1. **Activity: Reverse a string using stack implemented using dynamic 1D array**

**Algorithm:**

1. Get stringsize from user
2. Create 2 character arrays, string and stack of given input size
3. Push characters from string to stack one by one.
4. Pop characters from stack and store in string
5. Print string

**Test Case Table:**

|  |  |  |
| --- | --- | --- |
| Input | Output | Pass/Fail |
| This is fun | nuf si sihT | Pass |
| Hi how are you | uoy era woh iH | Pass |

**Program:**

#include <stdio.h>

#include <stdlib.h>

int size, top = -1;

void clear\_buff(){

    while ((getchar()) != '\n');

}

void push(char \*stack, char a){

    if (top == size-1) printf("Stack full..");

    else{

        top++;

        stack[top] = a;

    }

}

void display(char \*stack){

    if (top == -1)

    {

        printf("Stack empty.");

        return;

    }

    for (int i = top; i >= 0; i--)

    {

        printf("%c", stack[i]);

    }

}

char pop(char \*stack){

    if (top == -1)

    {

        printf("Stack is empty.");

        return 0;

    }

    char c = stack[top];

    top--;

    return c;

}

int main(){

    printf("Enter size of string: ");

    int n;

    scanf("%d", &n);

    char \*str = (char \*)malloc(n\*sizeof(n));

    printf("Enter string: ");

    clear\_buff();

    scanf("%[^\n]%\*c", str);

    printf("String entered: %s\n", str);

    size = n;

    char stack[n+1];

    for (int i = 0; i < n; i++)

    {

        push(stack,\*(str + i));

    }

    for (int i = 0; i < n; i++)

    {

        \*(str + i) = pop(stack);

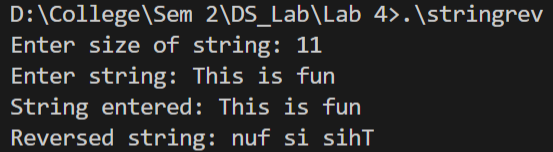
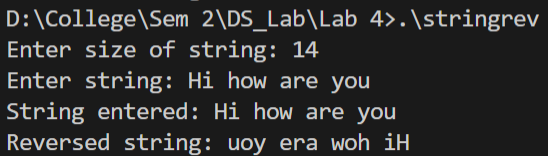
    }

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

return 0;

}

**Screenshot of compilation and execution:**

** \**

1. **Activity:** Convert infix to postfix expression using stack implemented using linked list

**Algorithm:**

1. Get String input
2. Traverse through each character of string
   1. If the character is a number print it directly
   2. If ‘(‘ is encountered, push it to stack
   3. If ‘)’ is encountered, pop all elements in stack till we get first ’(‘ in the stack
   4. If an operator is encountered:
      1. If its precedence is higher than the operator at top of stack push it to stack
      2. If precedence is lower than that of stack top, pop stack until we reach an element that as lower precedence than the encountered operator

**Test Case Table:**

|  |  |  |
| --- | --- | --- |
| Input | Output | Pass/Fail |
| (1/(2-3)\*4+5) | 123-/4\*5+ | Pass |
| (1+2\*3-4)/(5\*6) | 123\*+4-56\*/ | Pass |

**Program:**

#include <stdio.h>

#include <stdlib.h>

struct node{

    char c;

    struct node \*next;

} \*head=NULL, \*ptr=NULL;

//Insert Beginning

void push(char c){

    if (c == '\n')

    {

        return;

    }

    struct node \*temp;

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

    if (temp == NULL)

    {

        printf("Memory full. Not allocated.");

    }

    temp->c = c;

    temp->next = head;

    head = temp;

}

// Delete beginning

char pop(){

    if (head == NULL)

    {

        return ' ';

    }

    if (head->next == NULL)

    {

        char c = head->c;

        free(head);

        head = NULL;

        return c;

    }

    ptr = head->next;

    char c = head->c;

    free(head);

    head = ptr;

    return c;

}

//Priority function

int priority(char c){

    if (c == '(') return 0;

    else if (c == '+' || c == '-') return 1;

    else if (c == '\*' || c == '/') return 2;

    return 0;

}

int main(){

    //Static defined variables for input

    char infix[100], \*p;

    printf("Enter expression: ");

    scanf("%[^\n]s", infix);

    p = infix;

    //Travering through elements in input string

    while(\*p != '\0'){

        //Print directly if number encountered

        if (\*p >= '0' && \*p <= '9')

        {

            printf("%c", \*p);

        }

        //Push ( to stack

        else if(\*p == '('){

            push(\*p);

        }

        //Pop from stack till ( is encountered

        else if(\*p == ')'){

            char x;

            while ((x = pop()) != '(')

            {

                printf("%c", x);

            }

        }

        //Modify stack according to priority rule

        else{

            char stack\_top;

            if (head == NULL);

            else{

                stack\_top = head->c;

                while (priority(stack\_top)>=priority(\*p)){

                    printf("%c", pop());

                    if (head == NULL)

                    {

                        break;

                    }

                    stack\_top = head->c;

                }

            }

            push(\*p);

        }

        //Move to next character

        p++;

    }

    //Empty the stack

    while (head != NULL)

    {

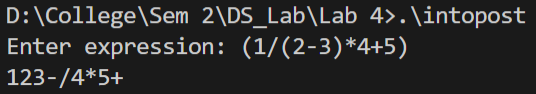
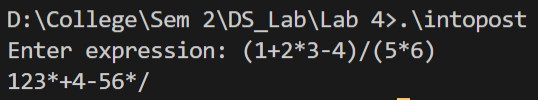
        printf("%c",pop());

    }

    return 0;

}

**Screenshot of compilation and execution:**

** **

1. **Activity:** Evaluate postfix expression using stack implemented with dynamic 1D array.

**Algorithm:**

1. Input statement size and dynamically input statement
2. Traverse through the string:
   1. If character is a space, skip one loop iteration
   2. If character is a number, get full number until ‘ ‘ is reached then push into stack
   3. If character is an operator, pop 2 numbers, do desired operation and then push result into stack
3. Print the final result left in stack

**Test Case Table:**

|  |  |  |
| --- | --- | --- |
| Input | Output | Pass/Fail |
| 100 200 + 2 / 5 \* 7 + | 757 | Pass |
| 5 3 + 8 2 - \* | 48 | Pass |

**Program:**

#include <stdio.h>

#include <stdlib.h>

int size, top = -1;

void push(int \*stack, int c){

    if (top == size-1) printf("Stack full..");

    else{

        top++;

        stack[top] = c;

    }

}

int pop(int \*stack){

    if (top == -1)

    {

        printf("Stack is empty.");

        return 0;

    }

    int c = stack[top];

    top--;

    return c;

}

int main(){

    printf("Note enter expression with spaces.\n");

    printf("Enter statement size: ");

    scanf("%d", &size);

    char \*str, \*p;

    int \*stack;

    str = (char \*)malloc(size\*sizeof(char));

    printf("Enter expression: ");

    while ((getchar())!='\n');

    scanf("%[^\n]s", str);

    p = str;

    stack = (int \*)malloc(size\*sizeof(int));

    p--;

    while (\*p != '\0')

    {

        p++;

        if(\*p == ' ') continue;

//If number encountered

        else if (\*p >= '0' && \*p <= '9')

        {

            int n = 0;

            while (\*p >= '0' && \*p <= '9')

            {

                n = n\*10 + (int)(\*p - '0');

                p++;

            }

            push(stack, n);

        }

//Operator

        else if(\*p == '+' || \*p == '-' || \*p == '/' || \*p == '\*'){

            int n1, n2;

            n1 = pop(stack);

            n2 = pop(stack);

            switch (\*p)

            {

            case '+':

                n1 += n2;

                push(stack, n1);

                break;

            case '-':

                n1 = n2 - n1;

                push(stack, n1);

                break;

            case '\*':

                n1 \*= n2;

                push(stack, n1);

                break;

            case '/':

                n1 = (int)n2/n1;

                push(stack, n1);

                break;

            default:

                break;

            }

        }

    }

    printf("Answer: %d", pop(stack));

    return 0;

}

**Screenshot of compilation and execution:**

