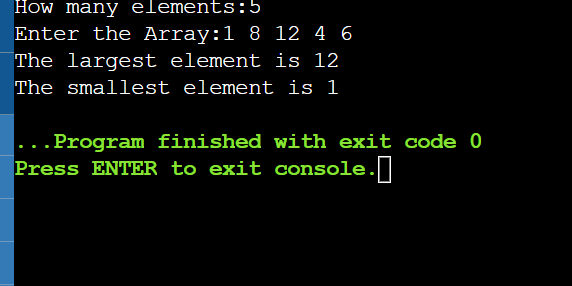
printf("The largest element is %d",large);

printf("\nThe smallest element is %d",small);

return 0;

}

**SAMPLE OUTPUT:**

****

**PROBLEM 1.5P:** CREATE AN ARRAY A1 WITH N ELEMENTS.

* 1. COPY ALL ELEMENTS OF A1 INTO ANOTHER ARRAY A2 USING POINTERS. DISPLAY THE CONTENTS OF BOTH THE ARRAYS USING POINTERS**.**

**SOLUTION APPROACH:** Step by step descriptive logic to copy one array to another using pointers.

1. Input size and elements in first array, store it in some variable say size and source\_array.
2. Declare another array say dest\_array to store copy of source\_array.
3. Declare a pointer to source\_array say \*source\_ptr = source\_array and one more pointer to dest\_array say \*dest\_ptr = dest\_array.
4. Copy elements from source\_ptr to desc\_ptr using \*desc\_ptr = \*source\_ptr.
5. Increment pointers source\_ptr and desc\_ptr by 1.
6. Repeat step 3 and 4 till source\_ptr exists in source\_arr memory range.

**SOURCE CODE:**

/\*\*

\* C program to copy an array to another array using pointers

\*/

//MUSKAN SINGH

//DATE 20 JAN 2022

#include <stdio.h>

#define MAX\_SIZE 100 // Maximum array size

/\* Function declaration to print array \*/

void printArray(int arr[], int size);

int main()

{

int source\_arr[MAX\_SIZE], dest\_arr[MAX\_SIZE];

int size, i;

int \*source\_ptr = source\_arr; // Pointer to source\_arr

int \*dest\_ptr = dest\_arr; // Pointer to dest\_arr

int \*end\_ptr;

/\*

\* Input size and elements in source array

\*/

printf("Enter size of array: ");

scanf("%d", &size);

printf("Enter elements in array: ");

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

{

scanf("%d", (source\_ptr + i));

}

// Pointer to last element of source\_arr

end\_ptr = &source\_arr[size - 1];

/\* Print source and destination array before copying \*/

printf("\nSource array before copying: ");

printArray(source\_arr, size);

printf("\nDestination array before copying: ");

printArray(dest\_arr, size);

/\*

\* Run loop till source\_ptr exists in source\_arr

\* memory range.

\*/

while(source\_ptr <= end\_ptr)

{

\*dest\_ptr = \*source\_ptr;

// Increment source\_ptr and dest\_ptr

source\_ptr++;

dest\_ptr++;

}

/\* Print source and destination array after copying \*/

printf("\n\nSource array after copying: ");

printArray(source\_arr, size);

printf("\nDestination array after copying: ");

printArray(dest\_arr, size);

return 0;

}

/\*\*

\* Function to print array elements.

\*

\* @arr Integer array to print.

\* @size Size of array.

\*/

void printArray(int \*arr, int size)

{

int i;

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

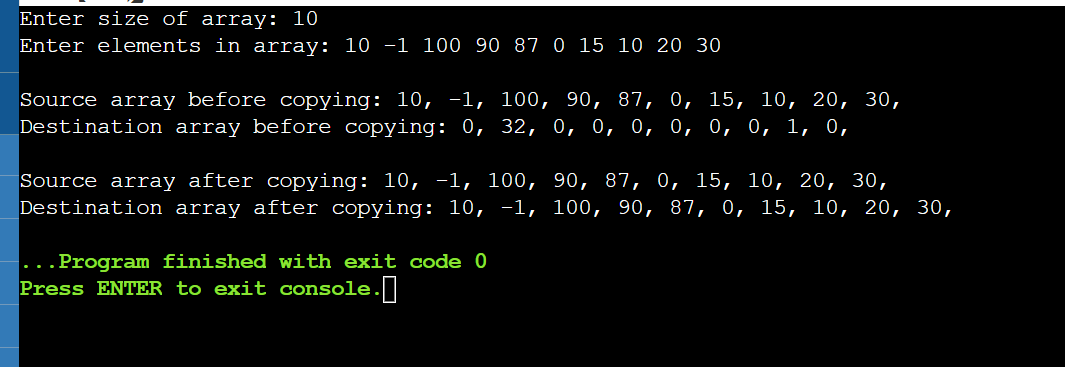
{

printf("%d, ", \*(arr + i));

}

}

**SAMPLE OUTPUT:**

****

* 1. MERGE THE CONTENTS OF A1 AND A2 INTO A NEW ARRAY A3

**SOLUTION APPROACH:**

1. Create an array arr3[] of size n1 + n2.

2.Copy all n1 elements of arr1[] to arr3[]

3.Traverse arr2[] and one by one insert elements of arr3[] to arr1[].

**SOURCE CODE:**

/\*

\* C Program to Merge the Elements of 2 Sorted Array

\*/

#include <stdio.h>

void main()

{

int array1[50], array2[50], array3[100], m, n, i, j, k = 0;

printf("\n Enter size of array Array 1: ");

scanf("%d", &m);

printf("\n Enter sorted elements of array 1: \n");

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

{

scanf("%d", &array1[i]);

}

printf("\n Enter size of array 2: ");

scanf("%d", &n);

printf("\n Enter sorted elements of array 2: \n");

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

{

scanf("%d", &array2[i]);

}

i = 0;

j = 0;

while (i < m && j < n)

{

if (array1[i] < array2[j])

{

array3[k] = array1[i];

i++;

}

else

{

array3[k] = array2[j];

j++;

}

k++;

}

if (i >= m)

{

while (j < n)

{

array3[k] = array2[j];

j++;

k++;

}

}

if (j >= n)

{

while (i < m)

{

array3[k] = array1[i];

i++;

k++;

}

}

printf("\n After merging: \n");

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

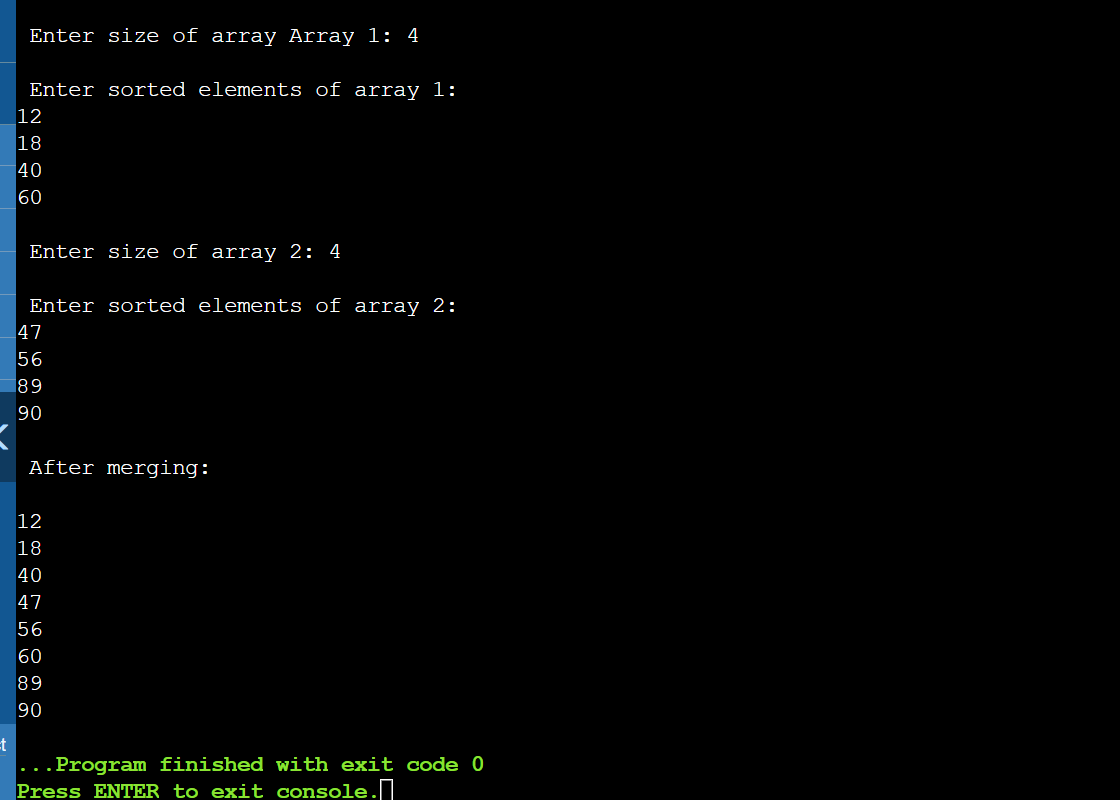
{

printf("\n%d", array3[i]);

}

}

**SAMPLE OUTPUT:**

****

**PROBLEM 1.6P:** FIND THE TOTAL NUMBER OF ALPHABETS, DIGITS OR SPECIAL CHARACTERS IN A STRING.

**SOLUTION APPROACH:**

The ASCII values range of A to Z is 65 to 90, and a to z is 97 to 122 and 0 t0 9 is 48 to 57.Read the entered string and initialize to s. Iterate through string using for loop with the structure for(i=0;s[i];i++)If the ASCII value of s[i] is in the range between 65 to 90 or 97 to 122 then increase the alphabets count. If the ASCII value of s[i] is in the range between 48 to 57 then increase the digits count. If the ASCII value of s[i] is not in the range of the above two conditions then increase the special characters count.Print the alphabets count, digits count and special characters count.

**SOURCE CODE:**

#include <stdio.h>

#include <string.h>

int main()

{

char s[1000];

int i,alphabets=0,digits=0,specialcharacters=0;

printf("Enter the string : ");

gets(s);

for(i=0;s[i];i++)

{

if((s[i]>=65 && s[i]<=90)|| (s[i]>=97 && s[i]<=122) )

alphabets++;

else if(s[i]>=48 && s[i]<=57)

digits++;

else

specialcharacters++;

}

printf("Alphabets = %d\n",alphabets);

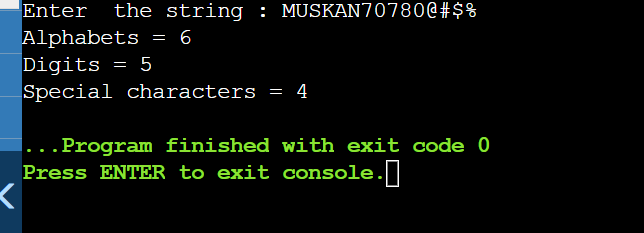
printf("Digits = %d\n",digits);

printf("Special characters = %d", specialcharacters);

return 0;

}

**SAMPLE OUTPUT:**

****

**PROBLEM 1.7P:** FIND WHETHER THE ENTERED STRING IS PALINDROME OR NOT.

**SOLUTION APPROACH:**

1. Find length of str. Let length be n.   
   2) Initialize low and high indexes as 0 and n-1 respectively.   
   3) Do following while low index ‘l’ is smaller than high index ‘h’.   
   …a) If str[l] is not same as str[h], then return false.   
   …b) Increment l and decrement h, i.e., do l++ and h–.   
   4) If we reach here, it means we didn’t find a mis  
   Following is C implementation to check if a given string is palindrome or not.

**SOURCE CODE:**

//MUSKAN SINGH

//DATE 20 JAN 2022

#include <stdio.h>

#include <string.h>

// A function to check if a string str is palindrome

void isPalindrome(char str[])

{

// Start from leftmost and rightmost corners of str

int l = 0;

int h = strlen(str) - 1;

// Keep comparing characters while they are same

while (h > l)

{

if (str[l++] != str[h--])

{

printf("%s is not a palindrome\n", str);

return;

}

}

printf("%s is a palindrome\n", str);

}

// Driver program to test above function

int main()

{

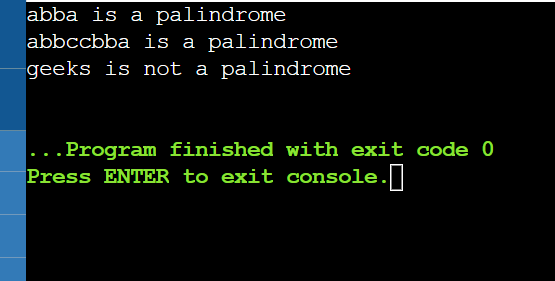
isPalindrome("abba");

isPalindrome("abbccbba");

isPalindrome("geeks");

return 0;

**SAMPLE OUTPUT:**

****

**PROBLEM 1.8P:** COUNT THE NUMBER OF WORD IN A STRING.

**SOLUTION APPROACH:** The idea is to maintain two states: IN and OUT. The state OUT indicates that a separator is seen. State IN indicates that a word character is seen. We increment word count when previous state is OUT and next character is a word character.

**SOURCE CODE:**

//MUSKAN SINGH

//DATE 20 JAN 2022

/\* C program to count no of words from given input string. \*/

#include <stdio.h>

#define OUT 0

#define IN 1

// returns number of words in str

unsigned countWords(char \*str)

{

int state = OUT;

unsigned wc = 0; // word count

// Scan all characters one by one

while (\*str)

{

// If next character is a separator, set the

// state as OUT

if (\*str == ' ' || \*str == '\n' || \*str == '\t')

state = OUT;

// If next character is not a word separator and

// state is OUT, then set the state as IN and

// increment word count

else if (state == OUT)

{

state = IN;

++wc;

}

// Move to next character

++str;

}

return wc;

}

// Driver program to tes above functions

int main(void)

{

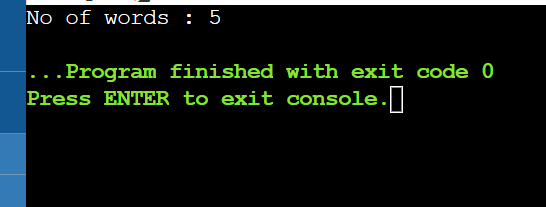
char str[] = "One two three\n four\tfive ";

printf("No of words : %u", countWords(str));

return 0;

}

**SAMPLE OUTPUT:**

****

**PROBLEM 1.11P:** ADD TWO COMPLEX NUMBERS BY PASSING STRUCTURE TO A FUNCTION AS ARGUMENT.

**SOLUTION APPROACH:** In order to add two complex numbers in C programming language, the user has to take two complex numbers as structure members and perform addition operation on those two numbers by creating a user-defined function

**SOURCE CODE:**

#include <stdio.h>

typedef struct complex{

float real;

float imag;

} complex;

complex addition(complex num1, complex num2);

int main(){

complex num1, num2, value;

printf("entering real and imag parts of first complex no:\n ");

scanf("%f %f", &num1.real, &num1.imag);

printf("entering real and imag parts of second complex no:\n ");

scanf("%f %f", &num2.real, &num2.imag);

value= addition(num1, num2);

printf("result = %.1f + %.1fi", value.real, value.imag);

return 0;

}

complex addition(complex num1, complex num2){

complex temp;

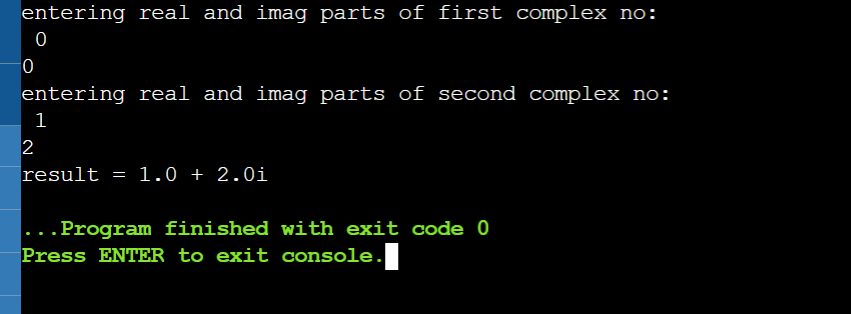
temp.real = num1.real + num2.real;

temp.imag = num1.imag + num2.imag;

return (temp);

}

**SAMPLE OUTPUT:**

****

**EXPERIMENT - 02**

**PROBLEM 2.1L:**DESIGN A STRUCTURE ‘SUBJECT’ TO STORE THE DETAILS OF THE SUBJECTS LIKE SUBJECT NAME AND SUBJECT CODE. USING STRUCTURE POINTERS ALLOCATE MEMORY FOR THE STRUCTURE DYNAMICALLY SO AS TO OBTAIN DETAILS OF ‘N’ SUBJECTS USING FOR LOOP.

**SOLUTION APPROACH:** WE WILL CREATE THE POINTER P OF STRUCT SUBJECT DATA TYPE AND DYNAMICALLY ALLOCATE ITS MEMORY USING malloc() FUNCTION**.**

**SOURCE CODE:**

#include <stdio.h>

#include <stdlib.h>

struct subject {

char subject\_name[50];

int code;

};

int main() {

struct subject \*ptr;

int noOfRecords;

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

scanf("%d", &noOfRecords);

// Memory allocation for noOfRecords structures

ptr = (struct subject \*)malloc(noOfRecords \* sizeof(struct subject));

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

printf("Enter subject and code:\n");

scanf("%s %d", (ptr + i)->subject\_name, &(ptr + i)->code);

}

printf("Displaying Information:\n");

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

printf("%s\t%d\n", (ptr + i)->subject\_name, (ptr + i)->code);

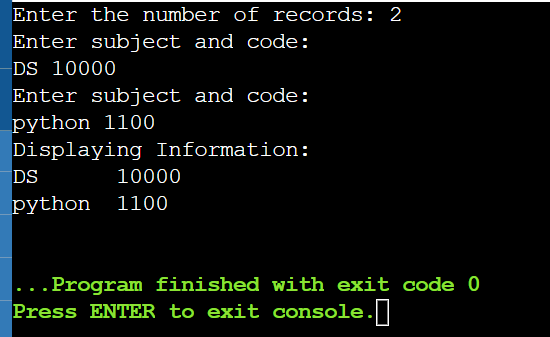
}

free(ptr);

return 0;

}

**OUTPUT:**

****

**PROBLEM 2.2L:** CREATE AN INTEGER ARRAY OF USER DEFINED SIZE N1 WITH DYNAMIC MEMORY ALLOCATION. STORE DATA AFTER READING FROM KEYBOARD. EXPAND THE SIZE OF ARRAY WITH N2. READ NEW VALUES (N2 VALUES FROM KEYBOARD). PRINT STATE OF ARRAY WITH ALL (NI+N2) VALUES**.**

**SOLUTION APPROACH**: MAKE USE OF MALLOC TO DYMANICALLY ALLOCATE THE MEMORY TO N1 AND THEN USE REALLOC TO EXPAND THE SIZE OF ARRAY TO N2**.**

**SOURCE CODE:**

//MUSKAN SINGH

//DATE 20 JAN 2022

#include<stdio.h>

#include<stdlib.h>

int main()

{

int \*ptr;

int n1,i;

printf("enter no. of elements");

scanf("%d",&n1);

printf("entered no. of elements: %d\n", n1);

ptr=(int \*)malloc(n1\*sizeof(int));

if(ptr==NULL)

{

printf("memory not allocated\n");

exit(0);

}

else{

printf("memory succeefully allocated");

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

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

}

}

int n2;

printf("enter no. of elements");

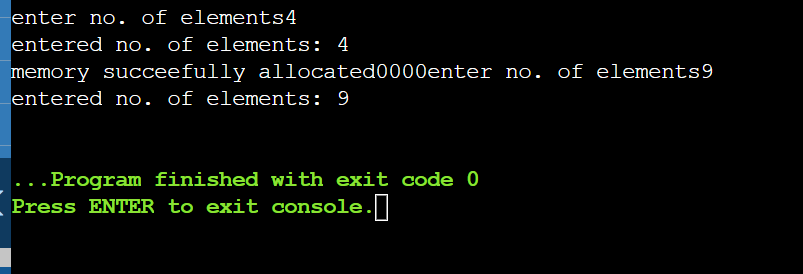
scanf("%d",&n2);

printf("entered no. of elements: %d\n", n2);

ptr=(int \*)realloc(n1,n2);

}

**OUTPUT:**

****

**EXPERIMENT-3**

**TITLE**: LINK LIST DATA STRUCTURE AND ITS APPLICATIONS

**OBJECTIVE**: TO EXPERIMENT THE CONCEPT OF POINTERS, STRUCTURE AND DYNAMIC MEMORY ALLOCATION TO REALIZE LINKED LIST AND ITS APPLICATION**.**

**PROBLEM 3.1L:** IMPLEMENT SINGLE LINKED LIST DATA STRUCTURE AND ITS OPERATIONS LIKE INSERT AND DELETE IN THE BEGINNING, END AND NTH POSITION OF THE LIST AND DISPLAY THE ITEMS STORED IN THE LINKED LIST.

**SOLUTION APPROACH**: MAKE VARIOUS FUNCTIONS AND USE SWITCH CASES

**SOURCE CODE:**

#include<stdio.h>

#include<stdlib.h>

struct node{

int data;

struct node \*link;

};

struct node \* create\_node(){

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

}

void display(struct node \*start){ //traversing

struct node \*head=start;

if (head==NULL){

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

return;

}

while (head!=NULL){

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

head=head->link;

}

}

struct node \* insert\_beg(struct node \*start, int ins){ //updating the start value by new node

if (start==NULL){

start=create\_node();

start->data=ins;

start->link=NULL;

return start;

}

struct node \*new=create\_node();

new->link=start;

start=new;

new->data=ins;

return start;

}

struct node \* insert\_end(struct node \*start, int ins){ //find the last node and update the link of last node with new node

struct node \*new=create\_node();

struct node \*temp=start;

if (start==NULL){

start=create\_node();

start->data=ins;

start->link=NULL;

return start;

}

while (temp->link!=NULL)

temp=temp->link;

new->data=ins;

new->link=NULL;

temp->link=new;

return start;

}

struct node \* insert\_pos(struct node \*start,int ins, int pos){ //if pos is 1 it means we are talking about starting element. So this is a case

struct node \*new=create\_node(); //of inserting in beginning.

struct node \*temp=start;

if (pos==1){

start=insert\_beg(start,ins);

return start;

}else{

int loc=1;

while (temp->link!=NULL){ //Find the position of delete and also the location of the node

loc=loc+1;

if (pos==loc)

break;

temp=temp->link;

}

if (temp->link==NULL){ //while traversing if element is not found it means the postion to insert is not present

printf("Position is out of range. Number of element of linked list is %d.\n",loc);

return start;

}

new->data=ins; //updating the link of pervious node with new node and new node with previous node

new->link=temp->link;

temp->link=new;

return start;

}

}

struct node \* del\_beg(struct node \*start){ //Making start value point from second node

struct node \*temp=start;

if (start==NULL){

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

return NULL;

}

start=start->link;

printf("\nData of deleted node is %d.\n",temp->data);

return start;

}

struct node \* del\_end(struct node \*start){ //Find end node then update the pointer of previous node with null

struct node \*temp=start;

struct node \*ptr;

if (start==NULL){

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

return NULL;

}

if (temp->link==NULL){

printf("\nData of deleted node is %d.\n",temp->data);

start=NULL;

return start;

}

while (temp->link!=NULL){

ptr=temp;

temp=temp->link;

}

ptr->link=NULL;

printf("\nData of deleted node is %d.\n",temp->data);

return start;

}

struct node \* del\_pos(struct node \*start,int pos){

struct node \*temp=start;

int loc=1;

struct node \*ptr; //Find the location and find the pointer to the location and the previous

if (pos==1){

start=del\_beg(start);

return start;

}else{

while (temp->link!=NULL){

loc=loc+1;

ptr=temp;

temp=temp->link;

if (pos==loc) //Finds the pointers

break;

}

if (temp->link==NULL){

if (pos>loc)

printf("Position is out of range. Number of element of linked list is %d.\n",loc);

else

start=del\_end(start);

return start;

}

printf("\nData of deleted node is %d.\n",temp->data);

ptr->link=temp->link; //updates the values

return start;

}

}

struct node \* del\_val(struct node \*start,int val){

struct node \*temp=start;

int count=0;

while (temp!=NULL){ //find position of the value and call del\_pos function

count=count+1;

if (temp->data==val){

start=del\_pos(start,count);

return start;

}

temp=temp->link;

}

if (temp==NULL) //if temp is null it means value was not found

printf("Value doesn't exist.\n");

return start;

}

struct node \* del(struct node \*start){ //ask if user wants to delete position or value.

int choice;

struct node \*temp=start;

if (temp==NULL){

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

return NULL;

}

printf("Do you want to delete position[0] or value[1]: ");

scanf("%d",&choice);

if (choice==0){ //Pick a choice

int pos;

printf("Enter position to delete: ");

scanf("%d",&pos);

start=del\_pos(start,pos); //calls function to delete location

}else if (choice==1){

int val;

printf("Enter value to delete: ");

scanf("%d",&val);

start=del\_val(start,val); //call function to delete value

}else

printf("Enter a valid choice.\n");

return start;

}

void main(){

struct node \*START=NULL;

int count=0;

do{

int choice;

printf("[1 to insert at start]\n");

printf("[2 to insert at end]\n");

printf("[3 to insert at any position]\n");

printf("[4 to delete at start]\n");

printf("[5 to delete at end]\n");

printf("[6 to delete any position/value]\n");

printf("[7 to display]\nEnter your choice: ");

scanf("%d",&choice);

int ins,pos;

switch(choice){

case 1:

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

scanf("%d",&ins);

insert\_beg(START,ins);

break;

//

case 2:

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

scanf("%d",&ins);

START=insert\_end(START,ins);

break;

//

case 3:

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

scanf("%d",&ins);

printf("Enter position to insert: ");

scanf("%d",&pos);

START=insert\_pos(START,ins,pos);

break;

//

case 4:

START=del\_beg(START);

break;

//

case 5:

START=del\_end(START);

break;

//

case 6:

START=del(START);

break;

//

case 7:

display(START);

break;

//

default:

printf("Enter a valid choice.\n");

}

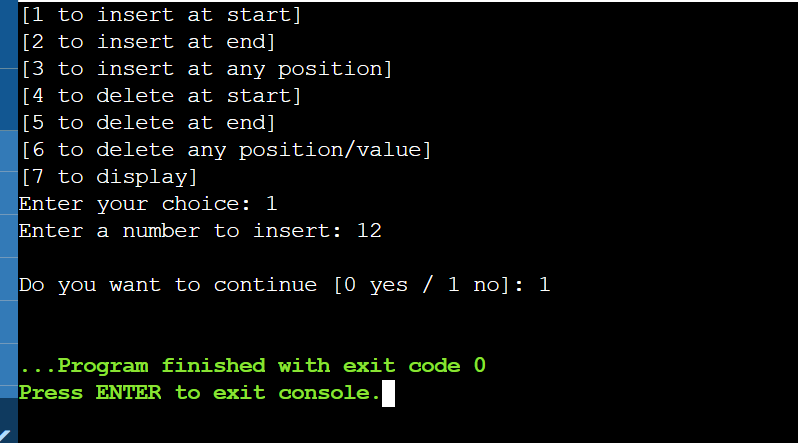
printf("\nDo you want to continue [0 yes / 1 no]: ");

scanf("%d",&count);

}while(count==0);

}

**SAMPLE OUTPUT:**

****

**PROBLEM 3.2L:** USING SINGLE LINKED LIST AND FUNCTIONS IMPLEMENT STACK AND ITS OPERATIONS LIKE INSERT, DELETE AND DISPLAY.

**SOLUTION APPROACH: FOR PUSHING:**

1. Create a node first and allocate memory to it.
2. If the list is empty then the item is to be pushed as the start node of the list. This includes assigning value to the data part of the node and assign null to the address part of the node.
3. If there are some nodes in the list already, then we have to add the new element in the beginning of the list (to not violate the property of the stack). For this purpose, assign the address of the starting element to the address field of the new node and make the new node, the starting node of the list.

**FOR DELETING:**

1. **Check for the underflow condition:** The underflow condition occurs when we try to pop from an already empty stack. The stack will be empty if the head pointer of the list points to null.
2. **Adjust the head pointer accordingly:** In stack, the elements are popped only from one end, therefore, the value stored in the head pointer must be deleted and the node must be freed. The next node of the head node now becomes the head node.

**FOR DISPLAYING:**

1. Copy the head pointer into a temporary pointer.
2. Move the temporary pointer through all the nodes of the list and print the value field attached to every node.

**SOURCE CODE:**

#include <stdio.h>

#include <stdlib.h>

void push();

void pop();

void display();

struct node

{

int val;

struct node \*next;

};

struct node \*head;

void main ()

{

int choice=0;

printf("\n\*\*\*\*\*\*\*\*\*Stack operations using linked list\*\*\*\*\*\*\*\*\*\n");

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

while(choice != 4)

{

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

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

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

scanf("%d",&choice);

switch(choice)

{

case 1:

{

push();

break;

}

case 2:

{

pop();

break;

}

case 3:

{

display();

break;

}

case 4:

{

printf("Exiting....");

break;

}

default:

{

printf("Please Enter valid choice ");

}

};

}

}

void push ()

{

int val;

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

if(ptr == NULL)

{

printf("not able to push the element");

}

else

{

printf("Enter the value");

scanf("%d",&val);

if(head==NULL)

{

ptr->val = val;

ptr -> next = NULL;

head=ptr;

}

else

{

ptr->val = val;

ptr->next = head;

head=ptr;

}

printf("Item pushed");

}

}

void pop()

{

int item;

struct node \*ptr;

if (head == NULL)

{

printf("Underflow");

}

else

{

item = head->val;

ptr = head;

head = head->next;

free(ptr);

printf("Item popped");

}

}

void display()

{

int i;

struct node \*ptr;

ptr=head;

if(ptr == NULL)

{

printf("Stack is empty\n");

}

else

{

printf("Printing Stack elements \n");

while(ptr!=NULL)

{

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

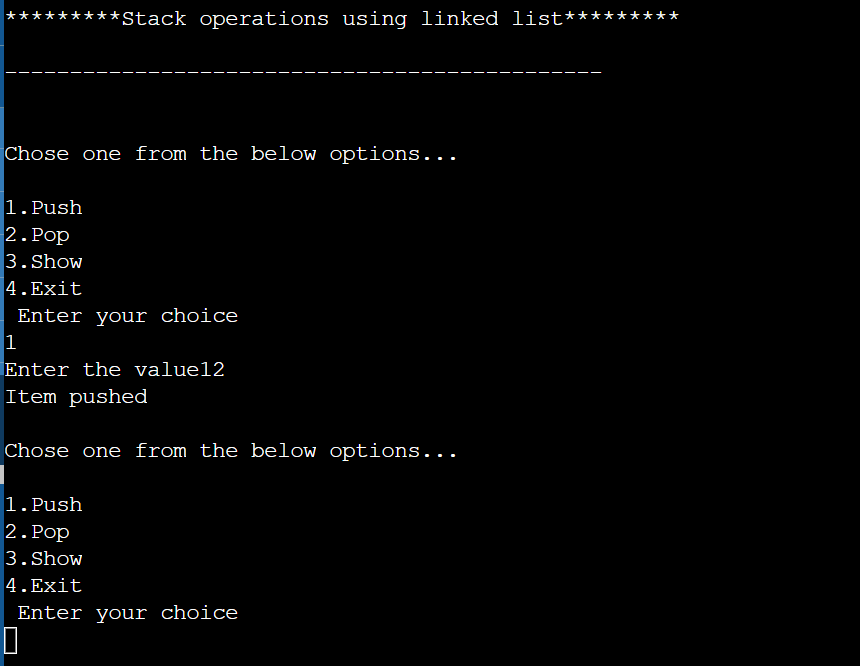
ptr = ptr->next;

}

}

}

**SAMPLE OUTPUT:**

****

**PROBLEM 3.1P:** ADD TWO POLYNOMIALS USING LINKED LIST.

**SOLUTION APPROACH:**

**SOURCE CODE:**

#include<stdio.h>

#include<stdlib.h>

/\*\*

\* The structure for the polynomial

\* This is a linked list with two variable

\* int coeff The Coefficient

\* int pow The power of x

\*/

typedef struct link {

int coeff;

int pow;

struct link \* next;

} my\_poly;

/\*\* The prototypes \*/

void my\_create\_poly(my\_poly \*\*);

void my\_show\_poly(my\_poly \*);

void my\_add\_poly(my\_poly \*\*, my\_poly \*, my\_poly \*);

/\*\*

\* The simple menu driven main function

\*/

int main(void) {

int ch;

do {

my\_poly \* poly1, \* poly2, \* poly3;

printf("\nCreate 1st expression\n");

my\_create\_poly(&poly1);

printf("\nStored the 1st expression");

my\_show\_poly(poly1);

printf("\nCreate 2nd expression\n");

my\_create\_poly(&poly2);

printf("\nStored the 2nd expression");

my\_show\_poly(poly2);

my\_add\_poly(&poly3, poly1, poly2);

my\_show\_poly(poly3);

printf("\nAdd two more expressions? (Y = 1/N = 0): ");

scanf("%d", &ch);

} while (ch);

return 0;

}

/\*\*

\* The create polynomial function

\* @param my\_poly \*\* node The pointer to the head of the polynomial

\* We will modify the parameter and will store the base address

\* @return void

\*/

void my\_create\_poly(my\_poly \*\* node) {

int flag; //A flag to control the menu

int coeff, pow;

my\_poly \* tmp\_node; //To hold the temporary last address

tmp\_node = (my\_poly \*) malloc(sizeof(my\_poly)); //create the first node

\*node = tmp\_node; //Store the head address to the reference variable

do {

//Get the user data

printf("\nEnter Coeff:");

scanf("%d", &coeff);

tmp\_node->coeff = coeff;

printf("\nEnter Pow:");

scanf("%d", &pow);

tmp\_node->pow = pow;

//Done storing user data

//Now increase the Linked on user condition

tmp\_node->next = NULL;

//Ask user for continuation

printf("\nContinue adding more terms to the polynomial list?(Y = 1/N = 0): ");

scanf("%d", &flag);

//printf("\nFLAG: %c\n", flag);

//Grow the linked list on condition

if(flag) {

tmp\_node->next = (my\_poly \*) malloc(sizeof(my\_poly)); //Grow the list

tmp\_node = tmp\_node->next;

tmp\_node->next = NULL;

}

} while (flag);

}

/\*\*

\* The show polynomial function

\* Prints the Polynomial in user readable format

\* @param my\_poly \* node The polynomial linked list

\* @return void

\*/

void my\_show\_poly(my\_poly \* node) {

printf("\nThe polynomial expression is:\n");

while(node != NULL) {

printf("%dx^%d", node->coeff, node->pow);

node = node->next;

if(node != NULL)

printf(" + ");

}

}

/\*\*

\* The polynomial add function

\* Adds two polynomial to a given variable

\* @param my\_poly \*\* result Stores the result

\* @param my\_poly \* poly1 The first polynomial expression

\* @param my\_poly \* poly2 The second polynomial expression

\* @return void

\*/

void my\_add\_poly(my\_poly \*\* result, my\_poly \* poly1, my\_poly \* poly2) {

my\_poly \* tmp\_node; //Temporary storage for the linked list

tmp\_node = (my\_poly \*) malloc(sizeof(my\_poly));

tmp\_node->next = NULL;

\*result = tmp\_node; //Copy the head address to the result linked list

//Loop while both of the linked lists have value

while(poly1 && poly2) {

if (poly1->pow > poly2->pow) {

tmp\_node->pow = poly1->pow;

tmp\_node->coeff = poly1->coeff;

poly1 = poly1->next;

}

else if (poly1->pow < poly2->pow) {

tmp\_node->pow = poly2->pow;

tmp\_node->coeff = poly2->coeff;

poly2 = poly2->next;

}

else {

tmp\_node->pow = poly1->pow;

tmp\_node->coeff = poly1->coeff + poly2->coeff;

poly1 = poly1->next;

poly2 = poly2->next;

}

//Grow the linked list on condition

if(poly1 && poly2) {

tmp\_node->next = (my\_poly \*) malloc(sizeof(my\_poly));

tmp\_node = tmp\_node->next;

tmp\_node->next = NULL;

}

}

//Loop while either of the linked lists has value

while(poly1 || poly2) {

//We have to create the list at beginning

//As the last while loop will not create any unnecessary node

tmp\_node->next = (my\_poly \*) malloc(sizeof(my\_poly));

tmp\_node = tmp\_node->next;

tmp\_node->next = NULL;

if(poly1) {

tmp\_node->pow = poly1->pow;

tmp\_node->coeff = poly1->coeff;

poly1 = poly1->next;

}

if(poly2) {

tmp\_node->pow = poly2->pow;

tmp\_node->coeff = poly2->coeff;

poly2 = poly2->next;

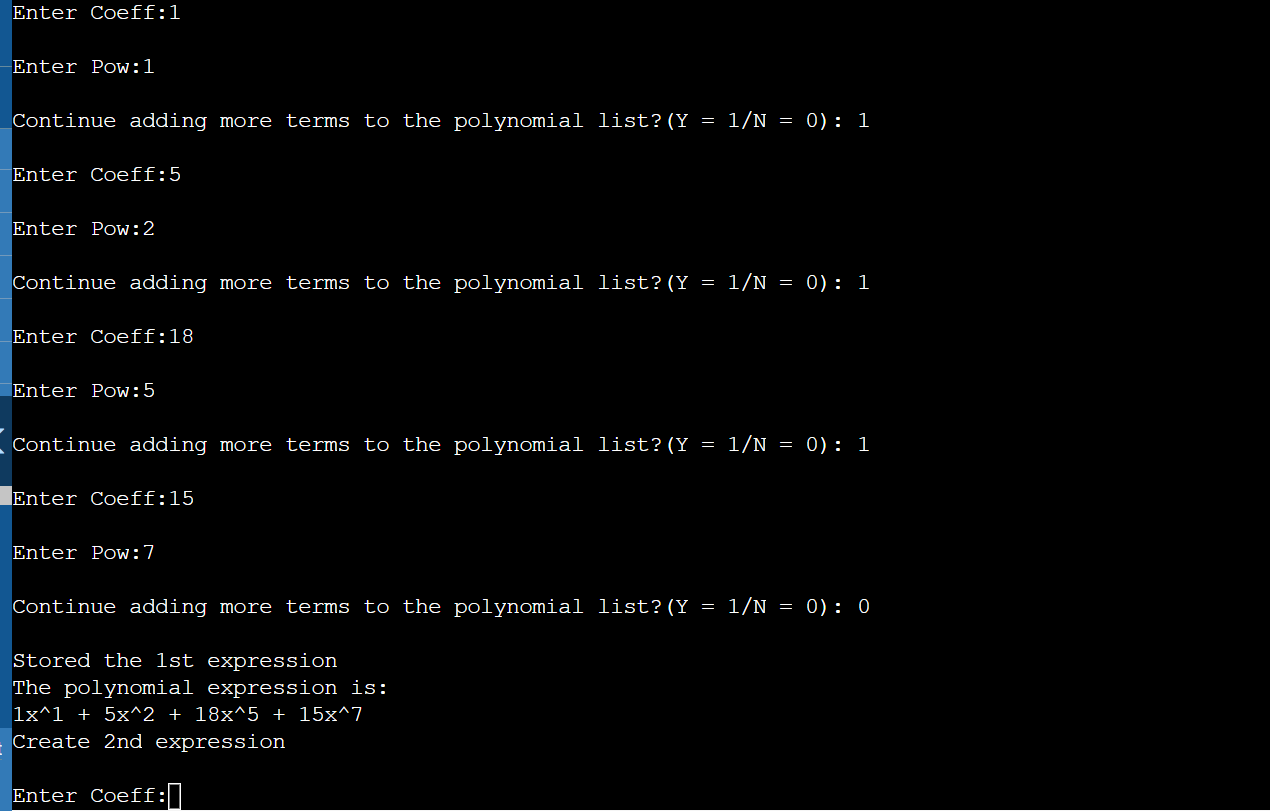
}

}

printf("\nAddition Complete");

}

**SAMPLE OUTPUT:**

****

**PROBLEM 3.2P:** CREATE A SORTED LINKED LIST. IMPLEMENT INSERTION AND SEARCHING OPERATION ON THE LIST. ALSO WRITE A FUNCTION DISPLAY THAT PRINTS THE CURRENT STATE OF THE LINKED LIST.

**SOLUTION APPROACH:**

1) If Linked list is empty then make the node as

head and return it.

2) If the value of the node to be inserted is smaller

than the value of the head node, then insert the node

at the start and make it head.

3) In a loop, find the appropriate node after

which the input node (let 9) is to be inserted.

To find the appropriate node start from the head,

keep moving until you reach a node GN (10 in

the below diagram) who's value is greater than

the input node. The node just before GN is the

appropriate node (7).

4) Insert the node (9) after the appropriate node

(7) found in step 3.

**SOURCE CODE:**

#include <stdio.h>

#include <stdlib.h>

/\* Link list node \*/

struct Node {

int data;

struct Node\* next;

};

/\* function to insert a new\_node

in a list. Note that this

function expects a pointer

to head\_ref as this can modify the

head of the input linked

list (similar to push())\*/

void sortedInsert(struct Node\*\* head\_ref,

struct Node\* new\_node)

{

struct Node\* current;

/\* Special case for the head end \*/

if (\*head\_ref == NULL

|| (\*head\_ref)->data

>= new\_node->data) {

new\_node->next = \*head\_ref;

\*head\_ref = new\_node;

}

else {

/\* Locate the node before

the point of insertion \*/

current = \*head\_ref;

while (current->next != NULL

&& current->next->data < new\_node->data) {

current = current->next;

}

new\_node->next = current->next;

current->next = new\_node;

}

}

/\* BELOW FUNCTIONS ARE JUST UTILITY TO TEST sortedInsert \*/

/\* A utility function to create a new node \*/

struct Node\* newNode(int new\_data)

{

/\* allocate node \*/

struct Node\* new\_node

= (struct Node\*)malloc(

sizeof(struct Node));

/\* put in the data \*/

new\_node->data = new\_data;

new\_node->next = NULL;

return new\_node;

}

/\* Function to print linked list \*/

void printList(struct Node\* head)

{

struct Node\* temp = head;

while (temp != NULL) {

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

temp = temp->next;

}

}

/\* Driver program to test count function\*/

int main()

{

/\* Start with the empty list \*/

struct Node\* head = NULL;

struct Node\* new\_node = newNode(5);

sortedInsert(&head, new\_node);

new\_node = newNode(10);

sortedInsert(&head, new\_node);

new\_node = newNode(7);

sortedInsert(&head, new\_node);

new\_node = newNode(3);

sortedInsert(&head, new\_node);

new\_node = newNode(1);

sortedInsert(&head, new\_node);

new\_node = newNode(9);

sortedInsert(&head, new\_node);

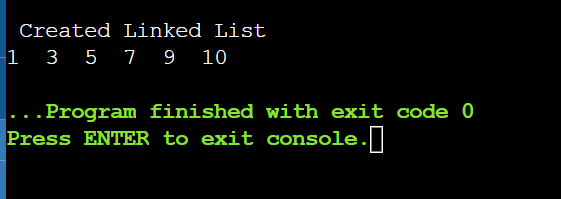
printf("\n Created Linked List\n");

printList(head);

return 0;

}

**SAMPLE OUTPUT:**

****

**PROBLEM 3.3P:** IMPLEMENT CIRCULAR LINKED LIST AND ITS OPERATIONS.

**SOLUTION APPROACH:** MAKE DIFFERENT FUNCTIONS FOR EACH OPERATIONS AND GIVE THEIR DEFINITION AND IMPLEMENT THE PROGRAM USING MENU DRIVEN CONCEPT AND BY MAKING USE OF SWITCH CASES.

**SOURCE CODE:**

#include<stdio.h>

#include<stdlib.h>

struct node

{

int data;

struct node \*next;

};

struct node \*head;

void beginsert ();

void lastinsert ();

void randominsert();

void begin\_delete();

void last\_delete();

void random\_delete();

void display();

void search();

void main ()

{

int choice =0;

while(choice != 7)

{

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

printf("\nChoose one option from the following list ...\n");

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

printf("\n1.Insert in begining\n2.Insert at last\n3.Delete from Beginning\n4.Delete from last\n5.Search for an element\n6.Show\n7.Exit\n");

printf("\nEnter your choice?\n");

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

switch(choice)

{

case 1:

beginsert();

break;

case 2:

lastinsert();

break;

case 3:

begin\_delete();

break;

case 4:

last\_delete();

break;

case 5:

search();

break;

case 6:

display();

break;

case 7:

exit(0);

break;

default:

printf("Please enter valid choice..");

}

}

}

void beginsert()

{

struct node \*ptr,\*temp;

int item;

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

if(ptr == NULL)

{

printf("\nOVERFLOW");

}

else

{

printf("\nEnter the node data?");

scanf("%d",&item);

ptr -> data = item;

if(head == NULL)

{

head = ptr;

ptr -> next = head;

}

else

{

temp = head;

while(temp->next != head)

temp = temp->next;

ptr->next = head;

temp -> next = ptr;

head = ptr;

}

printf("\nnode inserted\n");

}

}

void lastinsert()

{

struct node \*ptr,\*temp;

int item;

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

if(ptr == NULL)

{

printf("\nOVERFLOW\n");

}

else

{

printf("\nEnter Data?");

scanf("%d",&item);

ptr->data = item;

if(head == NULL)

{

head = ptr;

ptr -> next = head;

}

else

{

temp = head;

while(temp -> next != head)

{

temp = temp -> next;

}

temp -> next = ptr;

ptr -> next = head;

}

printf("\nnode inserted\n");

}

}

void begin\_delete()

{

struct node \*ptr;

if(head == NULL)

{

printf("\nUNDERFLOW");

}

else if(head->next == head)

{

head = NULL;

free(head);

printf("\nnode deleted\n");

}

else

{ ptr = head;

while(ptr -> next != head)

ptr = ptr -> next;

ptr->next = head->next;

free(head);

head = ptr->next;

printf("\nnode deleted\n");

}

}

void last\_delete()

{

struct node \*ptr, \*preptr;

if(head==NULL)

{

printf("\nUNDERFLOW");

}

else if (head ->next == head)

{

head = NULL;

free(head);

printf("\nnode deleted\n");

}

else

{

ptr = head;

while(ptr ->next != head)

{

preptr=ptr;

ptr = ptr->next;

}

preptr->next = ptr -> next;

free(ptr);

printf("\nnode deleted\n");

}

}

void search()

{

struct node \*ptr;

int item,i=0,flag=1;

ptr = head;

if(ptr == NULL)

{

printf("\nEmpty List\n");

}

else

{

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

scanf("%d",&item);

if(head ->data == item)

{

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

flag=0;

}

else

{

while (ptr->next != head)

{

if(ptr->data == item)

{

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

flag=0;

break;

}

else

{

flag=1;

}

i++;

ptr = ptr -> next;

}

}

if(flag != 0)

{

printf("Item not found\n");

}

}

}

void display()

{

struct node \*ptr;

ptr=head;

if(head == NULL)

{

printf("\nnothing to print");

}

else

{

printf("\n printing values ... \n");

while(ptr -> next != head)

{

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

ptr = ptr -> next;

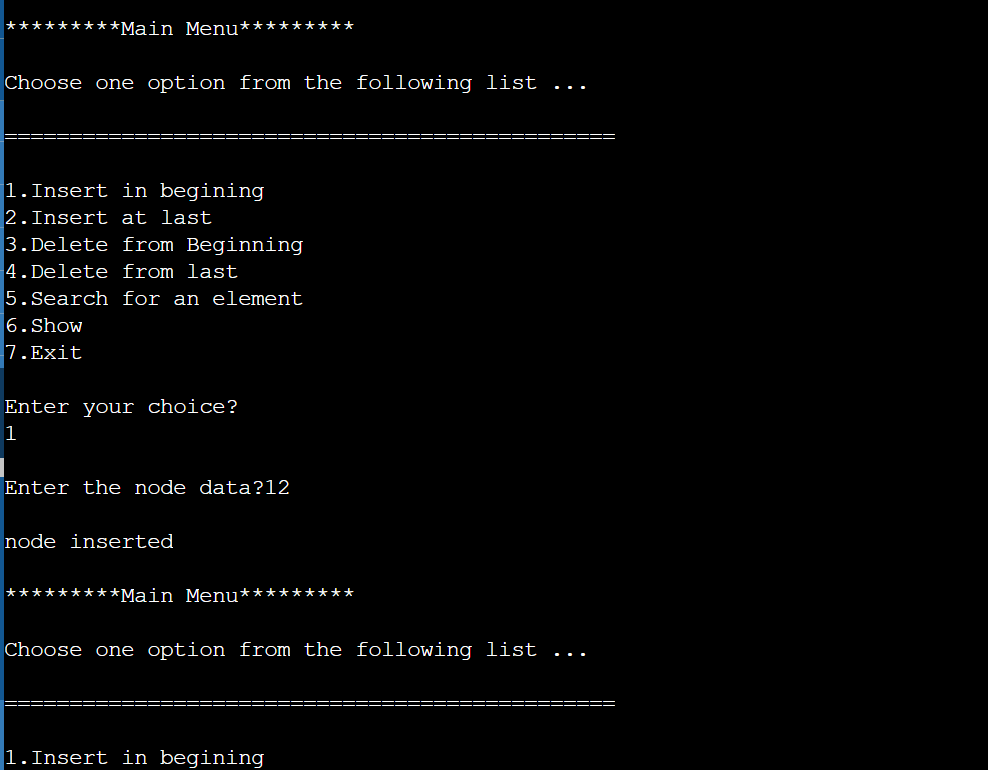
}

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

}

}

**SAMPLE OUTPUT:**

****

**PROBLEM 3.4P:** IMPLEMENT DOUBLY LINKED LIST AND ITS OPERATIONS

**SOLUTION APPROACH**: MAKE DIFFERENT FUNCTIONS FOR EACH OPERATIONS AND GIVE THEIR DEFINITION AND IMPLEMENT THE PROGRAM USING MENU DRIVEN CONCEPT AND BY MAKING USE OF SWITCH CASES.

**SOURCE CODE:**

#include<stdio.h>

#include<stdlib.h>

struct node

{

struct node \*prev;

struct node \*next;

int data;

};

struct node \*head;

void insertion\_beginning();

void insertion\_last();

void insertion\_specified();

void deletion\_beginning();

void deletion\_last();

void deletion\_specified();

void display();

void search();

void main ()

{

int choice =0;

while(choice != 9)

{

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

printf("\nChoose one option from the following list ...\n");

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

printf("\n1.Insert in begining\n2.Insert at last\n3.Insert at any random location\n4.Delete from Beginning\n 5.Delete from last\n6.Delete the node after the given data\n7.Search\n8.Show\n9.Exit\n");

printf("\nEnter your choice?\n");

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

switch(choice)

{

case 1:

insertion\_beginning();

break;

case 2:

insertion\_last();

break;

case 3:

insertion\_specified();

break;

case 4:

deletion\_beginning();

break;

case 5:

deletion\_last();

break;

case 6:

deletion\_specified();

break;

case 7:

search();

break;

case 8:

display();

break;

case 9:

exit(0);

break;

default:

printf("Please enter valid choice..");

}

}

}

void insertion\_beginning()

{

struct node \*ptr;

int item;

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

if(ptr == NULL)

{

printf("\nOVERFLOW");

}

else

{

printf("\nEnter Item value");

scanf("%d",&item);

if(head==NULL)

{

ptr->next = NULL;

ptr->prev=NULL;

ptr->data=item;

head=ptr;

}

else

{

ptr->data=item;

ptr->prev=NULL;

ptr->next = head;

head->prev=ptr;

head=ptr;

}

printf("\nNode inserted\n");

}

}

void insertion\_last()

{

struct node \*ptr,\*temp;

int item;

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

if(ptr == NULL)

{

printf("\nOVERFLOW");

}

else

{

printf("\nEnter value");

scanf("%d",&item);

ptr->data=item;

if(head == NULL)

{

ptr->next = NULL;

ptr->prev = NULL;

head = ptr;

}

else

{

temp = head;

while(temp->next!=NULL)

{

temp = temp->next;

}

temp->next = ptr;

ptr ->prev=temp;

ptr->next = NULL;

}

}

printf("\nnode inserted\n");

}

void insertion\_specified()

{

struct node \*ptr,\*temp;

int item,loc,i;

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

if(ptr == NULL)

{

printf("\n OVERFLOW");

}

else

{

temp=head;

printf("Enter the location");

scanf("%d",&loc);

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

{

temp = temp->next;

if(temp == NULL)

{

printf("\n There are less than %d elements", loc);

return;

}

}

printf("Enter value");

scanf("%d",&item);

ptr->data = item;

ptr->next = temp->next;

ptr -> prev = temp;

temp->next = ptr;

temp->next->prev=ptr;

printf("\nnode inserted\n");

}

}

void deletion\_beginning()

{

struct node \*ptr;

if(head == NULL)

{

printf("\n UNDERFLOW");

}

else if(head->next == NULL)

{

head = NULL;

free(head);

printf("\nnode deleted\n");

}

else

{

ptr = head;

head = head -> next;

head -> prev = NULL;

free(ptr);

printf("\nnode deleted\n");

}

}

void deletion\_last()

{

struct node \*ptr;

if(head == NULL)

{

printf("\n UNDERFLOW");

}

else if(head->next == NULL)

{

head = NULL;

free(head);

printf("\nnode deleted\n");

}

else

{

ptr = head;

if(ptr->next != NULL)

{

ptr = ptr -> next;

}

ptr -> prev -> next = NULL;

free(ptr);

printf("\nnode deleted\n");

}

}

void deletion\_specified()

{

struct node \*ptr, \*temp;

int val;

printf("\n Enter the data after which the node is to be deleted : ");

scanf("%d", &val);

ptr = head;

while(ptr -> data != val)

ptr = ptr -> next;

if(ptr -> next == NULL)

{

printf("\nCan't delete\n");

}

else if(ptr -> next -> next == NULL)

{

ptr ->next = NULL;

}

else

{

temp = ptr -> next;

ptr -> next = temp -> next;

temp -> next -> prev = ptr;

free(temp);

printf("\nnode deleted\n");

}

}

void display()

{

struct node \*ptr;

printf("\n printing values...\n");

ptr = head;

while(ptr != NULL)

{

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

ptr=ptr->next;

}

}

void search()

{

struct node \*ptr;

int item,i=0,flag;

ptr = head;

if(ptr == NULL)

{

printf("\nEmpty List\n");

}

else

{

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

scanf("%d",&item);

while (ptr!=NULL)

{

if(ptr->data == item)

{

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

flag=0;

break;

}

else

{

flag=1;

}

i++;

ptr = ptr -> next;

}

if(flag==1)

{

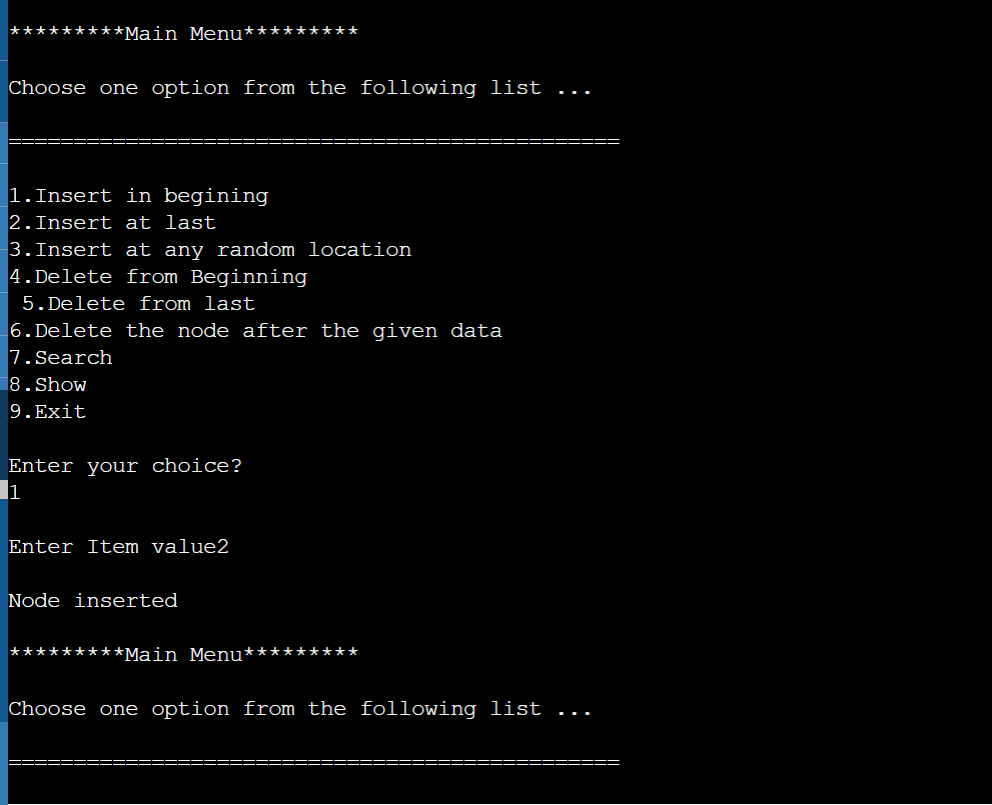
printf("\nItem not found\n");

}

}

}

**SAMPLE OUTPUT:**

****

**EXPERIMENT 4**

**TITLE: STACK DATA STRUCTURE**

**OBJECTIVE:** TO DEMONSTRATE USE OF ARRAYS AND LINKED LIST TO IMPLEMENT STACK OPERATIONS AND APPLICATIONS OF STACK.

**PROBLEM 4.1L:** USING ARRAY AND FUNCTIONS IMPLEMENT STACK AND ITS OPERATIONS LIKE PUSH,POP,PEEK.

**SOLUTION APPROACH:**

**SOURCE CODE:**

#include<stdio.h>

int stack[100],choice,n,top,x,i;

void push(void);

void pop(void);

void display(void);

int main()

{

//clrscr();

top=-1;

printf("\n Enter the size of STACK[MAX=100]:");

scanf("%d",&n);

printf("\n\t STACK OPERATIONS USING ARRAY");

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

printf("\n\t 1.PUSH\n\t 2.POP\n\t 3.DISPLAY\n\t 4.EXIT");

do

{

printf("\n Enter the Choice:");

scanf("%d",&choice);

switch(choice)

{

case 1:

{

push();

break;

}

case 2:

{

pop();

break;

}

case 3:

{

display();

break;

}

case 4:

{

printf("\n\t EXIT POINT ");

break;

}

default:

{

printf ("\n\t Please Enter a Valid Choice(1/2/3/4)");

}

}

}

while(choice!=4);

return 0;

}

void push()

{

if(top>=n-1)

{

printf("\n\tSTACK is over flow");

}

else

{

printf(" Enter a value to be pushed:");

scanf("%d",&x);

top++;

stack[top]=x;

}

}

void pop()

{

if(top<=-1)

{

printf("\n\t Stack is under flow");

}

else

{

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

top--;

}

}

void display()

{

if(top>=0)

{

printf("\n The elements in STACK \n");

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

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

printf("\n Press Next Choice");

}

else

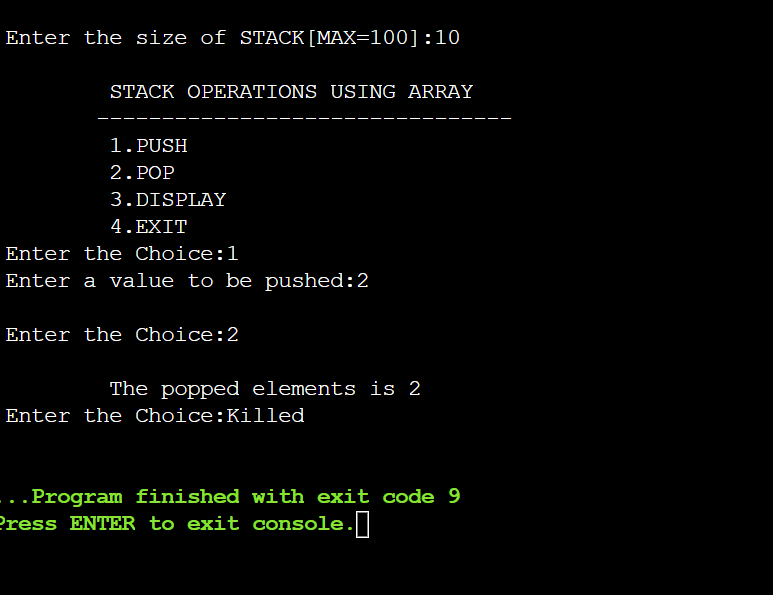
{

printf("\n The STACK is empty");

}

}

**SAMPLE OUTPUT:**

****

**PROBLEM 4.2L**: USE THE STACK OPERATIONS DEVELOPED IN PROBLEM 1.4L AND REVERSE A STRING USING STACK.

**SOLUTION APPROACH:**

1) Create an empty stack.

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

3) One by one pop all characters from stack and put

them back to string.

**SOURCE CODE:**

// C program to reverse a string using stack

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include <limits.h>

// A structure to represent a stack

struct Stack

{

int top;

unsigned capacity;

char\* array;

};

// function to create a stack of given

// capacity. It initializes size of stack as 0

struct Stack\* createStack(unsigned capacity)

{

struct Stack\* stack = (struct Stack\*) malloc(sizeof(struct Stack));

stack->capacity = capacity;

stack->top = -1;

stack->array = (char\*) malloc(stack->capacity \* sizeof(char));

return stack;

}

// Stack is full when top is equal to the last index

int isFull(struct Stack\* stack)

{ return stack->top == stack->capacity - 1; }

// Stack is empty when top is equal to -1

int isEmpty(struct Stack\* stack)

{ return stack->top == -1; }

// Function to add an item to stack.

// It increases top by 1

void push(struct Stack\* stack, char item)

{

if (isFull(stack))

return;

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

}

// Function to remove an item from stack.

// It decreases top by 1

char pop(struct Stack\* stack)

{

if (isEmpty(stack))

return INT\_MIN;

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

}

// A stack based function to reverse a string

void reverse(char str[])

{

// Create a stack of capacity

//equal to length of string

int n = strlen(str);

struct Stack\* stack = createStack(n);

// Push all characters of string to stack

int i;

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

push(stack, str[i]);

// Pop all characters of string and

// put them back to str

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

str[i] = pop(stack);

}

// Driver program to test above functions

int main()

{

char str[] = "MuskanSingh";

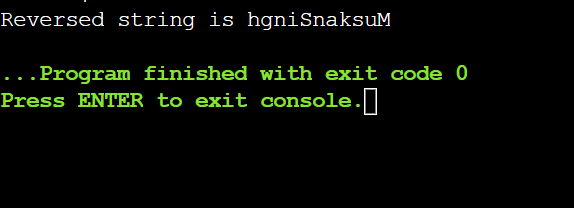
reverse(str);

printf("Reversed string is %s", str);

return 0;

}

**SAMPLE OUTPUT:**

****

**4.1P:** USING ARRAY AND FUNCTIONS IMPLEMENT TWO STACKS AND ITS OPERATIONS.

**SOLUTION APPROACH**:

**SOURCE CODE:**

#include<stdio.h>

void PUSH(int \*stack, int \*TOP, int ins, int max){

if (\*TOP==max)

printf("Overflow.");

else{

\*TOP=\*TOP+1;

stack[\*TOP]=ins;

}

}

int POP(int \*stack, int \*TOP){

if (\*TOP==-1)

printf("Underflow.");

else{

int item=stack[\*TOP];

\*TOP=\*TOP-1;

return item;

}

}

int PEEK(int \*stack, int \*TOP){

if (\*TOP==-1)

printf("Empty stack.");

else

return stack[\*TOP];

}

void display(int \*stack, int \*TOP){

if (\*TOP==-1){

printf("Empty stack.");

return;

}

for (int i=0; i<\*TOP+1; i++){

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

}

}

void main(){

int n1,n2,top1=-1,top2=-1;

printf("Enter max size of stack1: ");

scanf("%d",&n1);

printf("Enter max size of stack2: ");

scanf("%d",&n2);

int stack1[n1],stack2[n2];

////////////

PUSH(stack1,&top1,1,n1);

PUSH(stack1,&top1,2,n1);

PUSH(stack1,&top1,3,n1);

PUSH(stack1,&top1,4,n1);

PUSH(stack1,&top1,5,n1);

display(stack1,&top1);

printf("\n%d\n",POP(stack1,&top1));

printf("%d\n",POP(stack1,&top1));

printf("%d\n",POP(stack1,&top1));

display(stack1,&top1);

printf("\n%d\n",PEEK(stack1,&top1));

////////////

PUSH(stack2,&top2,1,n2);

PUSH(stack2,&top2,2,n2);

PUSH(stack2,&top2,3,n2);

PUSH(stack2,&top2,4,n2);

PUSH(stack2,&top2,5,n2);

display(stack2,&top2);

printf("\n%d\n",POP(stack2,&top2));

printf("%d\n",POP(stack2,&top2));

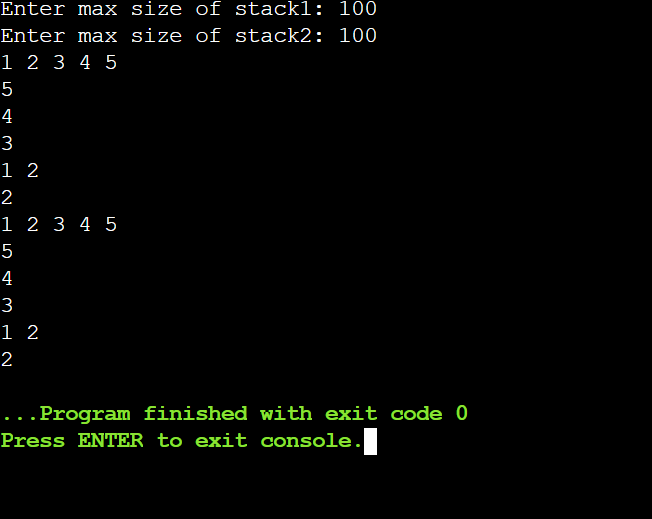
printf("%d\n",POP(stack2,&top2));

display(stack2,&top2);

printf("\n%d",PEEK(stack2,&top2));

}

**SAMPLE OUTPUT:**

****

**PROBLEM 4.2P:** CONVERT INFIX TO POSTFIX EXPRESSION USING STACK AND ARRAY.

**SOLUTION APPROACH:**

**SOURCE CODE:**

#include<stdio.h>

#include<string.h>

int TOP=-1;

void PUSH(char \*stack, char ins, int max){

if (TOP==max)

printf("Overflow.");

else{

TOP=TOP+1;

stack[TOP]=ins;

}

}

char POP(char \*stack){

if (TOP==-1)

printf("Underflow.");

else{

char item=stack[TOP];

TOP=TOP-1;

return item;

}

}

char PEEK(char \*stack){

if (TOP==-1)

printf("Empty stack.");

else

return stack[TOP];

}

void POST(char \*infix, char \*post){

int count=0;

int max=strlen(infix);

char stack[max],temp;

PUSH(stack,'(',max);

for (int i=0; TOP!=-1; i++){

if (infix[i]>64 && infix[i]<91){

post[count]=infix[i];

count=count+1;

}else if (infix[i]=='(')

PUSH(stack, infix[i], max);

else if (infix[i]=='+' || infix[i]=='-' || infix[i]=='\*' || infix[i]=='/' || infix[i]=='^'){

if (PEEK(stack)=='(')

PUSH(stack, infix[i], max);

else{

if (infix[i]=='+' || infix[i]=='-'){

while (PEEK(stack)=='^' || PEEK(stack)=='\*' || PEEK(stack)=='/' || PEEK(stack)=='+' || PEEK(stack)=='-'){

post[count]=POP(stack);

count++;

}

}else if (infix[i]=='\*' || infix[i]=='/'){

while (PEEK(stack)=='^' || PEEK(stack)=='\*' || PEEK(stack)=='/'){

post[count]=POP(stack);

count++;

}

}else{

while (PEEK(stack)=='^'){

post[count]=POP(stack);

count++;

}

}

PUSH(stack,infix[i],max);

}

}else if (infix[i]==')'){

while (PEEK(stack)!='('){

post[count]=POP(stack);

count++;

}

temp=POP(stack);

}

}

}

void main(){

char infix[]="A+(B\*I-(D/E^F)\*G)\*H)";

int n=strlen(infix);

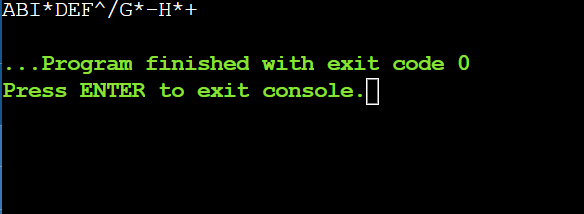
char post[n];

POST(infix,post);

printf("%s",post);

}

**SAMPLE OUTPUT:**

****

**PROBLEM 4.3P:** EVALUATE POSTFIX EXPRESSION USING STACK AND ARRAY.

**SOLUTION APPROACH:**

**SOURCE CODE:**

#include<stdio.h>

#include<string.h>

int TOP=-1;

void PUSH(char \*stack, char ins, int max){

if (TOP==max)

printf("Overflow.");

else{

TOP=TOP+1;

stack[TOP]=ins;

}

}

char POP(char \*stack){

if (TOP==-1)

printf("Underflow.");

else{

char item=stack[TOP];

TOP=TOP-1;

return item;

}

}

int eval(char \*post){

int max=strlen(post);

char stack[max],ans,A,B;

int res;

for (int i=0; post[i]!=')'; i++){

if (post[i]>47 && post[i]<58)

PUSH(stack,post[i],max);

else{

if (post[i]=='+'){

A=POP(stack);

B=POP(stack);

ans=(int)B+(int)A;

PUSH(stack,ans,max);

}else if (post[i]=='-'){

A=POP(stack);

B=POP(stack);

ans=(int)B-(int)A;

PUSH(stack,ans,max);

}else if (post[i]=='\*'){

A=POP(stack);

B=POP(stack);

ans=(int)B\*(int)A;

PUSH(stack,ans,max);

}else if (post[i]=='/'){

A=POP(stack);

B=POP(stack);

ans=(int)B/(int)A;

PUSH(stack,ans,max);

}

}

}

res=(int)POP(stack);

return res;

}

void main(){

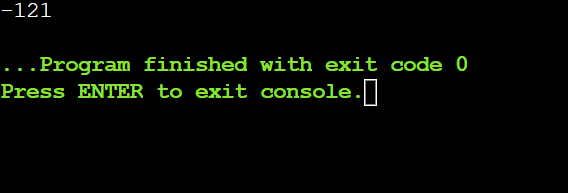
char post[]={'5','6','2','+','\*','8','4','/','-',')'};

int ans=eval(post);

printf("%d",ans);

}

**SAMPLE OUTPUT:**

****

**PROBLEM 4.4P:** IMPLEMENT QUEUE OPERATIONS USING STACK.

**SOLUTION APPROACH:**

**SOURCE CODE:**

#include<stdio.h>

#define N 5

int stack1[5], stack2[5]; // declaration of two stacks

// declaration of top variables.

int top1=-1, top2=-1;

int count=0;

// inserting the elements in stack1.

void push1(int data)

{

// Condition to check whether the stack1 is full or not.

if(top1==N-1)

{

printf("\n Stack is overflow...");

}

else

{

top1++; // Incrementing the value of top1

stack1[top1]=data; // pushing the data into stack1

}

}

// Removing the elements from the stack1.

int pop1()

{

// Condition to check whether the stack1 is empty or not.

if(top1==-1)

{

printf("\nStack is empty..");

}

else

{

int a=stack1[top1]; // Assigning the topmost value of stack1 to 'a' variable.

top1--; // decrementing the value of top1.

return a;

}

}

// pushing the data into the stack2.

void push2(int x)

{

// Condition to check whether the stack2 is full or not

if(top2==N-1)

{

printf("\nStack is full..");

}

else

{

top2++; // incrementing the value of top2.

stack2[top2]=x; // assigning the 'x' value to the Stack2

}

}

// Removing the elements from the Stack2

int pop2()

{

int element = stack2[top2]; // assigning the topmost value to element

top2--; // decrement the value of top2

return element;

}

void enqueue(int x)

{

push1(x);

count++;

}

void dequeue()

{

if((top1==-1) && (top2==-1))

{

printf("\nQueue is empty");

}

else

{

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

{

int element = pop1();

push2(element);

}

int b= pop2();

printf("\nThe dequeued element is %d", b);

printf("\n");

count--;

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

{

int a = pop2();

push1(a);

}

}}

void display()

{

for(int i=0;i<=top1;i++)

{

printf("%d , ", stack1[i]);

}

}

void main()

{

enqueue(10);

enqueue(20);

enqueue(30);

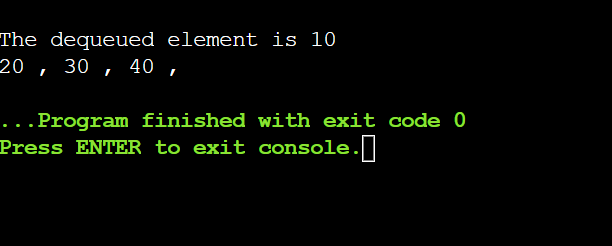
dequeue();

enqueue(40);

display();

}

**SAMPLE OUTPUT:**

****

**EXPERIMENT 5**

**TITLE:** QUEUE DATA STRUCTURE

**OBJECTIVE: TO DEMONSTRATE USE OF ARRAYS AND LINKED LIST TO IMPLEMENT QUEUE OPERATIONS.**

**PROBLEM 5.1L: USING CIRCULAR ARRAY AND FUNCTIONS IMPLEMENT QUEUE AND ITS OPERATIONS.**

**SOURCRE CODE:**

#include <stdio.h>

# define max 6

int queue[max]; // array declaration

int front=-1;

int rear=-1;

// function to insert an element in a circular queue

void enqueue(int element)

{

if(front==-1 && rear==-1) // condition to check queue is empty

{

front=0;

rear=0;

queue[rear]=element;

}

else if((rear+1)%max==front) // condition to check queue is full

{

printf("Queue is overflow..");

}

else

{

rear=(rear+1)%max; // rear is incremented

queue[rear]=element; // assigning a value to the queue at the rear position.

}

}

// function to delete the element from the queue

int dequeue()

{

if((front==-1) && (rear==-1)) // condition to check queue is empty

{

printf("\nQueue is underflow..");

}

else if(front==rear)

{

printf("\nThe dequeued element is %d", queue[front]);

front=-1;

rear=-1;

}

else

{

printf("\nThe dequeued element is %d", queue[front]);

front=(front+1)%max;

}

}

// function to display the elements of a queue

void display()

{

int i=front;

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

{

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

}

else

{

printf("\nElements in a Queue are :");

while(i<=rear)

{

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

i=(i+1)%max;

}

}

}

int main()

{

int choice=1,x; // variables declaration

while(choice<4 && choice!=0) // while loop

{

printf("\n Press 1: Insert an element");

printf("\nPress 2: Delete an element");

printf("\nPress 3: Display the element");

printf("\nEnter your choice");

scanf("%d", &choice);

switch(choice)

{

case 1:

printf("Enter the element which is to be inserted");

scanf("%d", &x);

enqueue(x);

break;

case 2:

dequeue();

break;

case 3:

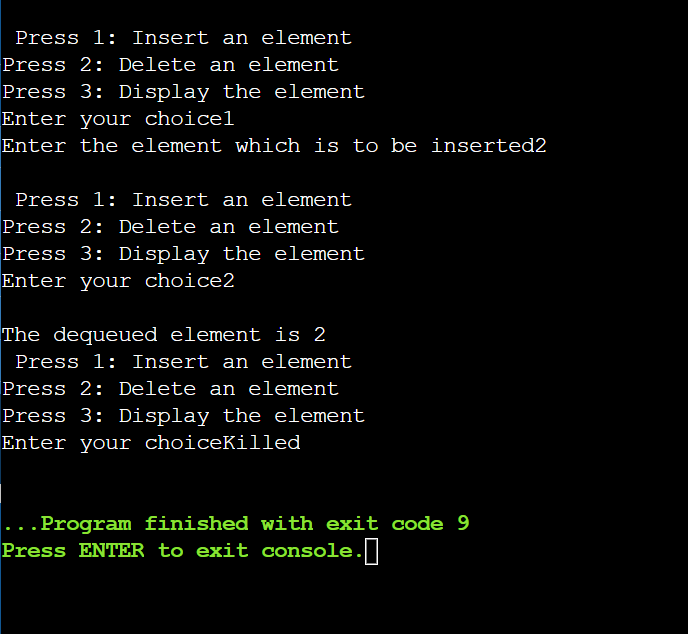
display();

}}

return 0;

}

**SAMPLE OUTPUT:**

****

**PROBLEM 5.2L: CHECK WHETHER THE STRING IS PALINDROME OR NOT USING QUEUE AND STACK.**

**SOURCE CODE:**

#include <stdio.h>

#include <string.h>

void push(char);

char pop();

char stack[100];

int top = -1;

void main()

{

char str[100];

int i, count = 0, len;

printf("Enter string to check it is palindrome or not : ");

scanf("%s", str);

len = strlen(str);

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

{

push(str[i]);

}

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

{

if (str[i] == pop())

count++;

}

if (count == len)

printf("%s is a Palindrome string\n", str);

else

printf("%s is not a palindrome string\n", str);

}

/\* Function to push character into stack \*/

void push(char c)

{

stack[++top] = c;

}

/\* Function to pop the top character from stack \*/

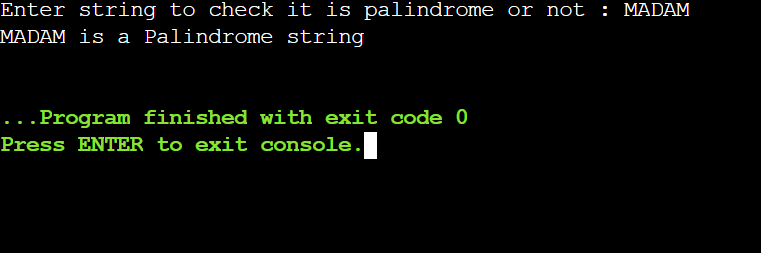
char pop()

{

return(stack[top--]);

}

**SAMPLE OUTPUT:**

****

**PROBLEM 5.1P: IMPLEMENT QUEUE DATA STRUCTURE USING LINKED LIST AND ITS OPERATIONS (ENQUEUE, DEQUEUE, DISPLAY).**

**SOLUTION APPROACH:**

**\*ENQUEUE:** *\*struct node*ptrptr->data = valptr->next = NULLif((front == NULL)&&(rear == NULL))front = rear = ptrelserear->next = ptrrear = ptr

**\*DEQUEUE:** *\*struct node*temptemp = frontif((front ==NULL)&&(rear == NULL))returnelsefront = front->nextfree(temp)

**\*DISPLAY:** *\*struct node*tempif((front ==NULL)&&(rear == NULL))returnelsetemp = frontwhile(temp)return temp->datatemp = temp->next

**SOURCE CODE:**

#include <stdio.h>

#include <stdlib.h>

// Structure to create a node with data and next pointer

struct node {

int data;

struct node \*next;

};

struct node \*front = NULL;

struct node \*rear = NULL;

// Enqueue() operation on a queue

void enqueue(int value) {

struct node \*ptr;

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

ptr->data = value;

ptr->next = NULL;

if ((front == NULL) && (rear == NULL)) {

front = rear = ptr;

}

else{

rear->next = ptr;

rear = ptr;

}

printf("Node is Inserted\n\n");

}

// Dequeue() operation on a queue

int dequeue() {

if (front == NULL) {

printf("\nUnderflow\n");

return -1;

}

else {

struct node \*temp = front;

int temp\_data = front->data;

front = front->next;

free(temp);

return temp\_data;

}

}

// Display all elements of queue

void display() {

struct node \*temp;

if ((front == NULL) && (rear == NULL)) {

printf("\nQueue is Empty\n");

}

else {

printf("The queue is \n");

temp = front;

while (temp) {

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

temp = temp->next;

}

printf("NULL\n\n");

}

}

void main() {

int choice, value;

printf("\nImplementation of Queue using Linked List\n");

while (choice != 4) {

printf("1.Enqueue\n2.Dequeue\n3.Display\n4.Exit\n");

printf("\nEnter your choice : ");

scanf("%d", &choice);

switch (choice) {

case 1:

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

scanf("%d", &value);

enqueue(value);

break;

case 2:

printf("Popped element is :%d\n", dequeue());

break;

case 3:

display();

break;

case 4:

exit(0);

break;

default:

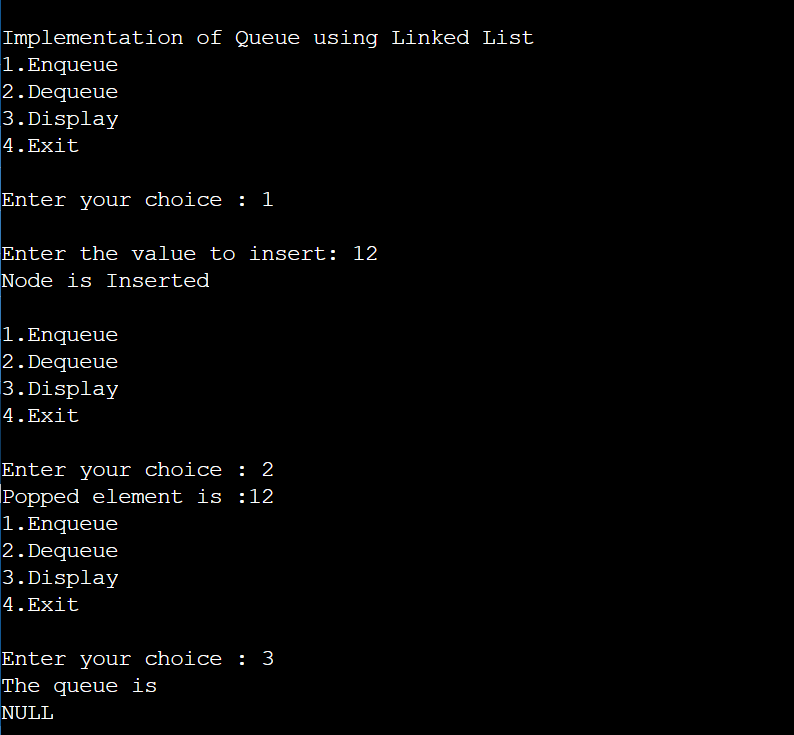
printf("\nWrong Choice\n");

}

}

}

**SAMPLE OUTPUT:**

****

**PROBLEM 5.2P: USING LINKED LIST AND FUNCTIONS IMPLEMENT PRIORITY QUEUE AND ITS OPERATIONS (INSERT, DELETE, DISPLAY).**

**SOLUTION APPROACH:**

**SOURCE CODE:**

# include<stdio.h>

# include<malloc.h>

typedef struct node

{

int priority;

int info;

struct node \*link;

}NODE;

NODE \*front = NULL;

// insert method

void insert(int data,int priority)

{

NODE \*temp,\*q;

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

temp->info = data;

temp->priority = priority;

// condition to check whether the first element is empty or the element to be inserted has more priority than the first element

if( front == NULL || priority < front->priority )

{

temp->link = front;

front = temp;

}

else

{

q = front;

while( q->link != NULL && q->link->priority <= priority )

q=q->link;

temp->link = q->link;

q->link = temp;

}

}

// delete method

void del()

{

NODE \*temp;

// condition to check whether the Queue is empty or not

if(front == NULL)

printf("Queue Underflow\n");

else

{

temp = front;

printf("Deleted item is %d\n", temp->info);

front = front->link;

free(temp);

}

}

// display method

void display()

{

NODE \*ptr;

ptr = front;

if(front == NULL)

printf("Queue is empty\n");

else

{

printf("Queue is :\n");

printf("Priority Item\n");

while(ptr != NULL)

{

printf("%5d %5d\n",ptr->priority,ptr->info);

ptr = ptr->link;

}

}

}

/\*End of display\*/

// main method

int main()

{

int choice, data, priority;

do

{

printf("1.Insert\n");

printf("2.Delete\n");

printf("3.Display\n");

printf("4.Quit\n");

printf("Enter your choice : ");

scanf("%d", &choice);

switch(choice)

{

case 1:

printf("Enter the data which is to be added in the queue : ");

scanf("%d",&data);

printf("Enter its priority : ");

scanf("%d",&priority);

insert(data,priority);

break;

case 2:

del();

break;

case 3:

display();

break;

case 4:

break;

default :

printf("Wrong choice\n");

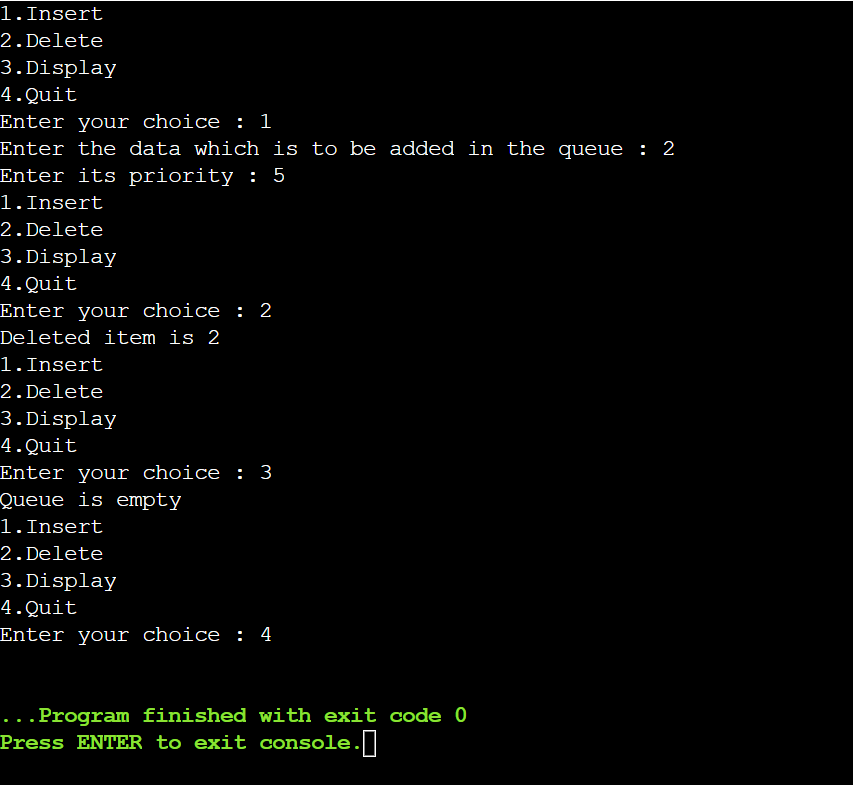
}

}while(choice!=4);

return 0;

}

**SAMPLE OUTPUT:**

****

**PROBLEM 5.3P: USING ARRAY AND FUNCTIONS IMPLEMENT DOUBLE ENDED QUEUE (INPUT RESTRICTED DEQUE AND OUTPUT RESTRICTED DEQUE) AND ITS OPERATIONS.**

**SOLUTION APPROACH:** fourfunctions insert\_left, insert\_right, delete\_left and delete\_right. As the naming specify these functions add or delete to the corresponding sides.

Then we got two display functions for both the different type types of a queue. The Size of array is 5 by default, to change, edit the second line of code.

**SOURCE CODE:**

# include<stdio.h>

# define MAX 5

int deque\_arr[MAX];

int left = -1;

int right = -1;

/\*Begin of insert\_right\*/

void insert\_right()

{

int added\_item;

if((left == 0 && right == MAX-1) || (left == right+1))

{ printf("Queue Overflow\n");

return;}

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

{ left = 0;

right = 0;}

else

if(right == MAX-1) /\*right is at last position of queue \*/

right = 0;

else

right = right+1;

printf("Input the element for adding in queue : ");

scanf("%d", &added\_item);

deque\_arr[right] = added\_item ;

}

/\*End of insert\_right\*/

/\*Begin of insert\_left\*/

void insert\_left()

{ int added\_item;

if((left == 0 && right == MAX-1) || (left == right+1))

{ printf("Queue Overflow \n");

return; }

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

{ left = 0;

right = 0; }

else

if(left== 0)

left=MAX-1;

else

left=left-1;

printf("Input the element for adding in queue : ");

scanf("%d", &added\_item);

deque\_arr[left] = added\_item ; }

/\*End of insert\_left\*/

/\*Begin of delete\_left\*/

void delete\_left()

{ if (left == -1)

{ printf("Queue Underflow\n");

return ; }

printf("Element deleted from queue is : %d\n",deque\_arr[left]);

if(left == right) /\*Queue has only one element \*/

{ left = -1;

right=-1; }

else

if(left == MAX-1)

left = 0;

else

left = left+1;

}

/\*End of delete\_left\*/

/\*Begin of delete\_right\*/

void delete\_right()

{if (left == -1)

{printf("Queue Underflow\n");

return ; }

printf("Element deleted from queue is : %d\n",deque\_arr[right]);

if(left == right) /\*queue has only one element\*/

{ left = -1;

right=-1; }

else

if(right == 0)

right=MAX-1;

else

right=right-1; }

/\*End of delete\_right\*/

/\*Begin of input\_que\*/

void display\_queue()

{ int front\_pos = left,rear\_pos = right;

if(left == -1)

{ printf("Queue is empty\n");

return; }

printf("Queue elements :\n");

if( front\_pos <= rear\_pos )

{ while(front\_pos <= rear\_pos)

{ printf("%d ",deque\_arr[front\_pos]);

front\_pos++; } }

else

{ while(front\_pos <= MAX-1)

{ printf("%d ",deque\_arr[front\_pos]);

front\_pos++; }

front\_pos = 0;

while(front\_pos <= rear\_pos)

{ printf("%d ",deque\_arr[front\_pos]);

front\_pos++;

}

}

printf("\n");

}

/\*End of display\_queue\*/

/\*Begin of input\_que\*/

void input\_que()

{ int choice;

do

{ printf("1.Insert at right\n");

printf("2.Delete from left\n");

printf("3.Delete from right\n");

printf("4.Display\n");

printf("5.Quit\n");

printf("Enter your choice : ");

scanf("%d",&choice);

switch(choice)

{ case 1:

insert\_right();

break;

case 2:

delete\_left();

break;

case 3:

delete\_right();

break;

case 4:

display\_queue();

break;

case 5:

break;

default:

printf("Wrong choice\n");

}

}while(choice!=5);

}

/\*End of input\_que\*/

/\*Begin of output\_que\*/

void output\_que()

{ int choice;

do

{ printf("1.Insert at right\n");

printf("2.Insert at left\n");

printf("3.Delete from left\n");

printf("4.Display\n");

printf("5.Quit\n");

printf("Enter your choice : ");

scanf("%d",&choice);

switch(choice)

{

case 1:

insert\_right();

break;

case 2:

insert\_left();

break;

case 3:

delete\_left();

break;

case 4:

display\_queue();

break;

case 5:

break;

default:

printf("Wrong choice\n");

}

}while(choice!=5);

}

/\*End of output\_que\*/

/\*Begin of main\*/

main()

{ int choice;

printf("1.Input restricted dequeue\n");

printf("2.Output restricted dequeue\n");

printf("Enter your choice : ");

scanf("%d",&choice);

switch(choice)

{

case 1 :

input\_que();

break;

case 2:

output\_que();

break;

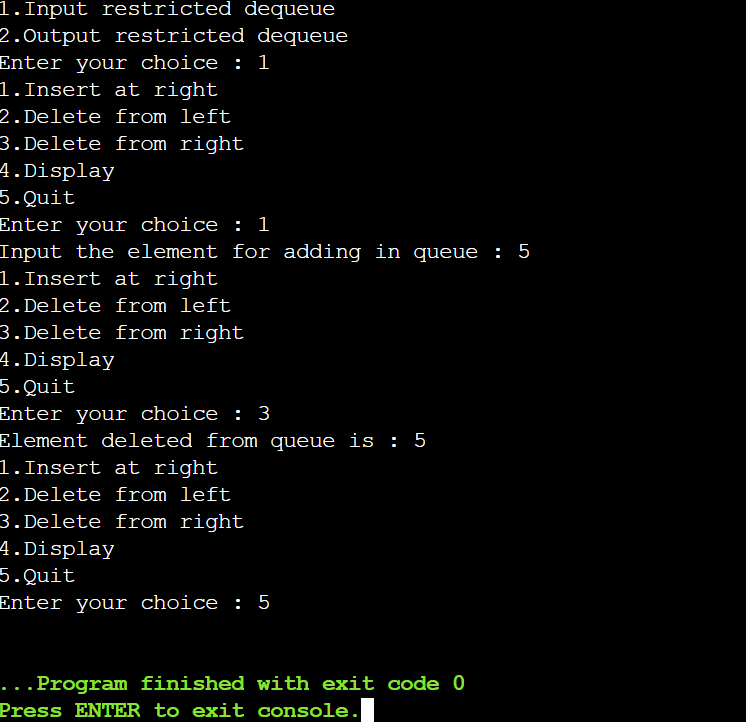
default:

printf("Wrong choice\n");

}

}

**SAMPLE OUTPUT:**

****

**EXPERIMENT 6**

**TITLE: SEARCHING & MERGING ALGORITHMS**

**OBJECTIVE: TO IMPLEMENT LINEAR SEARCH AND MERGING OPERATION.**

**PROBLEM 6.1L**: WAP TO IMPLEMENT LINEAR SEARCH ON A TWO WAY HEADER LINKED LIST**.**

**SOLUTION APPROACH:**

**SOURCE CODE:**

#include<stdio.h>

#include<stdlib.h>

void create(int);

void search();

struct node{

int data;

struct node \*prev;

struct node \*next;

};

struct node \*head;

void main()

{

int choice,item,pos;

do{

printf("\n1.create\n2.search\n3.exit\n4.enter your choice?");

scanf("%d",&choice);

switch(choice)

{

case 1:

printf("enter the item:");

scanf("%d",&item);

create(item);

break;

case 2:

search();

case 3:

exit(0);

break;

default:

printf("please enter a valid choice:\n");

}

}while(choice!=3);

}

void create(int item)

{

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

if(ptr==NULL)

{

printf("\noverflow");

}

else

{

if(head==NULL)

{

ptr->next=head;

ptr->prev=head;

ptr->data=item;

head=ptr;

}

else

{

ptr->data=item;

printf("press 0 to insert more?\n");

ptr->next=head;

ptr->prev=head;

ptr->data=item;

head=ptr;

}

printf("\nnode inserted");

}

}

void search()

{

struct node \*ptr;

int item,i=0,flag;

ptr=head;

if(ptr==NULL)

{

printf("\nempty list");

}

else

{

printf("enter the otem you want to search?\n");

scanf("%d",&item);

while(ptr!=NULL)

{

if(ptr->data==item)

{

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

flag=0;

break;

}

else

{

flag=1;

}

i++;

ptr=ptr->next;

}

if(flag==1)

{

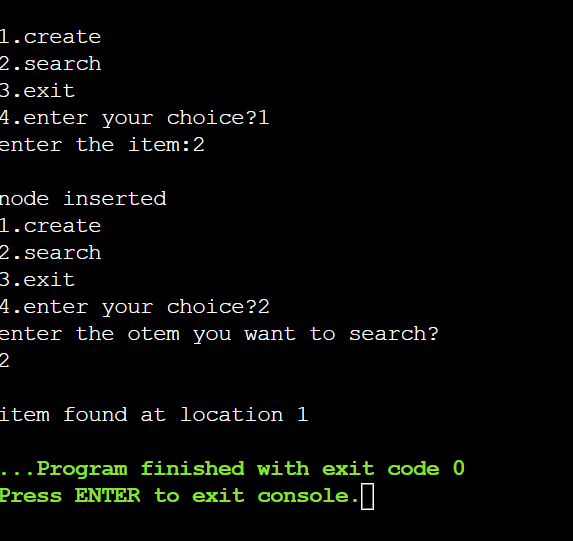
printf("item not found\n");

}

}

}

**SAMPLE OUTPUT:**

****

**PROBLEM 6.1P:** MERGE THE CONTENTS OF TWO SORTED LINEAR LISTS TO A SINGLE SORTED LINEAR LIST.

**SOLUTION APPROACH:**

The strategy here uses a temporary dummy node as the start of the result list. The pointer Tail always points to the last node in the result list, so appending new nodes is easy.

The dummy node gives the tail something to point to initially when the result list is empty. This dummy node is efficient, since it is only temporary, and it is allocated in the stack. The loop proceeds, removing one node from either ‘a’ or ‘b’, and adding it to the tail.

**SOURCE CODE:**

#include<stdio.h>

#include<stdlib.h>

#include<assert.h>

/\* Link list node \*/

struct Node

{

int data;

struct Node\* next;

};

/\* pull off the front node of the source and put it in dest \*/

void MoveNode(struct Node\*\* destRef, struct Node\*\* sourceRef);

/\* Takes two lists sorted in increasing order, and splices

their nodes together to make one big sorted list which

is returned. \*/

struct Node\* SortedMerge(struct Node\* a, struct Node\* b)

{

/\* a dummy first node to hang the result on \*/

struct Node dummy;

/\* tail points to the last result node \*/

struct Node\* tail = &dummy;

/\* so tail->next is the place to add new nodes

to the result. \*/

dummy.next = NULL;

while (1)

{

if (a == NULL)

{

/\* if either list runs out, use the

other list \*/

tail->next = b;

break;

}

else if (b == NULL)

{

tail->next = a;

break;

}

if (a->data <= b->data)

MoveNode(&(tail->next), &a);

else

MoveNode(&(tail->next), &b);

tail = tail->next;

}

return(dummy.next);

}

/\* UTILITY FUNCTIONS \*/

/\* MoveNode() function takes the node from the front of the

source, and move it to the front of the dest.

It is an error to call this with the source list empty.

Before calling MoveNode():

source == {1, 2, 3}

dest == {1, 2, 3}

After calling MoveNode():

source == {2, 3}

dest == {1, 1, 2, 3} \*/

void MoveNode(struct Node\*\* destRef, struct Node\*\* sourceRef)

{

/\* the front source node \*/

struct Node\* newNode = \*sourceRef;

assert(newNode != NULL);

/\* Advance the source pointer \*/

\*sourceRef = newNode->next;

/\* Link the old dest off the new node \*/

newNode->next = \*destRef;

/\* Move dest to point to the new node \*/

\*destRef = newNode;

}

/\* Function to insert a node at the beginning of the

linked list \*/

void push(struct Node\*\* head\_ref, int new\_data)

{

/\* allocate node \*/

struct Node\* new\_node =

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

/\* put in the data \*/

new\_node->data = new\_data;

/\* link the old list off the new node \*/

new\_node->next = (\*head\_ref);

/\* move the head to point to the new node \*/

(\*head\_ref) = new\_node;

}

/\* Function to print nodes in a given linked list \*/

void printList(struct Node \*node)

{

while (node!=NULL)

{

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

node = node->next;

}

}

/\* Driver program to test above functions\*/

int main()

{

/\* Start with the empty list \*/

struct Node\* res = NULL;

struct Node\* a = NULL;

struct Node\* b = NULL;

/\* Let us create two sorted linked lists to test

the functions

Created lists, a: 5->10->15, b: 2->3->20 \*/

push(&a, 15);

push(&a, 10);

push(&a, 5);

push(&b, 20);

push(&b, 3);

push(&b, 2);

/\* Remove duplicates from linked list \*/

res = SortedMerge(a, b);

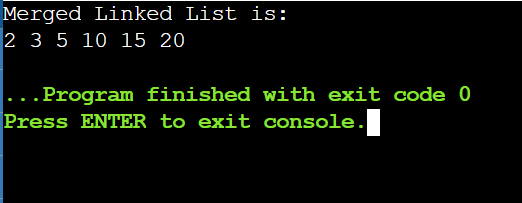
printf("Merged Linked List is: \n");

printList(res);

return 0;

}

**SAMPLE OUTPUT:**

****

Experiment-07

**Title: Implementing Searching algorithms**

**Objective:** To implement various searching algorithms.

**Problem 7.1:** Read the numbers from a text file sort them into an array using ‘Insertion Sort’ algorithm and write back in another text file.

**Source Code:**

#include <stdio.h>

#define Num 100

static int count = 0;

void ReadFile(int \*array) {

FILE \*fp = fopen("d1.txt","r");

while(1) {

if(feof(fp)) {

break;

}

fscanf(fp, "%d", &array[count]);

count++;

}

fclose(fp);

printf("\t\*\*FILE read Successfully\*\*\n");

}

void swap(int \*array, int i, int j) {

int temp = array[i];

array[i] = array[j];

array[j] = temp;

}

void InsertionSort(int \*array, int count) {

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

for (int j=i+1; j>0; j--) {

if(array[j] < array[j-1]) {

swap(array, j, j-1);

}

}

}

}

void DataDisplay(int \*array) {

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

printf("%d ",\*(array+i));

}

}

void WriteFile(int \*array) {

FILE \*fptr;

fptr = fopen("d2.txt","wb");

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

putw(\*(array+i), fptr);

}

fclose(fptr);

}

void main() {

int array[Num];

ReadFile(array);

printf("Original Data:\n");

DataDisplay(array);

InsertionSort(array, count);

printf("\n\nSorted Data:\n");

DataDisplay(array);

WriteFile(array);

}

**Problem 7.2:** Read the numbers from a text file sort them into an array using ‘Quick Sort’ algorithm and write back in another text file.

**Source Code:**

#include <stdio.h>

#define Num 100

static int count = 0;

void ReadFile(int \*array) {

FILE \*fp = fopen("d1.txt","r");

while(1) {

if(feof(fp)) {

break;

}

fscanf(fp, "%d", &array[count]);

count++;

}

fclose(fp);

printf("\t\*\*FILE read Successfully\*\*\n");

}

void swap(int \*array, int i, int j) {

int temp = array[i];

array[i] = array[j];

array[j] = temp;

}

**int** partition (**int** arr[], **int** low, **int** high) {

**int** pivot = arr[high];

**int** i = (low - 1);

**for** (**int** j = low; j <= high - 1; j++) {

**if** (arr[j] < pivot) {

            i++;

            swap(&arr[i], &arr[j]);

        }

    }

    swap(&arr[i + 1], &arr[high]);

**return** (i + 1);

}

**void** QuickSort(**int** arr[], **int** low, **int** high)

{

**if** (low < high) {

**int** pi = partition(arr, low, high);

        QuickSort(arr, low, pi - 1);

        QuickSort(arr, pi + 1, high);

    }

}

void DataDisplay(int \*array) {

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

printf("%d ",\*(array+i));

}

}

void WriteFile(int \*array) {

FILE \*fptr;

fptr = fopen("d2.txt","wb");

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

putw(\*(array+i), fptr);

}

fclose(fptr);

}

void main() {

int array[Num];

ReadFile(array);

printf("Original Data:\n");

DataDisplay(array);

QuickSort(array, count);

printf("\n\nSorted Data:\n");

DataDisplay(array);

WriteFile(array);

}

**Practice Activity – 7**

**Problem 7.1P:** Read the numbers from a text file sort them into an array using ‘Selection Sort’ algorithm and write back in another text file.

**Source Code:**

#include <stdio.h>

#define Num 100

static int count = 0;

void ReadFile(int \*array) {

FILE \*fp = fopen("d1.txt","r");

while(1) {

if(feof(fp)) {

break;

}

fscanf(fp, "%d", &array[count]);

count++;

}

fclose(fp);

printf("\t\*\*FILE read Successfully\*\*\n");

}

void swap(int \*array, int i, int j) {

int temp = array[i];

array[i] = array[j];

array[j] = temp;

}

**void** SelectionSort(**int** arr[], **int** n) {

**int** i, j, min\_idx;

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

        min\_idx = i;

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

**if** (arr[j] < arr[min\_idx])

            min\_idx = j;

        swap(&arr[min\_idx], &arr[i]);

    }

}

void DataDisplay(int \*array) {

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

printf("%d ",\*(array+i));

}

}

void WriteFile(int \*array) {

FILE \*fptr;

fptr = fopen("d2.txt","wb");

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

putw(\*(array+i), fptr);

}

fclose(fptr);

}

void main() {

int array[Num];

ReadFile(array);

printf("Original Data:\n");

DataDisplay(array);

SelectionSort(array, count);

printf("\n\nSorted Data:\n");

DataDisplay(array);

WriteFile(array);

}

**Problem 7.2P:** Read the numbers from a text file sort them into an array using ‘Merge Sort’ algorithm and write back in another text file.

**Source Code:**

#include <stdio.h>

#define Num 100

static int count = 0;

void ReadFile(int \*array) {

FILE \*fp = fopen("d1.txt","r");

while(1) {

if(feof(fp)) {

break;

}

fscanf(fp, "%d", &array[count]);

count++;

}

fclose(fp);

printf("\t\*\*FILE read Successfully\*\*\n");

}

void swap(int \*array, int i, int j) {

int temp = array[i];

array[i] = array[j];

array[j] = temp;

}

**void** merge(**int** arr[], **int** l, **int** m, **int** r) {

**int** i, j, k;

**int** n1 = m - l + 1;

**int** n2 = r - m;

**int** L[n1], R[n2];

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

        L[i] = arr[l + i];

**for** (j = 0; j < n2; j++)

         R[j] = arr[m + 1 + j];

i = 0;

     j = 0;

     k = l;

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

**if** (L[i] <= R[j]) {

            arr[k] = L[i];

            i++;

        }

**else** {

            arr[k] = R[j];

            j++;

        }

        k++;

    }

**while** (i < n1) {

         arr[k] = L[i];

         i++;

         k++;

    } **while** (j < n2) {

        arr[k] = R[j];

        j++;

        k++;

    }

}

**void** MergeSort(**int** arr[], **int** l, **int** r) {

**if** (l < r) {

**int** m = l + (r - l) / 2;

         MergeSort(arr, l, m);

         MergeSort(arr, m + 1, r);

        merge(arr, l, m, r);

    }

}

void DataDisplay(int \*array) {

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

printf("%d ",\*(array+i));

}

}

void WriteFile(int \*array) {

FILE \*fptr;

fptr = fopen("d2.txt","wb");

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

putw(\*(array+i), fptr);

}

fclose(fptr);

}

void main() {

int array[Num];

ReadFile(array);

printf("Original Data:\n");

DataDisplay(array);

MergeSort(array, count);

printf("\n\nSorted Data:\n");

DataDisplay(array);

WriteFile(array);

}

Experiment-08

**Title: Hash Tables**

**Objective:** Implementing hash tables with and without collision avoidance algorithms using array/linked list.

**Problem 8.1:** Implement a hash function on student SAP-ID and categorize them in to their 10 families based on the last three digits. Example: Student with SAP-ID 5000423 belongs to family 9 and student with SAP-ID 5000425 belongs to family 2 based on last three digits.

**Source Code:**

#include <stdio.h>

#include <stdlib.h>

struct set

{

int SAPID;

int MARKS;

};

struct set \*array;

int capacity = 10;

int size = 0;

int hashFunction(int SAPID){

int n,sum=0,m;

n = SAPID;

while(n>0){

m=n%10;

sum=sum+m;

n=n/10;

}

sum = sum-5;

return 0;

};

int checkPrime(int n){

int i;

if (n == 1 || n == 0){

return 0;

}

for (i = 2; i < n / 2; i++){

if (n % i == 0){

return 0;

}

}

}

int getPrime(int n){

if (n % 2 == 0){

n++;

}

while (!checkPrime(n)){

n += 2;

}

return n;

}

void init\_array(){

capacity = getPrime(capacity);

array = (struct set \*)malloc(capacity \* sizeof(struct set));

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

array[i].SAPID = 0;

array[i].MARKS = 0;

}

}

void insert(int SAPID, int MARKS){

int index = hashFunction(SAPID);

if (array[index].MARKS == 0){

array[index].SAPID = SAPID;

array[index].MARKS = MARKS;

size++;

printf("\n Key (%d) has been inserted \n", SAPID);

}

else if (array[index].SAPID == SAPID){

array[index].MARKS = MARKS;

}

else{

printf("\n Collision occured \n");

}

}

void remove\_element(int SAPID){

int index = hashFunction(SAPID);

if (array[index].MARKS == 0) {

printf("\n This SAPID does not exist \n");

}

else{

array[index].SAPID = 0;

array[index].MARKS = 0;

size--;

printf("\n Key (%d) has been removed \n", SAPID);

}

}

void display(){

int i;

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

if (array[i].MARKS == 0){

printf("\n array[%d]: / ", i);

}

else {

printf("\n SAPID: %d array[%d]: %d \t", array[i].SAPID, i, array[i].MARKS);

}

}

}

int size\_of\_hashtable(){

return size;

}

int main(){

int choice, SAPID, MARKS, n;

int c = 0;

init\_array();

do {

printf("1.Insert item in the Hash Table"

"\n2.Remove item from the Hash Table"

"\n3.Check the size of Hash Table"

"\n4.Display a Hash Table"

"\n\n Please enter your choice: ");

scanf("%d", &choice);

switch (choice){

case 1:

printf("Enter SAPID -:\t");

scanf("%d", &SAPID);

printf("Enter MARKS -:\t");

scanf("%d", &MARKS);

insert(SAPID, MARKS);

break;

case 2:

printf("Enter the SAPID to delete-:");

scanf("%d", &SAPID);

remove\_element(SAPID);

break;

case 3:

n = size\_of\_hashtable();

printf("Size of Hash Table is-:%d\n", n);

break;

case 4:

display();

break;

default:

printf("Invalid Input\n");

}

printf("\nDo you want to continue (press 1 for yes): ");

scanf("%d", &c);

} while (c == 1);

}

**8.2)IMPLEMENT A HASH TABLE USING ARRAYS.PERFORM INSERT, DELETE AND SEARCH OPERATIONS ON THE HASH TABLE USING THE ABOVE HASH FUNCTION AND WITH LINEAR PROBING AS COLLISION AVOIDANCE STRATEGY**.

**SOURCE CODE:**

#include <stdio.h>

#include <stdlib.h>

struct set

{

int SAPID;

int MARKS;

};

struct set \*array;

int capacity = 10;

int size = 0;

int hashFunction(int SAPID){

int n,sum=0,m;

n = SAPID;

while(n>0){

m=n%10;

sum=sum+m;

n=n/10;

}

sum = sum-5;

return 0;

};

int checkPrime(int n){

int i;

if (n == 1 || n == 0){

return 0;

}

for (i = 2; i < n / 2; i++){

if (n % i == 0){

return 0;

}

}

}

int getPrime(int n){

if (n % 2 == 0){

n++;

}

while (!checkPrime(n)){

n += 2;

}

return n;

}

void init\_array(){

capacity = getPrime(capacity);

array = (struct set \*)malloc(capacity \* sizeof(struct set));

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

array[i].SAPID = 0;

array[i].MARKS = 0;

}

}

void insert(int SAPID, int MARKS){

int index = hashFunction(SAPID);

if (array[index].MARKS == 0){

array[index].SAPID = SAPID;

array[index].MARKS = MARKS;

size++;

printf("\n Key (%d) has been inserted \n", SAPID);

}

else if (array[index].SAPID == SAPID){

array[index].MARKS = MARKS;

}

else{

printf("\n Collision occured \n");

}

}

void remove\_element(int SAPID){

int index = hashFunction(SAPID);

if (array[index].MARKS == 0) {

printf("\n This SAPID does not exist \n");

}

else{

array[index].SAPID = 0;

array[index].MARKS = 0;

size--;

printf("\n Key (%d) has been removed \n", SAPID);

}

}

void display(){

int i;

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

if (array[i].MARKS == 0){

printf("\n array[%d]: / ", i);

}

else {

printf("\n SAPID: %d array[%d]: %d \t", array[i].SAPID, i, array[i].MARKS);

}

}

}

int size\_of\_hashtable(){

return size;

}

int main(){

int choice, SAPID, MARKS, n;

int c = 0;

init\_array();

do {

printf("1.Insert item in the Hash Table"

"\n2.Remove item from the Hash Table"

"\n3.Check the size of Hash Table"

"\n4.Display a Hash Table"

"\n\n Please enter your choice: ");

scanf("%d", &choice);

switch (choice){

case 1:

printf("Enter SAPID -:\t");

scanf("%d", &SAPID);

printf("Enter MARKS -:\t");

scanf("%d", &MARKS);

insert(SAPID, MARKS);

break;

case 2:

printf("Enter the SAPID to delete-:");

scanf("%d", &SAPID);

remove\_element(SAPID);

break;

case 3:

n = size\_of\_hashtable();

printf("Size of Hash Table is-:%d\n", n);

break;

case 4:

display();

break;

default:

printf("Invalid Input\n");

}

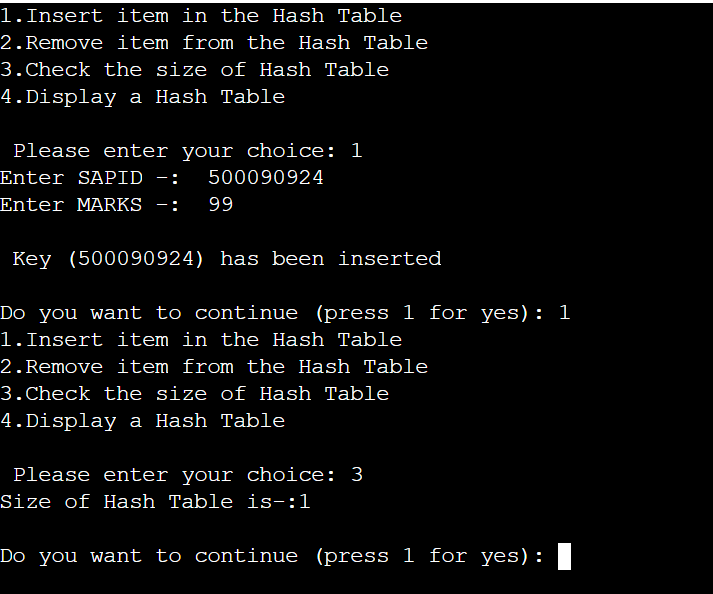
printf("\nDo you want to continue (press 1 for yes): ");

scanf("%d", &c);

} while (c == 1);

}

**SAMPLE OUTPUT**:

****