

Name: Jhaveri Varun Nimitt

UID: 2023800042

Batch: CSE A Batch C

Experiment No.: 1

Aim: Stack application

Problem: **Program to evaluate a Postfix Expression. Show contents of intermediate stack.**

APPROACH NO 1 : GLOBAL ARRAY STACKS:

Program:

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

#define MAX 100

int stack[MAX];
int top = -1;

void push(int val) {
    if (top < MAX - 1) {
        stack[++top] = val;
    }
}

int pop() {
    if (top != -1) {
        return stack[top--];
    }
    return -1;
}

void displayStack() {
    for (int i = 0; i <= top; i++) {
        printf("%d ", stack[i]);
    }
    printf("\n");
}
```

```

int test(char* expr) {
    int i = 0;
    while (expr[i] != '\0') {
        if (isdigit(expr[i])) {
            int num = 0;
            while (isdigit(expr[i])) {
                num = num * 10 + (expr[i] - '0');
                i++;
            }
            push(num);
            displayStack();
        } else if (expr[i] == ' ') {
            i++;
        } else {
            int val2 = pop();
            int val1 = pop();
            switch (expr[i]) {
                case '+': push(val1 + val2); break;
                case '-': push(val1 - val2); break;
                case '*': push(val1 * val2); break;
                case '/': push(val1 / val2); break;
            }
            displayStack();
            i++;
        }
    }
    return pop();
}

int main() {
    char expr[MAX];
    printf("lmao: ");
    fgets(expr, MAX, stdin);
    expr[strcspn(expr, "\n")] = 0;
    printf("%d", test(expr));
    return 0;
}

```

## Observations:

### Program Analysis

#### 1. Main Logic:

- The function parses an RPN expression. It processes the expression character by character:
  - **Digits:** Converts sequences of digits into numbers and pushes them onto the stack. The displayStack function is called after each push.
  - **Operators:** Pops two numbers from the stack, applies the operator (+, -, \*, /), and pushes the result back onto the stack. The displayStack function is called after each operation.
  - **Spaces:** Ignored and used to advance the index.

#### 2. final stuff:

- Reads an RPN expression from standard input, removes the trailing newline character, and calls a test to evaluate the expression. It then prints the result.

## OUTPUT:

```
[cyclops@parrot]-[~]
└─$ cd Desktop/
[cyclops@parrot]-[~/Desktop]
└─$ gcc postfix.c
[cyclops@parrot]-[~/Desktop]
└─$ ./a.out
lmao: 100 200 + 2 / 5 * 7 +
100
100 200
300
300 2
150
150 5
750
750 7
757
757 [cyclops@parrot]-[~/Desktop]
└─$
```

## APPROACH NO 2 : STRUCT STACK:

### Program:

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

#define MAX 100

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

void push(Stack *stack, int val) {
    if (stack->top < MAX - 1) {
        stack->items[++stack->top] = val;
    }
}

int pop(Stack *stack) {
    if (stack->top != -1) {
        return stack->items[stack->top--];
    }
    return -1;
}

void displayStack(Stack *stack) {
    for (int i = 0; i <= stack->top; i++) {
        printf("%d ", stack->items[i]);
    }
    printf("\n");
}

int evaluatePostfix(char* expr) {
    Stack stack;
    stack.top = -1;

    int i = 0;

    while (expr[i] != '\0') {
        if (isdigit(expr[i])) {
            int num = 0;
            while (isdigit(expr[i])) {
                num = num * 10 + (expr[i] - '0');
                i++;
            }
        }
    }
}
```

```

        push(&stack, num);
        displayStack(&stack);
    } else if (expr[i] == ' ') {
        i++;
    } else {
        int val2 = pop(&stack);
        int val1 = pop(&stack);
        switch (expr[i]) {
            case '+': push(&stack, val1 + val2); break;
            case '-': push(&stack, val1 - val2); break;
            case '*': push(&stack, val1 * val2); break;
            case '/': push(&stack, val1 / val2); break;
        }
        displayStack(&stack);
        i++;
    }
}
return pop(&stack);
}

int main() {
    char expr[MAX];
    printf("lmao: ");
    fgets(expr, MAX, stdin);
    expr[strcspn(expr, "\n")] = 0;
    printf("%d\n", evaluatePostfix(expr));
    return 0;
}

```

### **Observations:**

Logic is the same just used struct this time to represent the stack instead of a global array.

```
[cyclops@parrot]-[~]
└─ $cd Desktop/
[cyclops@parrot]-[~/Desktop]
└─ $gcc postfix.c
[cyclops@parrot]-[~/Desktop]
└─ $./a.out
lmao: 100 200 + 2 / 5 * 7 +
100
100 200
300
300 2
150
150 5
750
750 7
757
757 [cyclops@parrot]-[~/Desktop]
└─ $
```

**Output:**

## Question no 2:

Various signal towers are present in a city. Towers are aligned in a straight horizontal line(from left to right) and each tower transmits a signal in the right-to-left direction. Tower A shall block the signal of Tower B if Tower A is present to the left of Tower B and Tower A is taller than Tower B. So, the range of a signal of a given tower can be defined as:

{(the number of contiguous towers just to the left of the given tower whose height is less than or equal to the height of the given tower) + 1}.

You need to find the range of each tower.

### INPUT

First line contains an integer T specifying the number of test cases.

Second line contains an integer n specifying the number of towers.

Third line contains n space separated integers(H[i]) denoting the height of each tower.

### OUTPUT

Print the range of each tower (separated by a space).

### Constraints

1 <= T <= 10

2 <= n <= 10<sup>6</sup>

100 80 60 70 60 75 85

1 1 1 2 1 4 6

## Program:

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

#define MAX 1000000

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

void push(Stack *st, int value) {
    st->items[++st->top] = value;
}

void pop(Stack *st) {
    if (st->top >= 0) {
        st->top--;
    }
}

int top(Stack *st) {
    return st->items[st->top];
}

int isEmpty(Stack *st) {
    return st->top == -1;
}

void calculateRange(int heights[], int n, int R[]) {
    Stack st;
```

```

    st.top = -1;
    push(&st, 0);
    R[0] = 1;

    for (int i = 1; i < n; i++) {
        while (!isEmpty(&st) && heights[top(&st)] <= heights[i]) {
            pop(&st);
        }

        R[i] = isEmpty(&st) ? (i + 1) : (i - top(&st));
        push(&st, i);
    }
}

void printArray(int arr[], int n) {
    for (int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");
}

int main() {
    int t = 1;
    int n = 7;
    int heights[] = {100, 80, 60, 70, 60, 75, 85};
    int *R = (int*)malloc(n * sizeof(int));

    calculateRange(heights, n, R);
    printArray(R, n);

    free(R);

    return 0;
}

```

## **Observations:**

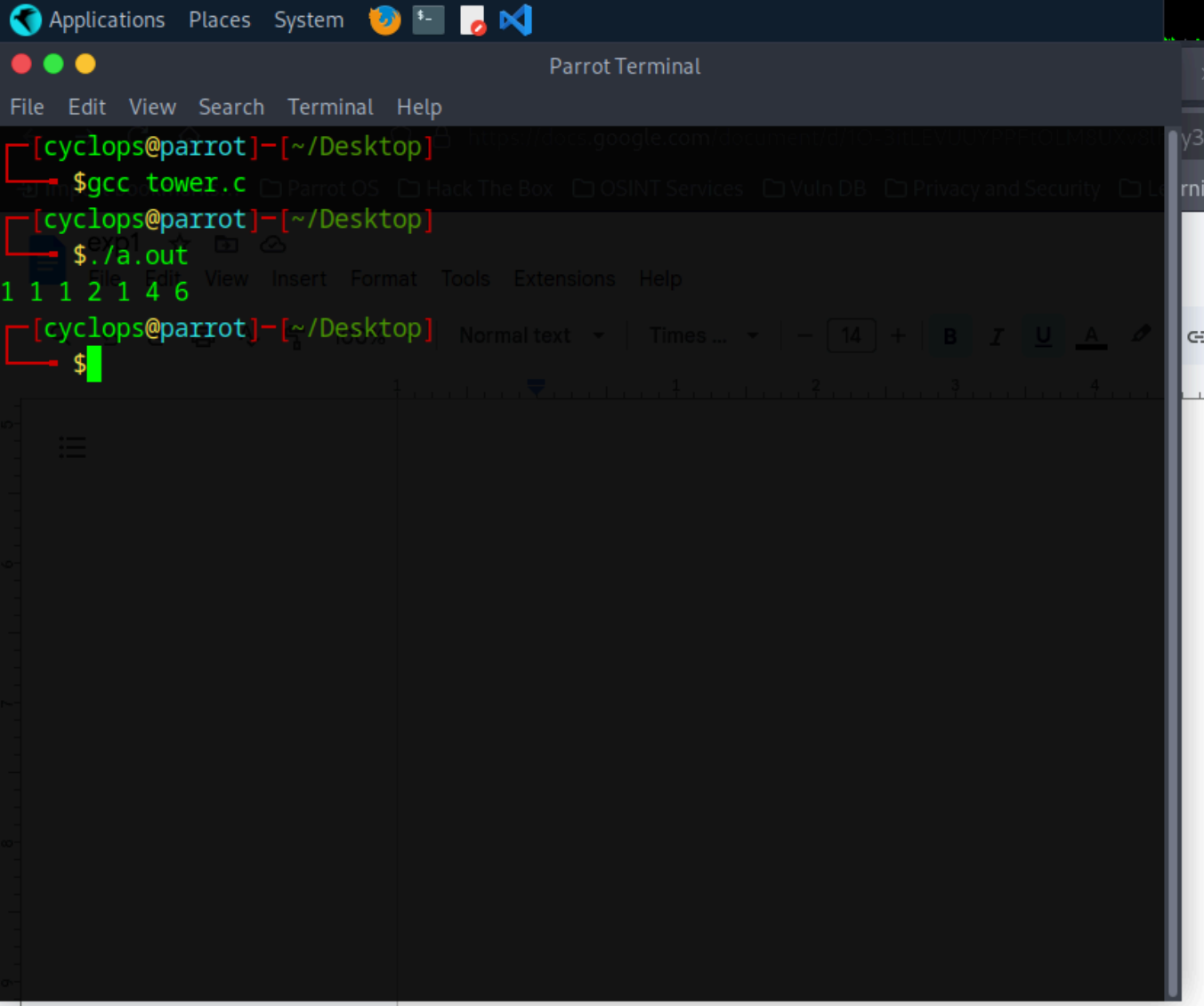
### **Logic**

1. what the stack does:
  - We utilize a stack to keep track of the indices of the towers.
  - The stack helps in efficiently determining the number of contiguous towers to the left that are shorter or equal in height compared to the current tower.
2. logics:
  - We start from the first tower and move to the right, maintaining the stack to store indices of the towers.:



- For each tower, if the stack is not empty and the height of the tower at the index on top of the stack is less than or equal to the current tower's height, pop the stack.
- If the stack becomes empty, it means the current tower can see all the way back to the start, so its range is  $i + 1$ .
- If the stack is not empty after the popping process, the current range is determined by the difference between the current index and the index on the top of the stack.
- After determining the range, push the current index onto the stack.

## OUTPUT:



```
[cyclops@parrot]-[~/Desktop]
$ gcc tower.c
[cyclops@parrot]-[~/Desktop]
$ ./a.out
1 1 1 2 1 4 6
[cyclops@parrot]-[~/Desktop]
$
```