S.No: 1	Exp. Name: ***Create an array and perform display, insert at location and delete at location operations.***	**Date: 2024-10-17**

ID: 1NH24DS405-T

Aim:

Implement a menu-driven program in C to perform the following array operations:

1. Creating an Array of N Integer Elements
2. Displaying Array Elements with Suitable Headings
3. Inserting an Element (ELEM) at a Given Valid Position (POS)
4. Deleting an Element at a Given Valid Position (POS)
5. Exiting the Program

Support the program with functions for each of the above operations.

Input Format

- The program begins by asking for the size of the array, *N*.
- The user will input *N* integer elements to populate the array.
- The program will then present a menu with the following choices:
    1. **Insert** :The user inputs the number and position at which to insert it.
    2. **Delete** : The user inputs the position from which to delete an element.
    3. **Display** : The array will be displayed.
    4. **Exit** : The program will terminate.

Output Format

After initializing the array and displaying it, the menu will be displayed.

Based on the user's selection:

6. **Insert:** The updated array is displayed after insertion. If the position entered by the user is not valid, print "**Location not found**"
7. **Delete:** The updated array is displayed after deletion.
8. **Display**: The current elements of the array are shown.
9. **Exit:** The program terminates.

If the user selects any other invalid option, print "**Invalid choice**"

Source Code:

array.c

```
#include<stdio.h>
int main(){
int i,arr[100];
int n = 0,ind,choice,x, isRunning=1;
printf("Enter size: ");
scanf("%d",&n);
printf("Enter elements: ");
for(i=0;i<n;i++){
        scanf("%d",&arr[i]);
}
printf("Array elements: ");
for(int i=0;i<n;i++) printf("%d ",arr[i]);
printf("\n");
printf("1. Insert a number\n");
printf("2. Delete a number\n");
printf("3. Display the array\n");
printf("4. Exit\n");
while(isRunning){
        printf("Enter your choice: ");
                scanf("%d",&choice);
                switch(choice){
                        case 1:printf("Enter number: ");
                                        scanf("%d",&x);
                                        printf("Enter location: ");
                                        scanf("%d",&ind);
                                        if(ind < 0 || ind >n) printf("Location not found ");
                                        else{
                                                                for(int i = n;i > ind;i--){
                                                        arr[i]=arr[i-1];
                                                }
                                                arr[ind] = x;
                                                n++;
                                        }
                                        break;
                        case 2:printf("Enter location: ");
                                scanf("%d",&ind);
                                if(ind<0 || ind>=n) printf("Location not found\n");
                                else{
                                        for(i = ind;i<n-1;i++){
                                        arr[i] = arr[i+1];
                                }
                                n--;
                                }
                        break;
                case 3:printf("Array elements: ");
                                for(int i=0;i<n;i++){
                                        printf("%d ",arr[i]);
                                }
                                printf("\n");
                        break;
                case 4: isRunning=0;
                                        break;
                default: printf("Invalid input!!");
                }
        }
}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
Enter size:
5
Enter elements:
12 15 16 14 18
Array elements: 12 15 16 14 18
1. Insert a number
2. Delete a number
3. Display the array
4. Exit
Enter your choice:
1
Enter number:
2
Enter location:
2
Enter your choice:
2
Enter location:
16
Location not found
Enter your choice:
3
Array elements: 12 15 2 16 14 18
Enter your choice:
4

Test Case - 2
User Output
Enter size:
4
Enter elements:
1 2 3 4
Array elements: 1 2 3 4
1. Insert a number
2. Delete a number
3. Display the array
4. Exit
Enter your choice:
1
Enter number:
5
Enter location:
4

Enter your choice:
2
Enter location:
1
Enter your choice:
3
Array elements: 1 3 4 5
Enter your choice:
4

S.No: 2	Exp. Name: ***Customer account management system***	Date: 2024-10-18

Aim:

Design, develop, and implement a C program that manages customer accounts. You need to create a structure to store the following information for each customer:

10. Name (`string`)
11. Account Number (`integer`)
12. Balance (`floating-point number`)

Implement the following functionalities in your program:

13. **Input Customer Data:** First, prompt the user to enter the number of customers (up to 20). For each customer, input their name, account number, and balance.
14. **Print Customers with Low Balance**: Write a function to print the names of all customers who have a balance less than $200.
15. **Update and Print High Balance Customers**: Write a function to add $100 to the balance of customers with more than $1000 and print their updated balance.

InputFormat

- The program first prompts for the number of customers.
- For each customer, the program will prompt for:
    iv. Name
    iv. Account Number
    iv. Balance

OutputFormat

- The first line of output will print the names of customers with a balance less than $200.
- The next lines of output will then add $100 to the balance of customers who have more than $1000 and print their account number and updated balance which are space-separated.

Source Code:

`manageAccount.c`

ID: 1NH24DS405-T
2023-27-CDS-3A2
New Horizon College

```
#include <stdio.h>
#include <string.h>
#define MAX_CUSTOMERS 20  // Assuming we are dealing with a maximum of 20 customers

// Define the structure for storing customer informatio
typedef struct {
    char name[100];
    int accountNumber;
    float balance;
} Customer;


// Function to print names of all customers with balance less than $200
void printCustomersWithLowBalance(Customer customers[], int count) {
    int i;
        for(i=0;i<count;i++){
                if(customers[i].balance<200){
                        printf("%s\n",customers[i].name);
                }
        }
}

// Function to add $100 to the balance of customers with more than $1000
// and print their updated balance
void updateBalanceForHighBalanceCustomers(Customer customers[], int count) {
   int i;
        for(int i=0;i<count;i++){
                if(customers[i].balance>1000){
                        customers[i].balance +=100;
                        printf("%d %.2f\n",customers[i].accountNumber,
customers[i].balance);
                }
        }
}


int main() {
    int i, numCustomers;
    Customer customers[MAX_CUSTOMERS];
    scanf("%d", &numCustomers);

    // Input customer information
    for (i = 0; i < numCustomers; i++) {
        printf("Enter details for customer %d\n", i + 1);
        scanf("%s",customers[i].name);
                    scanf("%d",&customers[i].accountNumber);
                    scanf("%f",&customers[i].balance);
    }
    printCustomersWithLowBalance(customers, numCustomers);
    updateBalanceForHighBalanceCustomers(customers, numCustomers);
    return 0;
}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
3
Enter details for customer 1
John
101
150
Enter details for customer 2
James
102
350
Enter details for customer 3
Jasmine
103
1200
John
103 1300.00

Test Case - 2
User Output
2
Enter details for customer 1
Kiran
201
10000
Enter details for customer 2
Kashish
202
100
Kashish
201 10100.00

S.No: 3	Exp. Name: ***Stack operations***	Date: 2024-10-18

Aim:

Design, Develop and Implement a menu driven Program in C for the following operations on STACK of Integers (Array Implementation of Stack with maximum size MAX)

19. Push an Element on to Stack
20. Pop an Element from Stack
21. Demonstrate how Stack can be used to check Palindrome
22. Demonstrate Overflow and Underflow situations on Stack
23. Display the status of Stack
24. Exit

Support the program with appropriate functions for each of the above operations

Input Format

25. **Menu Choice:** The user selects an option from the menu by entering a number.
26. **Push Operation:** If the user selects the option to push an element, they will enter an integer value.
27. **Pop Operation:** No additional input is required; the program will pop and display the top element of the stack.
28. **Palindrome Check:** If the user chooses to check for a palindrome, they will enter a string of alphanumeric characters.
29. **Display Status:** The program will display the current status of the stack without requiring additional input.

Output Format

30. **Push Operation:** If successful: No confirmation output is given beyond the prompt. If the stack is full: **Stack overflow**
31. **Pop Operation:** If the stack is not empty: Displays the integer value that was removed from the stack in the format "**Popped value: <element>**" . If the stack is empty: **Stack underflow**
32. **Palindrome Check:** A message indicating whether the provided string is a palindrome or not.
33. **Display Status:** If the stack is not empty: Displays the current elements of the stack from bottom to top separated by spaces. If the stack is empty: **Stack is empty**
34. **Exit:** The program terminates.

Source Code:

stackOperations.c

ID: 1NH24DS405-T
2023-27-CDS-3A2
New Horizon College

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define MAX 100

typedef struct {
    int data[MAX];
    int top;
} Stack;

void initialize(Stack *stack) {
    stack->top=-1;
}

int isEmpty(Stack *stack) {
        if(stack->top==-1){
       return 0;
        }
    return 1;
}

int isFull(Stack *stack) {
        if(stack->top<MAX){
                return 1;
        }
    return 0;
}

void push(Stack *stack, int value) {
        if(isFull(stack)){
                stack->data[++stack->top]=value;
        }
        else{
                printf("Stack overflow\n");
        }
}

int pop(Stack *stack) {
        if(isEmpty(stack)){
                return stack->data[stack->top--];
        }
        else{
                printf("Stack underflow\n");
        }
        return -1;
}

void displayStack(Stack *stack) {
        int i;
        if(isEmpty(stack)){
                printf("Stack elements: ");
                for(i=0;i<=stack->top;i++)
                        printf("%d ",stack->data[i]);
                printf("\n");
```

```
                printf("Stack is empty\n");
        }
}

void checkPalindrome(char *str) {
        int len,i,res=1;
        len=strlen(str);
        for(i=0;i<len;i++){
                if(str[i]!=str[len-i-1]){
                        res=0;
                }
                else{
                        res=1;
                }
        }
        res==1?printf("Palindrome\n"):printf("Not a palindrome\n");
}

int main() {
    Stack stack;
    initialize(&stack);

    int choice, value;
    char str[MAX];

    while (1) {
        printf("Menu:\n");
        printf("1. Push an element onto stack\n");
        printf("2. Pop an element from stack\n");
        printf("3. Check if a string is a palindrome\n");
        printf("4. Display stack status\n");
        printf("5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter an integer to push onto stack: ");
                scanf("%d", &value);
                push(&stack, value);
                break;
            case 2:
                value = pop(&stack);
                if (value != -1) { // Check for underflow
                    printf("Popped value: %d\n", value);
                }
                break;
            case 3:
                printf("Enter a string to check palindrome: ");
                scanf("%s", str);
                checkPalindrome(str);
                break;
            case 4:
                displayStack(&stack);
                break;
```

```
            default:
                printf("Invalid choice\n");
        }
    }

    return 0;
}
```


Execution Results - All test cases have succeeded!

Test Case - 1
User Output
Menu:
1. Push an element onto stack
2. Pop an element from stack
3. Check if a string is a palindrome
4. Display stack status
5. Exit
Enter your choice:
1
Enter an integer to push onto stack:
20
Menu:
1. Push an element onto stack
2. Pop an element from stack
3. Check if a string is a palindrome
4. Display stack status
5. Exit
Enter your choice:
1
Enter an integer to push onto stack:
30
Menu:
1. Push an element onto stack
2. Pop an element from stack
3. Check if a string is a palindrome
4. Display stack status
5. Exit
Enter your choice:
3
Enter a string to check palindrome:
madam
Palindrome
Menu:
1. Push an element onto stack
2. Pop an element from stack
3. Check if a string is a palindrome
4. Display stack status

5. Exit
Enter your choice:
4
Stack elements: 20 30
Menu:
1. Push an element onto stack
2. Pop an element from stack
3. Check if a string is a palindrome
4. Display stack status
5. Exit
Enter your choice:
5

Test Case - 2
User Output
Menu:
1. Push an element onto stack
2. Pop an element from stack
3. Check if a string is a palindrome
4. Display stack status
5. Exit
Enter your choice:
1
Enter an integer to push onto stack:
88
Menu:
1. Push an element onto stack
2. Pop an element from stack
3. Check if a string is a palindrome
4. Display stack status
5. Exit
Enter your choice:
2
Popped value: 88
Menu:
1. Push an element onto stack
2. Pop an element from stack
3. Check if a string is a palindrome
4. Display stack status
5. Exit
Enter your choice:
4
Stack is empty
Menu:
1. Push an element onto stack
2. Pop an element from stack
3. Check if a string is a palindrome
4. Display stack status
5. Exit

Enter your choice:
3
Enter a string to check palindrome:
toy
Not a palindrome
Menu:
1. Push an element onto stack
2. Pop an element from stack
3. Check if a string is a palindrome
4. Display stack status
5. Exit
Enter your choice:
5

S.No: 4	Exp. Name: ***C program to Convert an Infix expression into Postfix expression***	Date: 2024-11-14

ID: 1NH24DS405-T

Aim:

Write the code in the functions `convertInfix(char *e)`, `priority(char x)`, `push(char x)`, `pop()` and `isEmpty` in the below code according to the hints given as comment lines.

```
Sample Output -1
Enter the expression : A+B*(C+D)
Postfix expression : ABCD+*+
Sample Output -2
Enter the expression : A+D*C+E-F&D
Invalid symbols in infix expression. Only alphanumeric and { '+', '-','*', '%', '/' ,
'^'} are allowed.
Sample Output -3
Enter the expression : A+B*C+(D*E
Invalid infix expression : unbalanced parenthesis.
```

Source Code:

Infix2PostfixMain.c

```
#include<stdio.h>
#include "Infix2PostfixOperation.c"

int main() {
        char exp[20];
        char *e, x;
        printf("Enter the expression : ");
        scanf("%s",exp);
        e = exp;
        convertInfix(e);
        return 0;
}
```

Infix2PostfixOperation.c

2023-27-CDS-3A2

```
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#define STACK_MAX_SIZE 20
char stack [STACK_MAX_SIZE];

int top = -1;

//Return 1 if stack is empty else return 0.
int isEmpty(){
        if(top==-1) return 1;
        return 0;
}


//Push the character into stack
void push(char x) {
                stack[++top]=x;
        }


char pop(){
        if(!isEmpty()) return stack[top--];
                else return '\0';
}


// Return 0 if char is '('
// Return 1 if char is '+' or '-'
// Return 2 if char is '*' or '/' or '%' or '^'
int priority(char x) {
        if(x=='^') return 2;
        else if(x=='*'|| x=='/'||x=='%') return 2;
        else if(x=='+'|| x=='-') return 1;
        return 0;
}

//Output Format
//if expression is correct then output will be Postfix Expression : <postfix notation>
//If expression contains invalid operators then output will be "Invalid symbols in infix
expression. Only alphanumberic and { '+', '-','*', '%%', '/' } are allowed."
//If the expression contains unbalanced paranthesis the output will be "Invalid infix
expression : unbalanced parenthesis."
void convertInfix(char * e) {
        int len = strlen(e);
        char res[len+1];
        int j=0;
        for (int i=0;i<len;i++){
                if(isalnum(e[i])) res[j++]=e[i];
                else if(e[i]=='(') push(e[i]);
                else if(e[i]==')'){
                        while(!isEmpty() && stack[top] !='('){
                                res[j++]=pop();
                        }
                        if(!isEmpty()) pop();
```

```
parenthesis.\n");
                                return;
                        }
                }else if (e[i]=='+'||e[i]=='-'||e[i]=='/'||e[i]=='%'||e[i]=='^'||e[i] ==
'*'){
                        while(!isEmpty() && priority(e[i])<=priority(stack[top])) res[j++] =
pop();
                        push (e[i]);
                }else{
                        printf("Invalid symbols in infix expression. Only alphanumeric and {
'+', '-','*', '%%', '/', '^' } are allowed.\n");
                        return ;
                }
        }
        while(!isEmpty()){
                if(stack[top]=='('){
                        printf("Invalid infix expression : unbalanced parenthesis.\n");
                        return;
                }
                res[j++]=pop();
        }
        res[j]='\0';
        printf("Postfix expression : %s\n", res);
}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
Enter the expression :
((a+(b*c))^d)
Postfix expression : abc*+d^

Test Case - 2
User Output
Enter the expression :
((a+(b%c))^d
Invalid infix expression : unbalanced parenthesis.

Test Case - 3
User Output
Enter the expression :
A&B
Invalid symbols in infix expression. Only alphanumeric and { '+', '-','*', '%', '/', '^' } are allowed.

S.No: 5	Exp. Name: ***C program to evaluate a Postfix expression***	Date: 2024-11-12

ID: 1NH24DS405-T
2023-27-CDS-3A2
New Horizon College

Aim:

C program to evaluate a postfix expression.

Write the code in the functions **isEmpty()**, **push(int x)**, **pop()** and **evaluatePostfix(char *e)** in the below program according to hints given as comment lines.

Input Format

- The user will provide a postfix expression as a single string of characters. The expression can contain digits (0-9) and operators (+, -, *, /, %).

Output Format

- If the postfix expression is valid, the program prints the result of the evaluation in the format: **Result : <result>**
- If the postfix expression is invalid (e.g., insufficient operands for the operators or extra operands remaining), the program prints: **Invalid postfix expression.** Carefully observe the print statement and add '.' at end of it

Source Code:

```
PostfixEvaluation.c
```

```
#include <ctype.h>
#include <stdio.h>
#include <stdlib.h>
#define MAX 100
//Declare the required stack variables.
int stack[MAX];
int top=-1;

//Return 1 if stack is empty else return 0.
int isEmpty() {
        return top ==-1;
}

//Push the character into stack
void push(int x) {
        if(top<MAX-1){
                stack[++top]=x;
        }else{
                printf("Stack Overflow\n");
        }
}

//pop a character from stack
int pop() {
        if (!isEmpty()){
                return stack[top--];
        }else{
                printf("Stack Underflow\n");
                return -1;
        }
}

//Output Format - Result : <result> if the input postfix expression is vaild.
//Output Format - Invalid postfix expression,. - if the input expression is invalid.
//postfix expression is given as the parameter.
int evaluatePostfix(char *exp) {
        int i=0;
        int op1,op2;
        while (exp[i] !='\0') {
                if(isdigit(exp[i])){
                        push(exp[i]-'0');
                }
                else{
                        if(top<1){
                                return -1;
                        }
                        op2=pop();
                        op1=pop();
                        switch(exp[i]){
                                case '+':push(op1+op2);break;
                                case '-':push(op1-op2);break;
                                case '*':push(op1*op2);break;
                                case '/':push(op1/op2);break;
                                case '%':push(op1%op2);break;
                                default: return -1;
```

```
                i++;
        }
        if (top!=0){
                return -1;
        }
        return pop();
}

//Read a postfix expression and evaluate it.
int main() {
        char exp[MAX];
        printf("Enter the postfix expression : ");
        scanf("%s",exp);
        int result=evaluatePostfix(exp);
        if(result == -1){
                printf("Invalid postfix expression.\n");
        }else{
                printf("Result : %d\n",result);
        }
        return 0;
}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
Enter the postfix expression :
234+-
Result : -5

Test Case - 2
User Output
Enter the postfix expression :
-456+5+
Invalid postfix expression.

S.No: 6	Exp. Name: ***Tower of Hanoi using Stacks***	Date: 2024-11-15

Aim:

Write a C program to solve the Towers of Hanoi problem using user-defined stacks.

Input Format:

- The user provides an integer n which specifies the number of disks.

Output Format:

- The program uses the Tower of Hanoi algorithm to print the sequence of moves required to solve the puzzle, showing how disks are moved between the source, auxiliary, and destination pegs.

Source Code:

TowerOfHanoiMain.c

```
#include <stdio.h>
#include <stdlib.h>
#include "TowerOfHanoi.c"

int main()
{   int n;
    printf("no of disks: ");
    scanf("%d", &n);

        towerOfHanoi(n);
    return 0;
}
```

TowerOfHanoi.c

ID: 1NH24DS405-T
2023-27-CDS-3A2
New Horizon College

```
// stack node
struct StackNode
{       int disk;
        int source;
        int auxiliary;
        int destination;
    int step;
        struct StackNode *next;
};

// custom stack
struct MyStack
{       struct StackNode *top;
        int size;
};

struct MyStack *newStack()
{       //Make a stack
        struct MyStack *stack = (struct MyStack *) malloc(sizeof(struct MyStack));
        if (stack != NULL)
        {       stack->top = NULL;
                stack->size = 0;
                return stack;
        }
        else{
                printf("Stack overflow\n");
                return stack;
        }
}

int isEmpty(struct MyStack *stack){
        return (stack->size == 0 || stack->top == NULL);
}

// return top element of stack
struct StackNode *peek(struct MyStack *stack){
        if(isEmpty(stack)) return NULL;
        return stack->top;
}

//Create a new node of stack
struct StackNode *newNode(int disk, int source, int destination, int auxiliary,int step){
        struct StackNode *node =(struct StackNode *)malloc(sizeof(struct StackNode));
        node->disk=disk;
        node->source=source;
        node->destination=destination;
        node->auxiliary=auxiliary;
        node->step=step;
        node->next=NULL;
        return node;
}

// add node at the top of stack
void push(struct MyStack *stack, struct StackNode *node){
        node->next=stack->top;
```

```
}

// Remove top element of stack
void pop(struct MyStack *stack){
        if(isEmpty(stack)) return;
        struct StackNode *temp = stack->top;
        stack->top = stack->top->next;
        free(temp);
        stack->size--;
}

// Displaying the movement of current disk
void printResult(struct StackNode *node){
        printf("Move disk %d from tower [%c] to [%c]\n", node->disk, node->source, node-
>destination);
}

// Iterative tower of hanoi implementation
void towerOfHanoi(int total_disk){
        struct MyStack *stack = newStack();
        push(stack, newNode(total_disk, 'A','C','B',0));
        while(!isEmpty(stack)){
                struct StackNode *curr = peek(stack);
                int disk = curr->disk;

                if(disk==1){
                        printResult(curr);
                        pop(stack);
                }else if(curr->step == 0){
                        curr->step++;
                        push(stack,newNode(disk-1, curr->source,curr->auxiliary,curr-
>destination,0));
                }else if(curr->step == 1){
                        printResult(curr);
                        curr->step++;
                }else if(curr->step==2){
                        curr->step++;
                        push(stack, newNode(disk-1,curr->auxiliary, curr->destination, curr-
>source,0));
                }else pop(stack);
        }
        free(stack);
}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
no of disks:
3
Move disk 1 from tower [A] to [C]
Move disk 2 from tower [A] to [B]

Move disk 1 from tower [C] to [B]
Move disk 3 from tower [A] to [C]
Move disk 1 from tower [B] to [A]
Move disk 2 from tower [B] to [C]
Move disk 1 from tower [A] to [C]

Test Case - 2
User Output
no of disks:
2
Move disk 1 from tower [A] to [B]
Move disk 2 from tower [A] to [C]
Move disk 1 from tower [B] to [C]