Q. Insert and element at the end of the array and at a given position, Delete and element from the end of the array and delete an element from a given position in an array.

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

// array io functions

int \*input\_array(int \*size, char \*message);

void print\_array(int \*array, int size);

// insertion functions

void append\_elmnt(int \*array, int \*size, int element);

int insert\_elmnt(int \*array, int \*size, int element, int index);

// deletion functions

int pop\_elmnt(int \*arr, int \*size);

int remove\_elmnt(int \*arr, int \*size, int index);

int main()

{

    int \*array, size = 0, choice, element, position = -1, flag;

    while (1)

    {

        // printing the menu

        printf("\n");

        printf("1.Element Insertion at the end\n2.Element Insertion at an Positon\n3.Pop an Element\n4.Delete an Element by Postion\n5.Exit\n\n");

        printf("Enter your choice:\n--> ");

        scanf("%d", &choice);

        printf("\n");

        switch (choice)

        {

            {

            case 1:

                array = input\_array(&size, "Enter the elements of the array:\n");

                // reading the element to be inserted from the user

                printf("\nEnter the element to insert:\n--> ");

                scanf("%d", &element);

                printf("\n");

                // inserting the element at the end of the array

                append\_elmnt(array, &size, element);

                printf("Modified array is:\n");

                print\_array(array, size);

                break;

            case 2:

                array = input\_array(&size, "Enter the elements of the array:\n");

                // reading the element to be inserted from the user

                printf("\nEnter the element to insert:\n--> ");

                scanf("%d", &element);

                printf("\nEnter the position at which you want to insert the element:\n--> ");

                scanf("%d", &position);

                printf("\n");

                // inserting the element at the given index in the arrya

                flag = insert\_elmnt(array, &size, element, position - 1);

                if(!flag){

                    printf("Modified array is:\n");

                    print\_array(array, size);

                }

                else{

                    printf("Enter a valid position!\n");

                }

                break;

            case 3:

                array = input\_array(&size, "Enter the elements of the array:\n");

                // removing the element from the end

                flag = pop\_elmnt(array, &size);

                if(!flag){

                    printf("Modified array is:\n");

                    print\_array(array, size);

                }

                else{

                    printf("Enter a valid position!\n");

                }

                break;

            case 4:

                array = input\_array(&size, "Enter the elements of the array:\n");

                // reading the element to be inserted from the user

                printf("\nEnter the position of element you want to remove:\n--> ");

                scanf("%d", &position);

                printf("\n");

                // removing the element from the postion

                flag = remove\_elmnt(array, &size, position - 1);

                if(!flag){

                    printf("Modified array is:\n");

                    print\_array(array, size);

                }

                else{

                    printf("Enter a valid position!\n");

                }

                break;

            case 5:

                return 0;

            default:

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

                break;

            }

        }

    }

}

// array io functions

int \*input\_array(int \*size, char \*message)

{

    // initializing variables

    int \*array;

    // reading the size of the array from the user

    printf("Enter the size of the array:\n--> ");

    scanf("%d", size);

    // initializing the array of size 'size'

    array = (int \*)malloc(\*size \* sizeof(int)); // using \*size to access the size(value) of the array

    // reading the elements of the array from the user

    if(\*size > 0) {

        printf("\n%s", message);

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

        {

            printf("array[%d]: ", i);

            scanf("%d", &array[i]);

        }

    }

    return array;

}

void print\_array(int \*array, int size)

{

    printf("[");

    for (int i = 0; i < size; i++)

    {

        printf("%d ", array[i]);

    }

    printf("]\n");

}

// insertion functions

void append\_elmnt(int \*array, int \*size, int element)

{

    array = (int \*)realloc(array, \*size \* sizeof(int) + 1);

    \*size = \*size + 1;

    array[\*size - 1] = element;

}

int insert\_elmnt(int \*array, int \*size, int element, int index)

{

    if (index > \*size)

    {

        return -1;

    }

    // increasing the size of the array

    array = (int \*)realloc(array, \*size \* sizeof(int) + 1);

    \*size = \*size + 1;

    // shifting the elements of the array by one postion(1)

    for (int i = index; i < \*size; i++)

    {

        array[i + 1] = array[i];

    }

    array[index] = element;

    return 0;

}

// deletion functions

int pop\_elmnt(int \*array, int \*size)

{

    if (\*size == 0)

    {

        return -1;

    }

    // decreasing size of the array by one postion

    array = (int \*)realloc(array, \*size \* sizeof(int) - 1);

    \*size = \*size - 1;

    return 0;

}

int remove\_elmnt(int \*array, int \*size, int index)

{

    if (index > \*size)

    {

        return -1;

    }

    // shifting the elements by one postion (-1)

    for (int i = index; i < \*size; i++)

    {

        array[i] = array[i+1];

    }

    // decreasing size of the array by one postion

    array = (int \*)realloc(array, \*size \* sizeof(int) - 1);

    \*size = \*size - 1;

    return 0;

}

Q. Perform Linear Search, Binary Search and Bubble sort on an array.

// Menu driven program to search an element in an array using linear search and binary search and bubble sort

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

// array manipulation functions

int \*input\_array(int \*size, char \*message);

void print\_array(int \*array, int size);

// search functions

int linear\_search(int \*arr, int element, int size);

int binary\_search(int \*arr, int element, int size);

// sorting functions

void bubble\_sort(int \*arr, int size);

int main()

{

    int \*array, size = 0, choice, element, postion = -1;

    while (1)

    {

        // printing the menu

        printf("\n");

        printf("1.Linear Search\n2.Binary Search\n3.Bubble Sort\n4.Exit\n\n");

        printf("Enter your choice:\n--> ");

        scanf("%d", &choice);

        printf("\n");

        switch (choice)

        {

            {

            case 1:

                array = input\_array(&size, "Enter the elements of the array:\n");

                // reading the element to be searched from the user

                printf("\nEnter the element to search:\n--> ");

                scanf("%d", &element);

                printf("\n");

                // searching the element using linear search

                postion = linear\_search(array, element, size);

                printf("Element '%d' is found at postion: %d\n", element, postion + 1);

                break;

            case 2:

                array = input\_array(&size, "Enter the element in sorted manner:\n");

                // reading the element to be searched from the user

                printf("\nEnter the element to search:\n--> ");

                scanf("%d", &element);

                printf("\n");

                // searching the element using binary search

                postion = binary\_search(array, element, size);

                printf("Element '%d' is found at postion: %d\n", element, postion + 1);

                break;

            case 3:

                array = input\_array(&size, "Enter the elements of the array:\n");

                // sorting the array using bubble sort

                bubble\_sort(array, size);

                printf("\nThe sorted array is:\n");

                print\_array(array, size);

                break;

            case 4:

                return 0;

            default:

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

                break;

            }

        }

    }

}

// array manipulation functions

int \*input\_array(int \*size, char \*message)

{

    // initializing variables

    int \*array;

    // reading the size of the array from the user

    printf("Enter the size of the array:\n--> ");

    scanf("%d", size);

    // initializing the array of size 'size'

    array = (int \*)malloc(\*size \* sizeof(int)); // using \*size to access the size(value) of the array

    // reading the elements of the array from the user

    if(\*size > 0) {

        printf("\n%s", message);

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

        {

            printf("array[%d]: ", i);

            scanf("%d", &array[i]);

        }

    }

    return array;

}

void print\_array(int \*array, int size)

{

    printf("[");

    for (int i = 0; i < size; i++)

    {

        printf("%d ", array[i]);

    }

    printf("]\n");

}

// search functions

int linear\_search(int \*array, int element, int size)

{

    for (int i = 0; i < size; i++)

    {

        if (array[i] == element)

        {

            return i;

        }

    }

    return -1;

}

int binary\_search(int \*array, int element, int size)

{

    int lowerBound = 0, upperBound = size - 1, mid;

    for (int i = 0; i < size; i++)

    {

        mid = (lowerBound + upperBound) / 2;

        if (array[mid] == element)

        {

            return mid;

        }

        else if (array[mid] > element)

        {

            upperBound = mid - 1;

        }

        else

        {

            lowerBound = mid + 1;

        }

    }

}

// sorting functions

void bubble\_sort(int \*array, int size)

{

    for (int i = 0; i < size; i++)

    {

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

        {

            if (array[i] < array[j])

            {

                int temp = array[i];

                array[i] = array[j];

                array[j] = temp;

            }

        }

    }

}

Q. Perform String manipulations.

#include<stdio.h>

#include<string.h>

// string manipulation using functions

// string length

int string\_length(char \*string);

// string copy

int string\_copy(char \*string1, char \*string2);

// string concatanation

int string\_concat(char \*string1, char \*string2);

// string compare

int string\_compare(char \*string1, char \*string2);

// print string

int print\_string(char \*string);

int main(int argc, char const \*argv[])

{

    while(1){

        // inititalisation of menu

        int menu = 0;

        printf("\n\nSTRING MANIPULATION\n");

        printf("1. String Length\n");

        printf("2. String Copy\n");

        printf("3. String Concatanation\n");

        printf("4. String Compare\n");

        printf("5. Exit\n\n");

        printf("Choose an operation:\n --> ");

        scanf("%d", &menu);

        // initializing sizes of the strings

        int size1, size2;

        printf("Enter the size of the first string:\n --> ");

        scanf("%d", &size1);

        printf("Enter the size of the second string:\n --> ");

        scanf("%d", &size2);

        // initializing the strings

        char string1[size1], string2[size2];

        printf("Enter the first string:\n --> ");

        scanf("%s", string1);

        printf("Enter the second string:\n --> ");

        scanf("%s", string2);

        // switch case for menu

        switch (menu)

        {

            case 1:

                printf("The length of the first string is %d\n", string\_length(string1));

                printf("The length of the second string is %d\n", string\_length(string2));

                break;

            case 2:

                string\_copy(string1, string2);

                printf("The first string is now %s\n", string1);

                printf("The second string is now %s\n", string2);

                break;

            case 3:

                string\_concat(string1, string2);

                printf("The first string is now %s\n", string1);

                printf("The second string is now %s\n", string2);

                break;

            case 4:

                printf("The first string is %s\n", string1);

                printf("The second string is %s\n", string2);

                printf("The comparison of the two strings is %d\n", string\_compare(string1, string2));

                break;

            case 5:

                return 0;

                break;

        }

    }

    return 0;

}

// string length

int string\_length(char \*string)

{

    int i=0;

    while (string[i]!='\0')

    {

        i++;

    }

    return i;

}

// string copy

int string\_copy(char \*string1, char \*string2)

{

    // compare size of string1 and string2

    int size1= sizeof(string1)/sizeof(char)-1;

    int size2= sizeof(string2)/sizeof(char)-1;

    if(size1>size2){

        printf("Insufficient size");

        return -1;

    }

    int i=0;

    while (string1[i]!='\0')

    {

        string2[i]=string1[i];

        i++;

    }

    string2[i]='\0';

    return 0;

}

// string concatanation

int string\_concat(char \*string1, char \*string2)

{

    // compare size of string1 and string2

    int size1= sizeof(string1)/sizeof(char)-1;

    int size2= sizeof(string2)/sizeof(char)-1;

    if(size1>size2){

        printf("Insufficient size");

        return -1;

    }

    int i=0, j= string\_length(string2);

    while (string1[i]!='\0')

    {

        string2[j]=string1[i];

        i++;

        j++;

    }

    string2[j]='\0';

    return 0;

}

// string compare

int string\_compare(char \*string1, char \*string2){

    int i=0;

    while (string1[i]!='\0' && string2[i]!='\0')

    {

        if (string1[i]>string2[i])

        {

            return 1;

        }

        else if (string1[i]<string2[i])

        {

            return -1;

        }

        i++;

    }

    return 0;

}

Q. Perform operations on Stack Data Structure.

#include<stdio.h>

// stack operations

// push

int push(int \*stack, int top, int maxStack, int item);

// pop

int pop(int \*stack, int top);

// peek

int peek(int \*stack, int top);

// traverse

int traverse(int \*stack, int top);

int main(){

     // initializing the stack

        int maxStack = 0;

        printf("Enter the size of the stack:\n --> ");

        scanf("%d", &maxStack);

        int stack[maxStack];

        int top = -1, item = 0;

    while(1){

        // initializing the menu

        int menu = 0;

        printf("\n\nSTACK OPERATIONS\n");

        printf("1. Push\n");

        printf("2. Pop\n");

        printf("3. Peek\n");

        printf("4. Traverse\n");

        printf("5. Exit\n\n");

        printf("Choose an operation:\n --> ");

        scanf("%d", &menu);

        // switch case for menu

        switch (menu)

        {

            case 1:

                // initializing the item to be pushed

                printf("Enter the item to be pushed:\n --> ");

                scanf("%d", &item);

                top = push(stack, top, maxStack, item);

                break;

            case 2:

                top = pop(stack, top);

                break;

            case 3:

                printf("Top item is: %d:", peek(stack, top));

                break;

            case 4:

                traverse(stack, top);

                break;

            case 5:

                return 0;

            default:

                printf("Invalid choice\n");

                break;

        }

    }

}

// push

int push(int \*stack, int top, int maxStack,int item){

    if(top == maxStack){

        printf("Stack Overflow\n");

        return top;

    }

    else{

        top++;

        stack[top] = item;

        return top;

    }

}

// pop

int pop(int \*stack, int top){

    if(top == -1){

        printf("Stack Underflow\n");

        return top;

    }

    else{

        top--;

        return top;

    }

}

// peek

int peek(int \*stack, int top){

    if(top == -1){

        printf("Stack Underflow\n");

        return top;

    }

    else{

        return stack[top];

    }

}

// traverse

int traverse(int \*stack, int top){

    if(top == -1){

        printf("Stack Underflow\n");

        return top;

    }

    else{

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

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

        }

        return top;

    }

}