**ASSIGNMENT SET 2**

Name: IMON RAJ

Class Roll: 002010501098

Class: BCSE II

Sem: First

Session: 2021-2022

**Problem No. 1**

**Problem Statement**

Define an ADT for Polynomials. Write C data structure representation and functions for the operations on the Polynomials in a

Header file. Write a menu-driven main program in a separate file for testing the different operations and include the above header file.

**Solution Approach:**

For storing a polynomial in an array. We will create first an array of max fixed size and we will consider the array index to be the power of terms of a polynomial and array element value to be the co-efficient. We will do arithmetic operation like addition and subtraction of polynomials by traversing the polynomials array do operations in respective same powered elements. For multiplication we will need two loops for multiplying each element of one polynomial to each of other one. Each multiplication value will be added to corresponding power’s co-efficient in the resultant polynomial.

While printing we will not print the zero co-effs and while initialization, we will keep each co-efficient value to zero.

**Structured Pseudocode**

**typedef struct poly**

**{**

**float arr[MAX\_POWER + 1];** /\*MAXIMUM 100 POWERS POSSIBLE FOR A POLYNOMIAL\*/

**} Polynomial;**

**Polynomial Take\_Polynomial()** /\*TAKE THE VALUE OF A POLYNOMIAL\*/

**{**

**Polynomial p;**

**int i, choice, power;**

**float coeff;**

**for (i = 0; i < MAX\_POWER + 1; i++)**

**{**

**p.arr[i] = 0;**

**}**

**do**

**{**

**print("Enter the power: ");**

**take(power);**

**if (power > MAX\_POWER)**

**{**

**print("Max Power Available is MAX\_POWER”);**

**choice = 1;**

**continue;**

**}**

**print("Enter the co-efficient of x^power”);**

**take(coeff);**

**p.arr[power] = coeff;**

**print("To add one more power term press 1 or 0 to stop: ");**

**take(choice);**

**} while (choice == 1);**

**return p;**

**}**

**void Print\_polynomial(Polynomial p)** /\*PRINTS A POLYNOMIAL IN NORMAL READABLE FORMAT\*/

**{**

**int i, flag = 0;**

**for (i = MAX\_POWER; i >= 0; i--)**

**{**

**if (p.arr[i] != 0.0)**

**{**

**if (flag)**

**{**

**print(" + ");**

**}**

**if (p.arr[i] != 1 && p.arr[i] != (-1))**

**{**

**print(p.arr[i]);**

**}**

**if (i != 0 && i != 1)**

**{**

**print("x^ i”);**

**}**

**else if (i == 1)**

**{**

**print("x");**

**}**

**flag = 1;**

**}**

**}**

**if (!flag)**

**{**

**print(" 0");**

**}**

**}**

**Polynomial Add\_Polynomials(Polynomial p1, Polynomial p2)** /\*ADDS TWO POLYNOMIALS\*/

**{**

**Polynomial p3;**

**int i;**

**for (i = 0; i < MAX\_POWER + 1; i++)**

**{**

**p3.arr[i] = p1.arr[i] + p2.arr[i];**

**}**

**return p3;**

**}**

**Polynomial Subtract\_Polynomials(Polynomial p1, Polynomial p2)** /\*SUBTRACTION OF TWO POLYNOMIALS\*/

**{**

**Polynomial p3;**

**int i;**

**for (i = 0; i < MAX\_POWER + 1; i++)**

**{**

**p3.arr[i] = p1.arr[i] - p2.arr[i];**

**}**

**return p3;**

**}**

**Polynomial Multiply\_Polynomials(Polynomial p, Polynomial q)** /\*MULTIPLY TWO POLYNOMIALS\*/

**{**

**Polynomial result;**

**int i, j, out\_of\_range=0;**

**float mult;**

**for (i = 0; i < MAX\_POWER + 1; i++)**

**{**

**result.arr[i] = 0;**

**}**

**for (i = 0; i < MAX\_POWER + 1; i++)**

**{**

**if(p.arr[i]==0.0){**

**continue;**

**}**

**for (j = 0; j < MAX\_POWER + 1; j++)**

**{**

**if(q.arr[j]==0.0){**

**continue;**

**}**

**if((i+j)>100) {**

**print("Power after multply is out of range..");**

**out\_of\_range=1;**

**break;**

**}**

**mult = (p.arr[i])\*(q.arr[j]);**

**result.arr[i+j] += mult;**

**}**

**if(out\_of\_range){**

**break;**

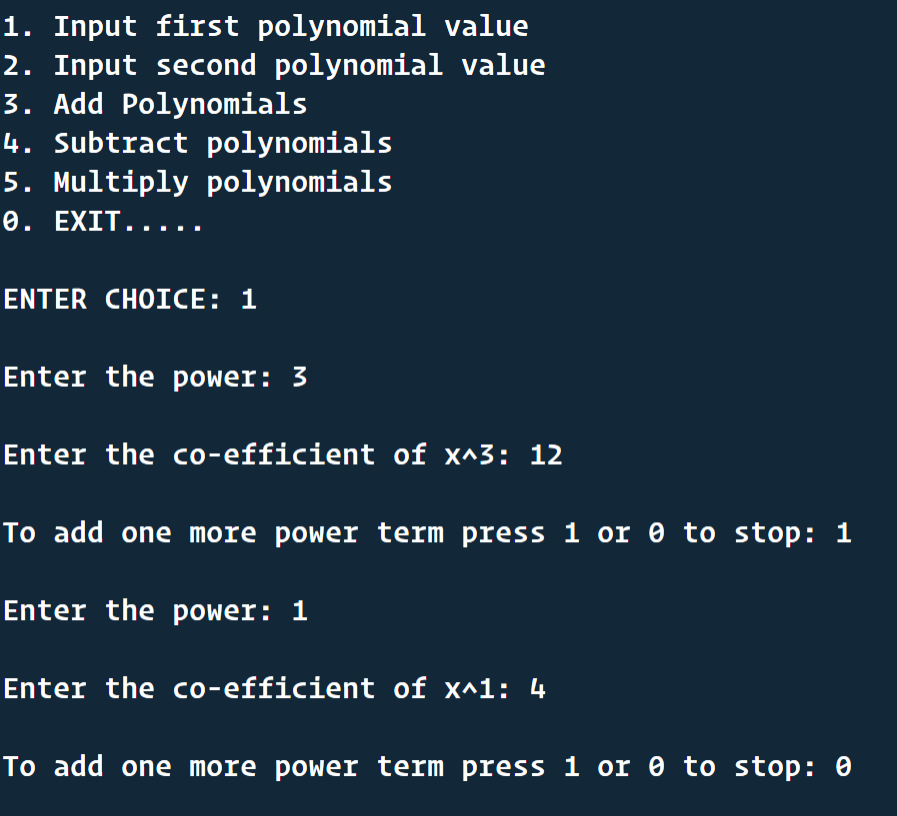
**}**

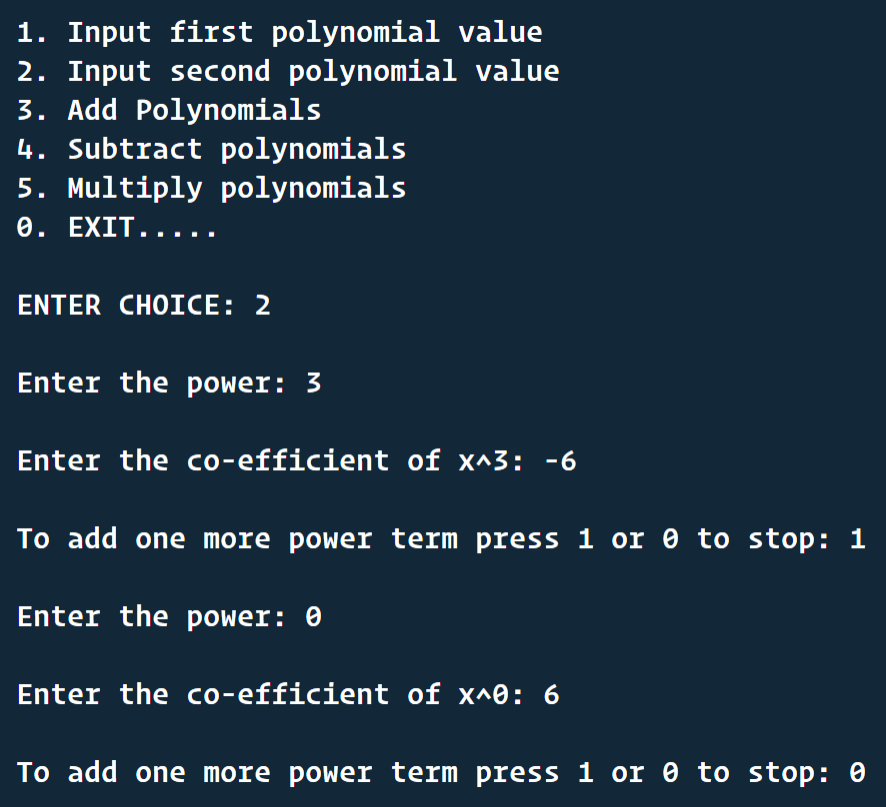
**}**

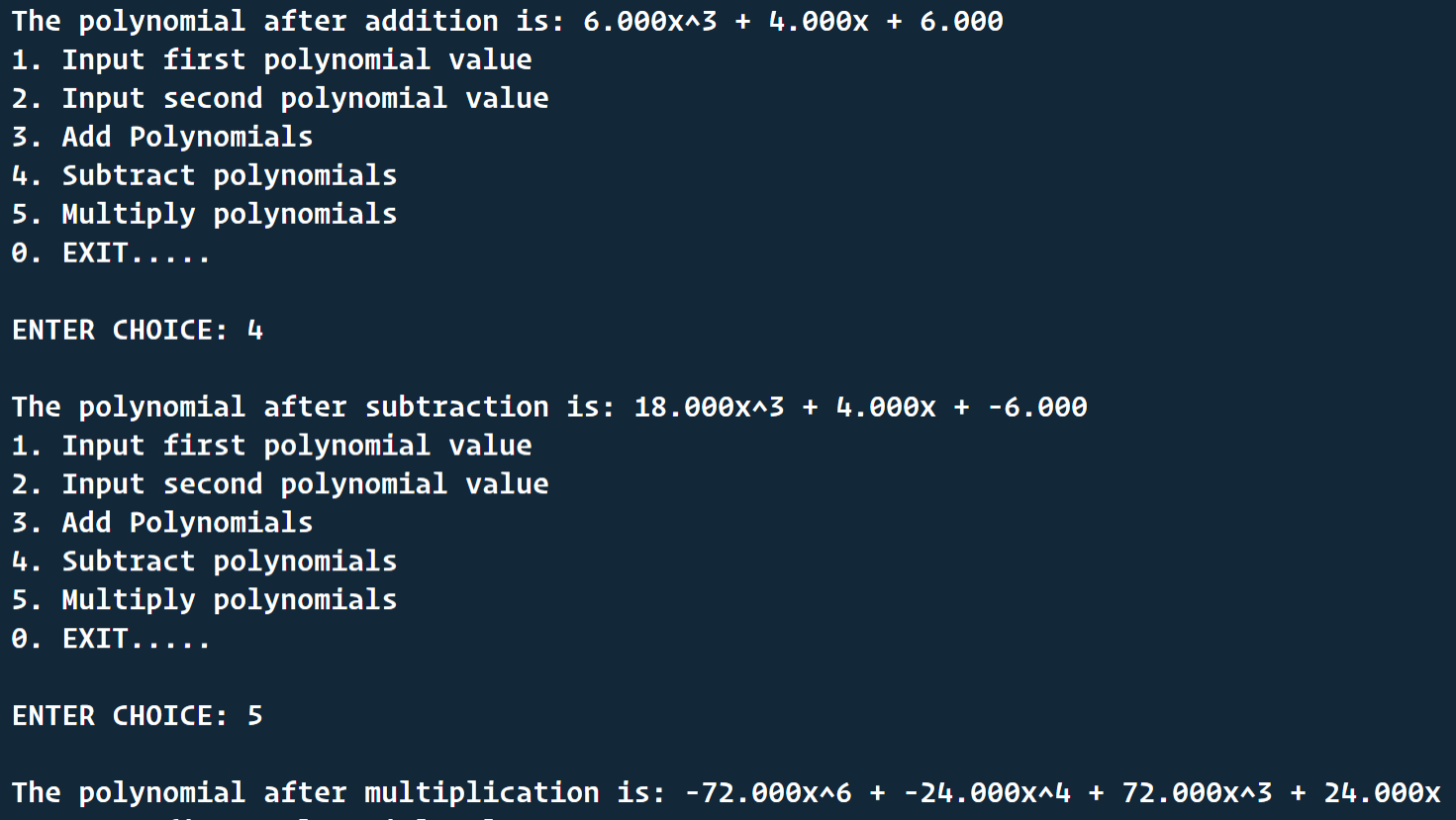
**return result;**

**}**

**Results:** In the menu driven program we can pass polynomials and do the operations using the above algorithms programmed in the header file. That will show us correct results and also notify us if there is power out of bound.







**Discussion:** This program helps us to understand how a data type can be created using struct and how different operations can be done on that type without letting the end user know what is happening inside. Creating a header file increase modularity in our program as the header file and main C file are separate.

**Problem No. 2**

**Problem Statement**

Define an ADT for Sparse Matrix. Write C data structure representation and functions for the operations on the Sparse Matrix in a Header file. Write a menu-driven main program in a separate file for testing the different operations and include the above header file

**Solution Approach:**

Number of non-zero elements, transpose of sparse matrix, creating tuple,displaying tuple. First we create a struct name sparse whose attributes are an array 'sp' and the number of rows of the newly formed sparse matrix which is equal to number of non-zero elements + 1 for keeping track of number of rows and columns of actual matrix. Then , we counted the number of non-zero elements and keep the count in a variable. After that, we created tuplefor non-zero elements, where the first and second element of each row denotes the row and column number of the elements in the actual matrix and in the last entry of a row we stored the element. We have a display function to show the actual matrix and the sparse matrix. Then we created a function to transpose this sparse matrix.First, we allocated memory required to store the elements in the target 3-tuple.Then we store the number of rows and columns.The transpose operation is carried out through a pair of for loops.the oure for loop runs till the non-zero elements of col number of columns are not scanned. In the inner for loop first we have obtained the position at which the column number of a non-zero element is stored . Then we have checked whether the column number of a non-zero element matches with the column numbercurrently being considered.Another variable is used for the tuple, to store the position at which data from souce tuple, should get copied. Similarly another variable is used for the tuple, to extract data from it. Then we copied the column position of a non-zero element from source tuple.This column number gets stored at the row position in target tuple. On similar lines the row position of a non-zero element of source tuple is copied at the column position of the target tuple.

**Structured Code(The header file)**

**#define MAX1 3**

**#define MAX2 3**

**typedef struct sparsetype**

**{**

**int \*sp;**

**int row;**

**} sparse;**

**void init(sparse \*p)**

**{**

**p->sp = NULL;**

**}**

**void creat\_array(sparse \*p)**

**{**

**int i, n;**

**p->sp = (int \*)malloc(MAX1 \* MAX2 \* sizeof(int));**

**for (i = 0; i < MAX1 \* MAX2; i++)**

**{**

**printf("Enter the elements no. %d : ", i);**

**scanf("%d", &n);**

**\*(p->sp + i) = n;**

**}**

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

**}**

**void display(sparse p)**

**{**

**int i;**

**for (i = 0; i < MAX1 \* MAX2; i++)**

**{**

**if (i % MAX2 == 0)**

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

**printf("%d ", p.sp[i]);**

**}**

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

**}**

**int count(sparse p)**

**{**

**int count = 0, i;**

**for (i = 0; i < MAX1 \* MAX2; i++)**

**{**

**if (p.sp[i] != 0)**

**count++;**

**}**

**return count;**

**}**

**void creat\_tuple(sparse \*p, sparse s)**

**{**

**int r = 0, c = -1, l = -1, i;**

**p->row = count(s) + 1;**

**p->sp = (int \*)malloc(p->row \* 3 \* sizeof(int));**

**p->sp[0] = MAX1;**

**p->sp[1] = MAX2;**

**p->sp[2] = p->row - 1;**

**l = 2;**

**for (i = 0; i < MAX1 \* MAX2; i++)**

**{**

**c++;**

**if (i % MAX2 == 0 && i != 0)**

**{**

**r++;**

**c = 0;**

**}**

**if (s.sp[i] != 0)**

**{**

**l++;**

**p->sp[l] = r;**

**l++;**

**p->sp[l] = c;**

**l++;**

**p->sp[l] = s.sp[i];**

**}**

**}**

**}**

**void display\_tuple(sparse p)**

**{**

**int i;**

**for (i = 0; i < p.row \* 3; i++)**

**{**

**if (i % 3 == 0)**

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

**printf("%d\t", p.sp[i]);**

**}**

**}**

**void delsparse(sparse \*p)**

**{**

**free(p->sp);**

**}**

**void transpose(sparse \*p, sparse s)**

**{**

**int x, q, pos\_1, pos\_2, col, elem, c, y;**

**p->sp = (int \*)malloc(s.row \* 3 \* sizeof(int));**

**p->row = s.row;**

**p->sp[0] = s.sp[1];**

**p->sp[1] = s.sp[0];**

**p->sp[2] = s.sp[2];**

**col = p->sp[1];**

**elem = p->sp[2];**

**if (elem <= 0)**

**return;**

**x = 1;**

**for (c = 0; c < col; c++)**

**{**

**for (y = 1; y <= elem; y++)**

**{**

**q = y \* 3 + 1;**

**if (s.sp[q] == c)**

**{**

**pos\_2 = x \* 3 + 0;**

**pos\_1 = y \* 3 + 1;**

**p->sp[pos\_2] = s.sp[pos\_1];**

**pos\_2 = x \* 3 + 1;**

**pos\_1 = y \* 3 + 0;**

**p->sp[pos\_2] = s.sp[pos\_1];**

**pos\_2 = x \* 3 + 2;**

**pos\_1 = y \* 3 + 2;**

**p->sp[pos\_2] = s.sp[pos\_1];**

**x++;**

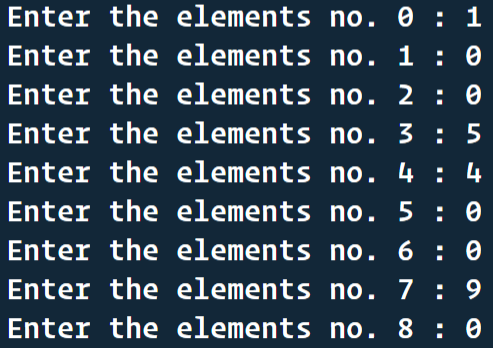
**}**

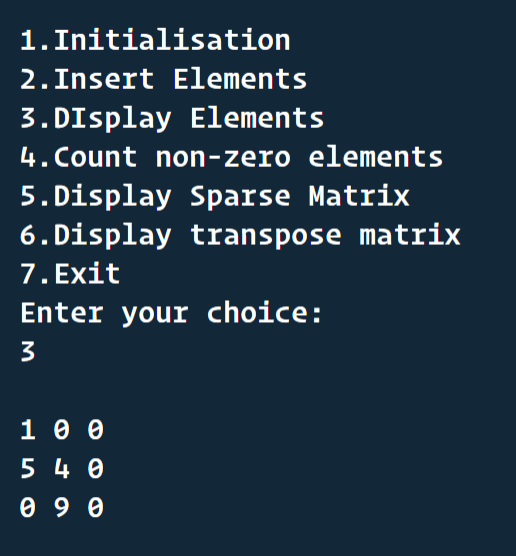
**}**

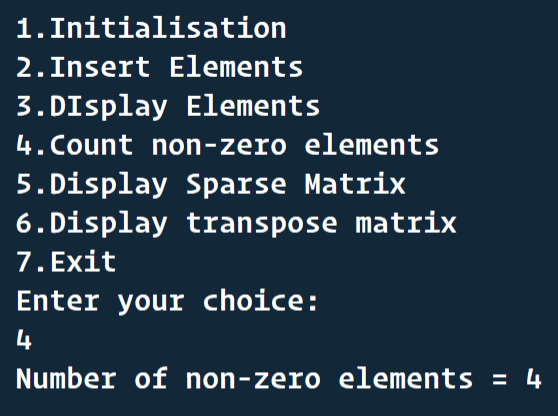
**}**

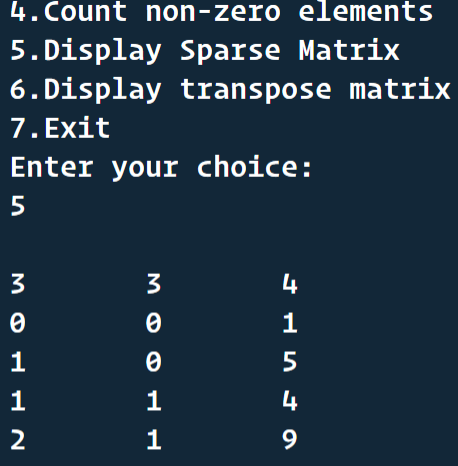
**}**

**Results:** In the menu driven program we can Create a sparse matrix and see how is it sparse and also find number of zeros in the sparse matrix. We can show the transpose of the matrix.









**Discussion:** This program helps us to understand how a data type can be created using struct and how different operations can be done on that type without letting the end user know what is happening inside. Creating a header file increase modularity in our program as the header file and main C file are separate.

**Problem No. 3**

**Problem Statement**

Define an ADT for List. Write C data structure representation and functions for the operations on the List in a Header file with array as the base data structure. Write a menu-driven main program in a separate file for testing the different operations and include the above header file. Two data structures with and without using sentinels in arrays are to be implemented.

**Solution Approach:**

For storing list elements, we will use array. And in the list ADT structure definition, we will have size and capacity to know whether it is empty or full or none. While inserting value we need to do a shift operation in the array, and in the deleting process we also have to do a shift. Inserting and deleting in the front is time consuming because of elements shifting. Finding an element is a linear time operation here. We can check if the list is empty or full just checking the size and capacity of list.

**Structured Pseudocode**

**struct list**

**{**

**int capacity;**

**int size;**

**int \*array;**

**};**

**typedef struct list \*ptrToNode;**

**typedef ptrToNode LIST;**

**typedef int POSITION;**

**int Isempty(LIST L)**

**{**

**return L->size == 0;**

**}**

**void MakeEmpty(LIST L)**

**{**

**if (Isempty(L))**

**print("LIST is already Empty");**

**else**

**{**

**L->size = 0;**

**print("Now List becomes Empty");**

**}**

**}**

**LIST Createlist(int max)**

**{**

**LIST L;**

**L = (struct list \*)malloc(sizeof(struct list));**

**if (L == NULL)**

**print("Fatal Error");**

**else**

**{**

**L->capacity = max;**

**L->array = (int \*)malloc(sizeof(int) \* max);**

**if (L->array == NULL)**

**print("Fatal Error");**

**else**

**{**

**L->size = 0;**

**print("List is Created successfully");**

**}**

**}**

**return L;**

**}**

**int Isfull(LIST L)**

**{**

**return L->size == L->capacity;**

**}**

**void Insert(int x, LIST L, POSITION P)**

**{**

**int i;**

**if (Isfull(L))**

**print("List is Full");**

**else**

**{**

**for (i = L->size - 1; i >= P; i--)**

**L->array[i + 1] = L->array[i];**

**L->size++;**

**L->array[P] = x;**

**}**

**}**

**POSITION Findprevious(int x, LIST L)**

**{**

**POSITION P;**

**P = -1;**

**while (P != L->size && L->array[P + 1] != x)**

**{**

**P++;**

**}**

**return P;**

**}**

**POSITION Find(int x, LIST L)**

**{**

**POSITION P;**

**P = 0;**

**int found = 0;**

**while (P != L->size && L->array[P] != x)**

**{**

**P++;**

**}**

**if (L->array[P] == x)**

**{**

**found = 1;**

**}**

**return (found) ? P : (-1);**

**}**

**void Delete(int x, LIST L)**

**{**

**int i;**

**POSITION P;**

**P = Find(x, L);**

**if (P == L->size)**

**print("\n Element not found in the list");**

**else**

**{**

**for (i = P; i < L->size; i++)**

**L->array[i] = L->array[i + 1];**

**L->size--;**

**}**

**}**

**LIST Deletelist(LIST L)**

**{**

**MakeEmpty(L);**

**free(L);**

**L = NULL;**

**return L;**

**}**

**void Display(LIST L)**

**{**

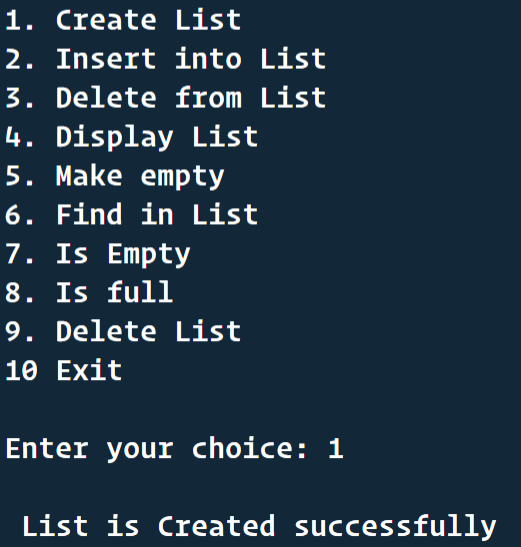
**int i;**

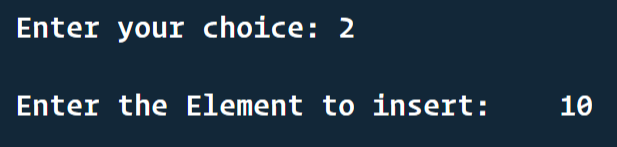
**for (i = 0; i < L->size; i++)**

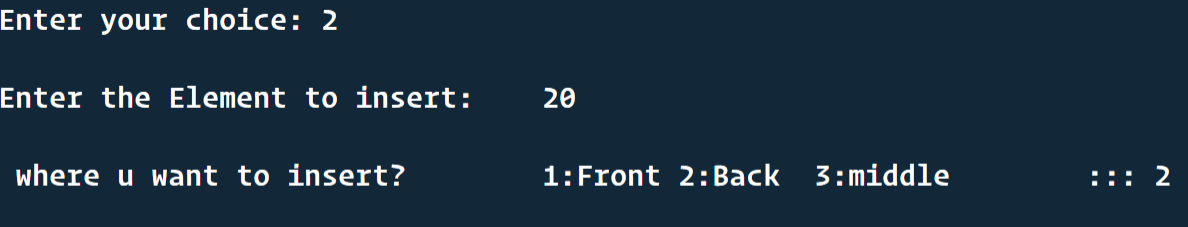
**print( L->array[i]);**

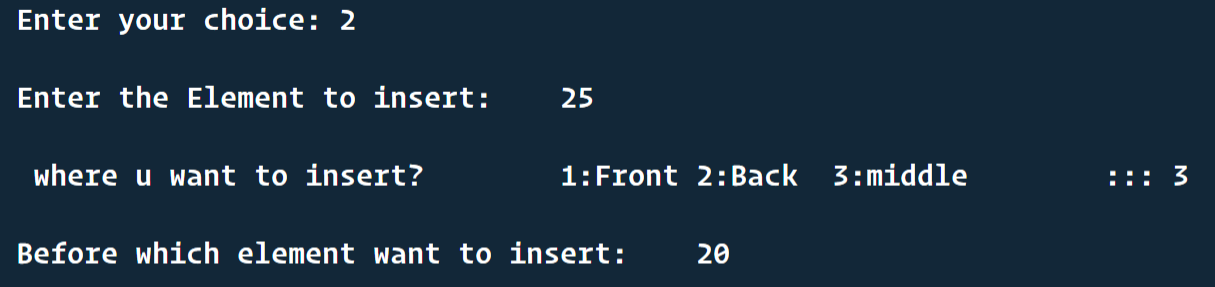
**}**

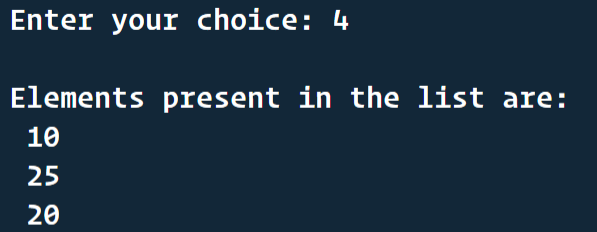
**Results:** In the menu driven program we can Create delete and update list. All the operations’ codes are in the header file. We can insert the elements in the front , back and middle of list. We can delete an element by specifying the element’s value. At any point in the menu driven program we can check the emptiness of the list and make it empty whenever needed. As this is a list it can store duplicate values.

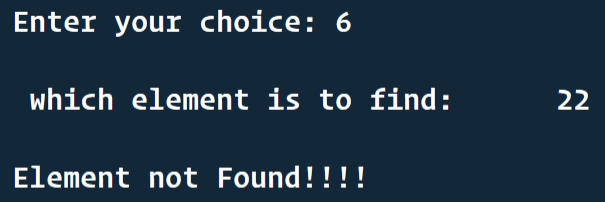


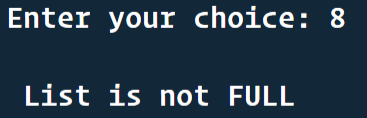


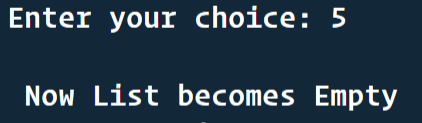












**Discussion:** This program helps us to understand how a data type can be created using struct and how different operations can be done on that type without letting the end user know what is happening inside. Creating a header file increase modularity in our program as the header file and main C file are separate.

**Problem No. 4**

**Problem Statement**

Define an ADT for Set.

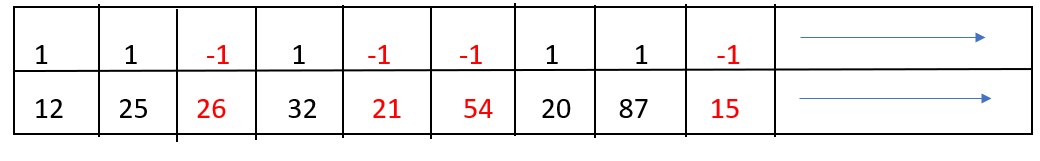
Write C data representation and functions for the operations on the Set in a Header file, with array as the base data structure. Write a menu-driven main program in a separate file for testing the different operations and include the above header file.

**Solution Approach:**

For creating an efficient Set Data type, considering all the properties of a Set we will use a two-dimensional array with 2 rows. Here one row will store the element value. And the other row will be a storage for flag value which will indicate whether an inner element is deleted or not. The flag will be 1 if the element is valid and will be -1 when the element is deleted, and in that particular column in the value row another element can be inserted. While inserting we will make sure that the element with same value is already there or not because the set can’t contain duplicate values.

For Union we will create a new set, copy first set elements to the new set and while copying the second set we will take care of duplicate values.

For intersection we will just insert the common elements to the output set.



***RED VALUES ARE DELETED AS FLAG IS -1. WE CAN INSERT***

***NEW ELEMENTS THERE MAKING THE FLAG 1.***

**Header File Code:**

**typedef struct set**

**{**

**int arr[2][MAX\_SET\_ELEMENTS];**

**int end\_position;**

**} Set;**

**Set Initialize\_Set()**

**{**

**Set AA;**

**int i;**

**for (i = 0; i < MAX\_SET\_ELEMENTS; i++)**

**{**

**AA.arr[1][i] = -1;**

**AA.arr[0][i] = 0;**

**}**

**AA.end\_position = 0;**

**return AA;**

**}**

**void Insert\_Element\_In\_Set(Set \*pS, int value)**

**{**

**int i;**

**for (i = 0; i <= (\*pS).end\_position; i++)**

**{**

**if ((\*pS).arr[1][i] == 1)**

**{**

**if ((\*pS).arr[0][i] == value)**

**{**

**printf("\nElement already exists in the set..");**

**return;**

**}**

**}**

**}**

**for (i = 0; i <= (\*pS).end\_position + 1; i++)**

**{**

**if (i > (MAX\_SET\_ELEMENTS - 1))**

**{**

**printf("\nSet is full..");**

**return;**

**}**

**if ((\*pS).arr[1][i] == (-1))**

**{**

**(\*pS).arr[1][i] = 1;**

**(\*pS).arr[0][i] = value;**

**break;**

**}**

**}**

**if (i == ((\*pS).end\_position + 1))**

**{**

**(\*pS).end\_position = i;**

**}**

**}**

**void Delete\_Element\_In\_Set(Set \*pS, int value)**

**{**

**int i;**

**for (i = 0; i <= (\*pS).end\_position; i++)**

**{**

**if ((\*pS).arr[1][i] == 1)**

**{**

**if ((\*pS).arr[0][i] == value)**

**{**

**(\*pS).arr[1][i] = -1;**

**(\*pS).arr[0][i] == 0;**

**printf("\nElement successfully deleted from the set..");**

**return;**

**}**

**}**

**}**

**printf("\nElement doesn't exist in set to delete..");**

**}**

**int Set\_Contains(Set s, int value)**

**{**

**int i;**

**for (i = 0; i <= s.end\_position; i++)**

**{**

**if (s.arr[1][i] == 1)**

**{**

**if (s.arr[0][i] == value)**

**{**

**return 1;**

**}**

**}**

**}**

**return 0;**

**}**

**Set Union\_Of\_Sets(Set s1, Set s2)**

**{**

**Set union\_set = Initialize\_Set();**

**int i, j = 0;**

**for (i = 0; i <= s1.end\_position; i++)**

**{**

**if (s1.arr[1][i] == 1)**

**{**

**if (!Set\_Contains(s2, s1.arr[0][i]))**

**{**

**Insert\_Element\_In\_Set(&union\_set, s1.arr[0][i]);**

**j++;**

**}**

**}**

**}**

**for (i = 0; i <= s2.end\_position; i++)**

**{**

**if (s2.arr[1][i] == 1)**

**{**

**if(j==MAX\_SET\_ELEMENTS){**

**printf("\nThe union set exceeds set capacity..");**

**union\_set = Initialize\_Set();**

**return union\_set;**

**}**

**Insert\_Element\_In\_Set(&union\_set, s2.arr[0][i]);**

**j++;**

**}**

**}**

**return union\_set;**

**}**

**Set Intersection\_Of\_Sets(Set s1, Set s2){**

**Set intersection\_set = Initialize\_Set();**

**int i;**

**for (i = 0; i <= s1.end\_position; i++)**

**{**

**if (s1.arr[1][i] == 1)**

**{**

**if (Set\_Contains(s2, s1.arr[0][i]))**

**{**

**Insert\_Element\_In\_Set(&intersection\_set, s1.arr[0][i]);**

**}**

**}**

**}**

**return intersection\_set;**

**}**

**void Print\_Set(Set S)**

**{**

**int i, flag=0;**

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

**for (i = 0; i <= S.end\_position; i++)**

**{**

**if (S.arr[1][i] == 1)**

**{**

**if(flag){**

**printf(",");**

**}else{**

**flag=1;**

**}**

**printf(" %d ", S.arr[0][i]);**

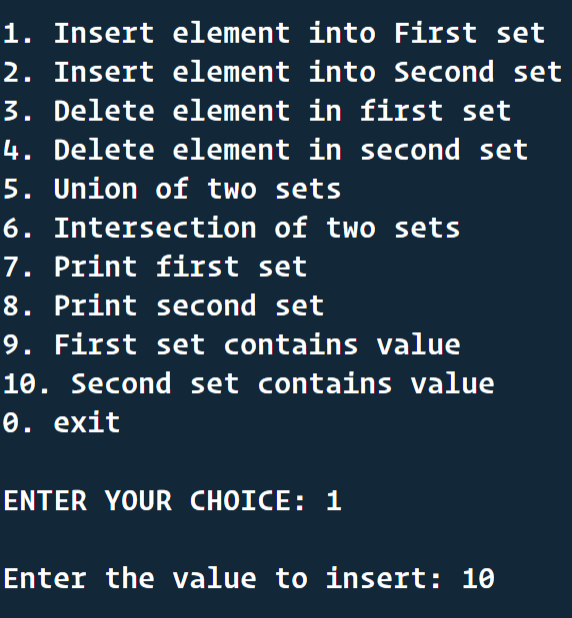
**}**

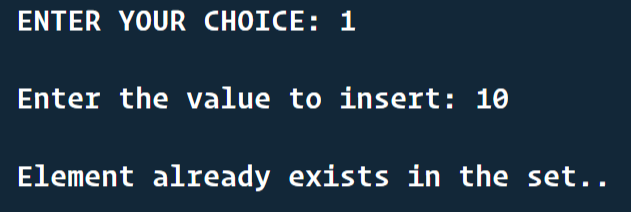
**}**

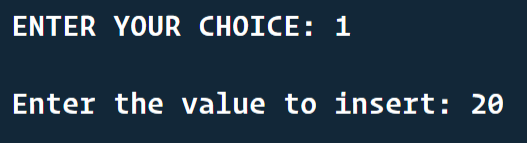
**printf("}");**

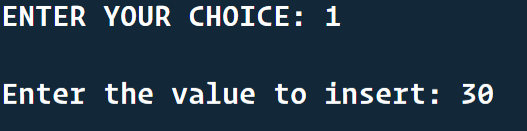
**}**

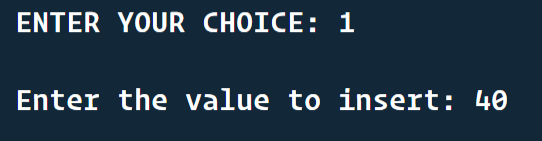
**Results:** In the menu driven program we can Create sets insert elements in them and find intersection and union sets of them. We can print the sets anytime. We can even check whether a set contains a particular element or not. As there is a maximum size of set in our program, the number of elements in the union set is taken care of.

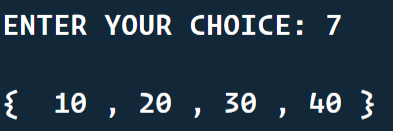


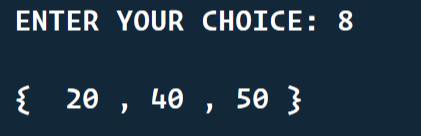


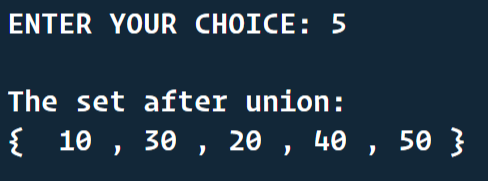


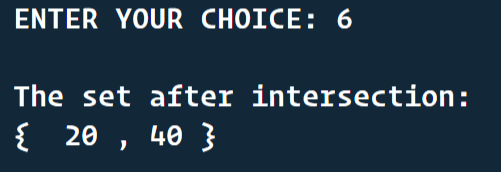


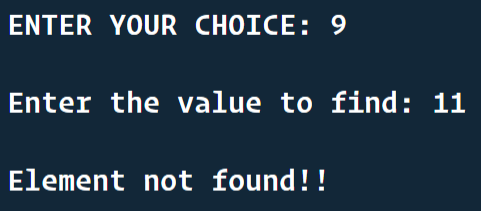


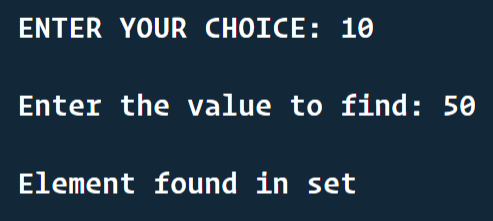












**Discussion:** This program helps us to understand how a data type can be created using struct and how different operations can be done on that type without letting the end user know what is happening inside. Here we get the idea of logical deletion of an element using a flag variable which indicates the validity of an element at a particular time instant. Creating a header file increase modularity in our program as the header file and main C file are separate.

**Problem No. 5**

**Problem Statement**

Define an ADT for String.

Write C data representation and functions for the operations on the String in a Header file, with array as the base data structure, without using any inbuilt function in C. Write a menu-driven main program in a separate file for testing the different operations and include the above header file.

**Solution Approach:**

For String ADT, we will use char array internally with a max length. We can do the mentioned things –

* Take string input
* Copy a string to another
* Compare a string to another(lexicographically)
* Convert a string to uppercase or lowercase (considering ASCII values)
* We can concatenate two strings
* Find out the length of string

**Pseudo Code:**

**typedef struct string {** /\*INTERNALLY USING CHAR ARRAY TO STORE STRINGS\*/

**char str[MAX\_STRING\_LENGTH];**

**}String;**

**String take\_string() {**

**String ss;**

**char c;**

**while( scanf(“%c”, &c) AND c != ‘\n’){** /\* STORING INTO STRING UNTIL USER PRESSES NEW LINE \*/

**ss.str[i] = c;**

**i++;**

**if(i==MAX\_STRING\_LENGTH-1){** /\* TAKING CARE OF ARRAY OVERFLOW \*/

**break;**

**}**

**}**

**ss.str[i] = '\0';** /\* PUTTING ‘\0’ AFTER USER HAS GIVEN INPUT TO TERMINATE STRING \*/

**return ss;**

**}**

**String copy\_string(String s){**

**String s2;**

**int i=0;**

**while(s.str[i] != '\0'){** /\* COPYING CHARACTERS FROM FIRST STRING TO SECOND STRING BEFORE \0 OCCURS \*/

**s2.str[i] = s.str[i];**

**i++;**

**}**

**s2.str[i] = '\0';** /\* PUTTING A \0 AT LAST \*/

**return s2;**

**}**

**void print\_string(String s){**

**int i=0;**

**while(s.str[i] != '\0'){**

**print(s.str[i]);** /\* PRINTING ONE BY ONE CHARACTERS \*/

**i++;**

**}**

**}**

**int length\_string(String s){**

**int i=0;**

**while(s.str[i] != '\0'){**

**i++;** /\* COUNTING STRING LENGTH EXCLUDING \0 \*/

**}**

**return i;**

**}**

**int compare\_string(String s1, String s2){**

**int i=0;**

**while(s1.str[i] != '\0' AND s2.str[i] != '\0'){** /\* LOOP UNTIL ONE STRING ENDS TRAVERSING \*/

/\* AT ANY INSTANCE OR POSITION IF WE GET FIRST STRING’S CHARACTER IS GREATER/SMALLER WE WILL \*/

/\* RETURN ACCORDINGLY BUT IF THEY ARE EQUAL WE WILL KEEP TRAVERSING \*/

**if(s1.str[i] > s2.str[i]){**

**return 1;**

**} else if(s1.str[i] < s2.str[i]){**

**return -1;**

**}**

**i++;**

**if(i==MAX\_STRING\_LENGTH){**

**return 0;**

**}**

**}**

/\* WHEN SHORTER STRING ENDS, IF ALL CHARCTERS OF THE SHORTER STRING IS

EQUAL TO LONGER STRING AND LONGER STRING HAS SOME MORE

CHARACTERS, THEN LONGER STRING WILL BE LARGER\*/

**if(s1.str[i] !='\0' && s2.str[i] == '\0'){**

**return 1;**

**}else if(s1.str[i] =='\0' && s2.str[i] != '\0'){**

**return -1;**

**}else{**

**return 0;**

**}**

**}**

**String into\_uppercase(String s){**

**int i=0;**

**while (s.str[i] != '\0')**

**{**

**if(s.str[i] >= 97 AND s.str[i] <= 122){** /\* IF A CHARACTER IS LOWERCASE THEN ONLY DO ASCII OPERATION TO DO UPPERCASE \*/

**s.str[i] -= 32;** /\* CONVERTING INTO UPPERCASE USING ASCII VALUES \*/

**}**

**i++;**

**}**

**return s;**

**}**

**String into\_lowercase(String s){** /\* SIMILAR METHODOLOGY AS TO\_UPPERCASE \*/

**int i=0;**

**while (s.str[i] != '\0')**

**{**

**if(s.str[i] >= 65 && s.str[i] <= 90){**

**s.str[i] += 32;**

**}**

**i++;**

**}**

**return s;**

**}**

**String concat\_string(String s1, String s2){**

**String sss;**

**int len1 = length\_string(s1);**

**int len2 = length\_string(s2);**

**int length = len1+len2;**

**if(length>MAX\_STRING\_LENGTH){** /\* CHECKING IF THE CONCATED STRING LENGTH DOESN’T OVERFLOW \*/

**printf("\nThe concated string is out of size range..");**

**return sss;**

**}**

**int i=0;**

**for(i=0; i<len1; i++){** /\* COPYING FIRST STRING TO THE BEGINNING OF OUTPUT STRING \*/

**sss.str[i] = s1.str[i];**

**}**

**for(i=0; i<len2; i++){** /\* ADDING SECOND STRING CHARACTERS AFTER FIRST STRING CHARACTERS IN OUTPUT STRING \*/

**sss.str[len1+i] = s2.str[i];**

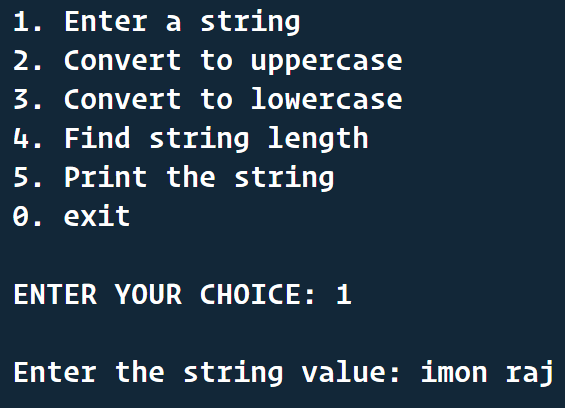
**}**

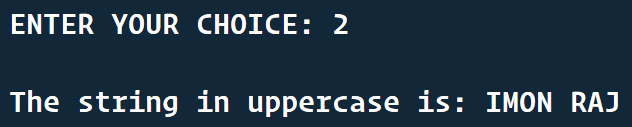
**sss.str[len1+len2] = '\0';**

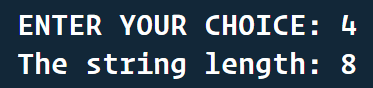
**return sss;**

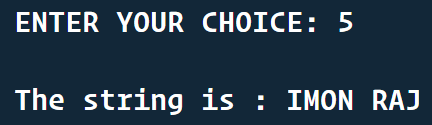
**}**

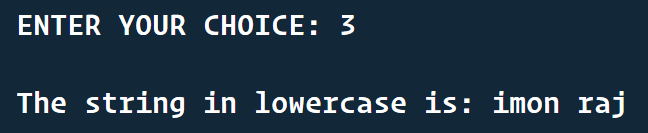
**Results:** In the menu driven program we can Create strings find their length, compare them, concatenate them, copy, find to uppercase and lowercase string of them without bothering about overflow conditions because that is handled in the header file.











**Discussion:** This program helps us to understand how a data type can be created using struct and how different operations can be done on strings without bothering pointer arithmetic array overflow as all those situations are maintained in the header file itself. Here we get the idea of ASCII value arithmetic for converting into uppercase and lowercase. Creating a header file increase modularity in our program as the header file and main C file are separate.

**Problem No. 6**

**Problem Statement**

Given a large single dimensional array of integers, write functions for sliding window filter with maximum, minimum, median, and average to generate an output array. The window size should be an odd integer like 3, 5 or 7. Explain what you will do with the boundary values.

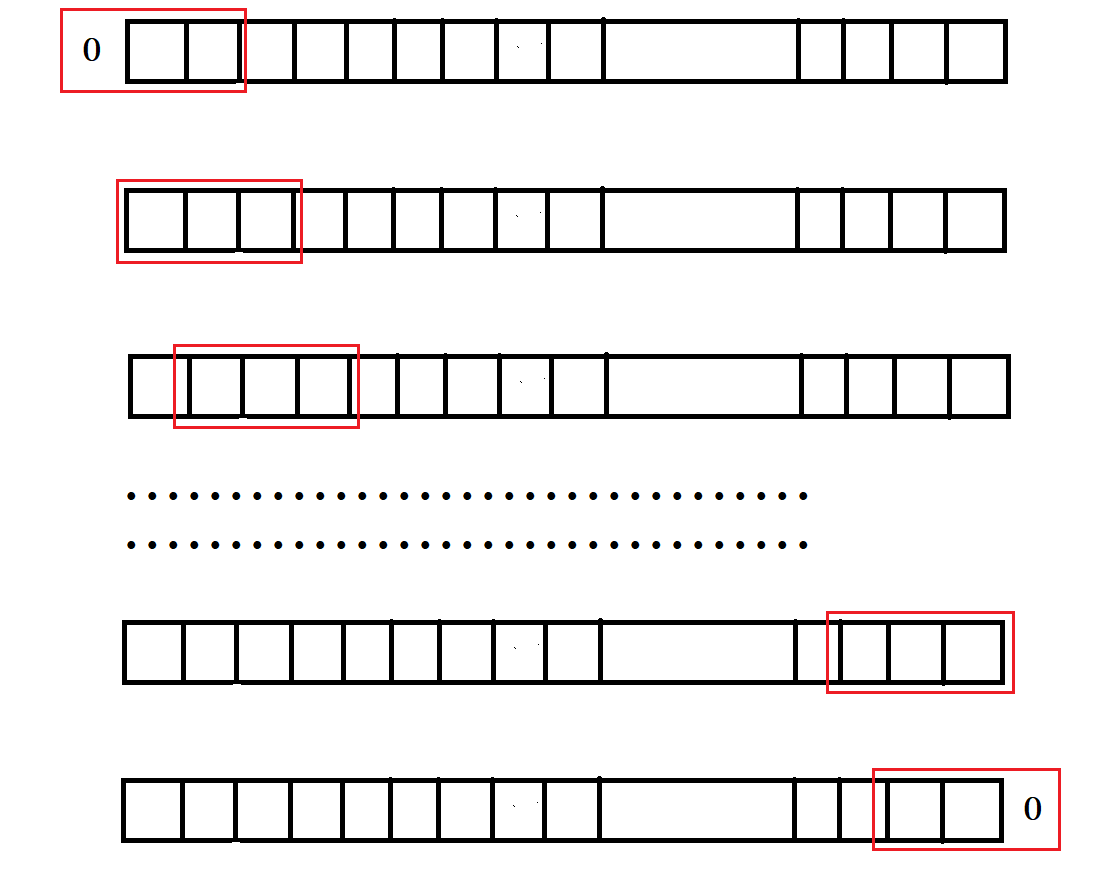
**Solution Approach:**

At the very beginning we will create some functions to find out Maximum, Minimum and Average of three numbers. Now for the array of integers if its length is l, then for finding the sliding window filter we have to traverse the array in such a way that all elements are covered and boundary values are handled well.

For boundary values, at start end I will start the sliding window at one position before of beginning and will assume that empty box value is zero.

This same method will be applicable for last end also.

In each iteration, we will take 3 values from the array and find their avg/max/min and print sequentially.



**Pseudo Code:**

**int max(int a, int b, int c){** /\*FOR FINDING MAX OF THREE NUMS\*/

**int m = (a>=b)?a:b;**

**return (c>=m)?c:m;**

**}**

**int min(int a, int b, int c){** /\*FOR FINDING MIN OF THREE NUMS\*/

**int m = (a<=b)?a:b;**

**return (c<=m)?c:m;**

**}**

**int median(int a, int b, int c)** /\*FOR FINDING MEDIAN OF THREE NUMS\*/

**{**

**if ((a <= b && a >= c) || (a >= b && a <= c))**

**{**

**return a;**

**}**

**else if ((b <= c && b >= a) || (b >= c && b <= a))**

**{**

**return b;**

**}**

**else if ((c <= a && c >= a) || (c >= a && c <= a))**

**{**

**return c;**

**}**

**}**

/\* WE WILL SHOW SLIDING WINDOW FILTER FUNCTION ONCE

HERE . THAT SAME METHOD WILL BE DEFINED IN THE REAL

CODE FOR MAX/ MIN/ MEDIAN \*/

**int \*FUNC\_sliding\_window\_filter(int \*arr, int n)** /\*FUNC will be replaced by max/ min/ median in three functions\*/

**{** /\* THIS FUNCTION RETURNS THE OUTPUT ARRAY \*/

**int \*array\_FUNC\_filtered = (int \*)malloc(n \* sizeof(int));**

**int i;**

**array\_FUNC\_filtered[0] = FUNC(0, arr[0], arr[1]);**

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

**{**

**array\_FUNC\_filtered[i] = FUNC(arr[i - 1], arr[i], arr[i + 1]);**

**}**

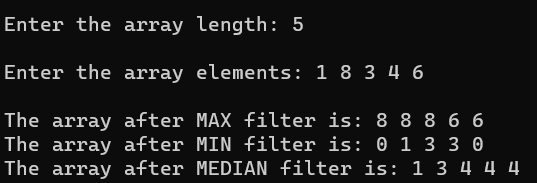
**array\_FUNC\_filtered[n - 1] = FUNC(arr[n - 2], arr[n - 1], 0);**

**return array\_FUNC\_filtered;**

**}**

**Results:**

In this problem, after giving an array as input we got the output on the basis of max/ min/ median of the numbers appearing in the sliding window. That output array is being printed. We have designed this question for 3 width window but we can even design it for 5 ,7 etc.



**Discussion:**  This program helps to understand how to traverse an array in such a way that no out of index garbage value is being accessed. Dividing the question into multiple functions helps to distribute the task to make it simpler.

**Problem No. 7**

**Problem Statement**

Take an arbitrary Matrix of positive integers, say, 128 X 128. Also take integer matrices of size 3 X 3 and 5 X 5. Find out an output matrix of size 128 X 128 by multiplying the small matrix with the corresponding submatrix of the large matrix with the centre of the small matrix placed at the individual positions within the large matrix. Explain how you will handle the boundary values.

**Solution Approach:**

First we created a matrix of the size provided by the user. Then we created an empty matrix to store the product after multiplication. For every small matrix, we considered the corresponding sub-matrix and multiplied them as we multiply two matrices.

**Pseudo Code:**

// Allocate memory dynamically

Function dynamicAllocation

{

int \*\*p;

Allocate memory to p for sizeof(int\*) \* no of rows

For i = 0 to r-1

Allocate memory to p[i] , sizeof(int) \* no of columns

Endfor

return p;

}

// Multiply matrix

int \*\*mul(int \*\*n, int \*\*m, int \*\*p, int B\_MAT\_N, int S\_MAT\_N)

{

int row, col, s\_col, val = 0, r\_index;

int s\_mat\_mid = (S\_MAT\_N - 1) / 2;

For row = 0 to B\_MAT\_N

For col = 0 to B\_MAT\_N

val = 0;

For s\_col = 0 to S\_MAT\_N

r\_index = row - s\_mat\_mid + s\_col;

If not out\_of\_range(r\_index, B\_MAT\_N)

val = val + n[r\_index][col] \* m[s\_mat\_mid][s\_col]

Endif

Endfor

p[row][col] = val;

Endfor

Endfor

return p;

}

// Take user input from user

Function AcceptValues(int \*\*p, int r)

{

int i, j;

For i = 0 to r

For j = 0 to r-1

INPUT from user and set to ith row and jth column of p

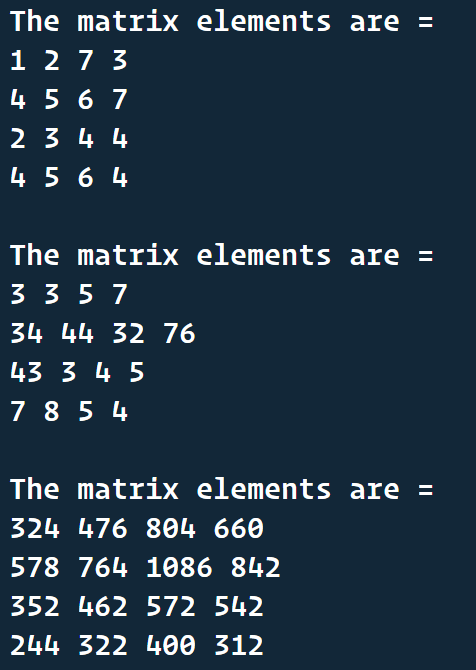
Endfor

Endfor

return p;

}

**Results:**



**Discussion:**

**Problem No. 8**

**Problem Statement**

Find whether an array is sorted or not, and the sorting order.

**Solution Approach:**

For this problem we will maintain two variables – **ascending, descending.** We will initialize both to zero(false). We will then traverse the array, whenever we will find an element which is greater than previous, we will make ascending = 1 and if it is smaller than previous then descending = 1. In each iteration we will ensure that both of ascending and descending are NOT true together**(ascending = descending = 1),** if that is the case then we are sure that the array is not sorted and return. But if we can complete the loop, at last we will check whether –

* ascending = 0, descending = 0 🡺 equal elements
* ascending = 1, descending = 0 🡺 Ascending sorted
* descending = 1, ascending = 0 🡺 Descending sorted

**Pseudo Code:**

//Lets say input to the function **is\_sorted\_array()** is the array to be checked and its length..

**is\_sorted\_array(int arr[], int n){**

**int ascending = 0, descending = 0;**

**int i;**

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

**{**

**if(arr[i-1] < arr[i]){**

**ascending = 1;**

**}else if(arr[i-1] > arr[i]){**

**descending = 1;**

**}**

**if(ascending\*descending){** /\*CHECK IF ASCENDING AND DESCENDING BOTH TRUE AT SAME TIME\*/

**print("ARRAY NOT SORTED");**

**return;**

**}**

**}**

**if(ascending){**

**print("The array is sorted in Ascending order..");**

**}else if(descending){**

**print("The array is sorted in Descending order...");**

**}else{**

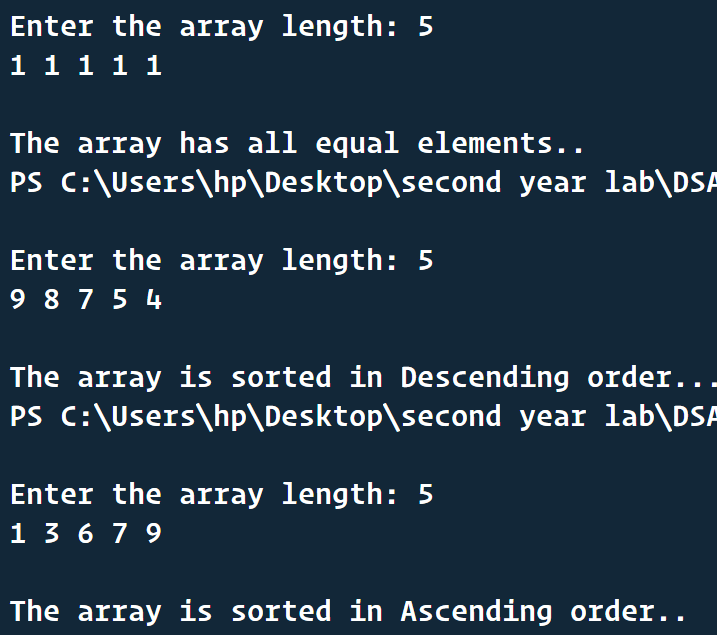
**print("The array has all equal elements..");**

**}**

**}**

**Results:**

For various array inputs, we got different outputs, each case is given below. This program is able to identify if the array is sorted and if yes, in which order and or maybe all elements are equal.



**Discussion:**

This program helps to avoid extra task because here whenever we see that both ascending and descending are true, there is no meaning to completely traverse the array as we are sure array is not sorted. Also as two variables are there, their different combinations of values indicates different scenario.

**Problem No. 9**

**Problem Statement**

Given two sorted arrays, write a function to merge the array in the sorting order.

**Solution Approach:**

At the very beginning we will create two pointers primary and secondary to track the two sorted arrays to be merged. Primary will point to the array whose first element is smaller, and secondary will point to the other array. Then we will traverse both the arrays in such a way that in the output array the elements from both input sources are in a sorted order. When one array is over, then the other array’s remaining elements are inserted into the output array.

**Pseudo Code:**

**int \* merge\_two\_sorted\_arrays(int \* arr1, int m, int \* arr2, int n){**

**int \* merged\_array = (int \*)malloc((m+n)\*sizeof(int));**

**int i = 1, j = 0, k = 1, A, B;**

**int \*primary, \*secondary;**

**if (arr1[0] > arr2[0])**

**{**

**primary = arr2;**

**secondary = arr1;**

**A = n;**

**B = m;**

**}**

**else**

**{**

**primary = arr1;**

**secondary = arr2;**

**A = m;**

**B = n;**

**}**

**merged\_array[0] = primary[0];**

**while (i < A && j < B)**

**{**

**if (secondary[j] < primary[i])**

**{**

**merged\_array[k] = secondary[j];**

**j++;**

**k++;**

**}**

**else**

**{**

**merged\_array[k] = primary[i];**

**i++;**

**k++;**

**}**

**}**

**while (i < A)**

**{**

**merged\_array[k] = primary[i];**

**i++;**

**k++;**

**}**

**while (j < B)**

**{**

**merged\_array[k] = secondary[j];**

**j++;**

**k++;**

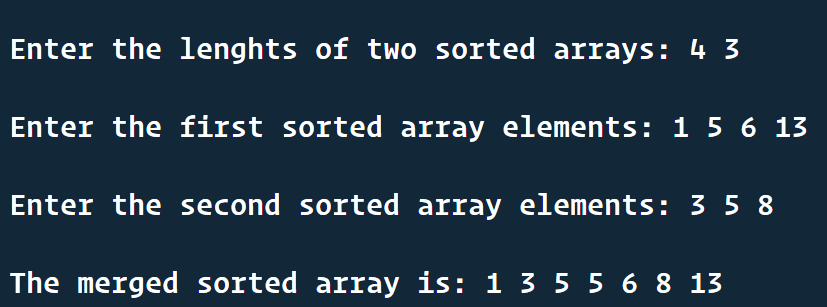
**}**

**return merged\_array;**

**}**

**Results:**

In this question we get an output array which has all the elements of both the sorted array and they are sorted in the output also. This program takes more time when the array sizes are equal because in that case more checking is there.



**Discussion:**

This problem gives us the concept using which we can merge two sorted linked list as well. In that case primary and secondary will be two node pointers and so on.