**Exercise 2**

#include <bits/stdc++.h>

using namespace std;

bool isOperand(char c)

{

// If the character is a digit then it must

// be an operand

return isdigit(c);

}

double evaluatePrefix(string exprsn)

{

stack<double> Stack;

for (int j = exprsn.size() - 1; j >= 0; j--) {

// Push operand to Stack

// To convert exprsn[j] to digit subtract

// '0' from exprsn[j].

if (isOperand(exprsn[j]))

Stack.push(exprsn[j] - '0');

else {

// Operator encountered

// Pop two elements from Stack

double o1 = Stack.top();

Stack.pop();

double o2 = Stack.top();

Stack.pop();

// Use switch case to operate on o1

// and o2 and perform o1 O o2.

switch (exprsn[j]) {

case '+':

Stack.push(o1 + o2);

break;

case '-':

Stack.push(o1 - o2);

break;

case '\*':

Stack.push(o1 \* o2);

break;

case '/':

Stack.push(o1 / o2);

break;

}

}

}

return Stack.top();

}

int main()

{

string exprsn = "+9\*26";

cout << evaluatePrefix(exprsn) << endl;

return 0;

}

***Output***

**A picture containing text

Description automatically generated**

**Exercise 6**

#include <bits/stdc++.h>

using namespace std;

// A structure to represent a stack

class Stack

{

public:

int top;

unsigned capacity;

char\* array;

};

// function to create a stack of given

// capacity. It initializes size of stack as 0

Stack\* createStack(unsigned capacity)

{

Stack\* stack = new Stack();

stack->capacity = capacity;

stack->top = -1;

stack->array = new char[(stack->capacity \* sizeof(char))];

return stack;

}

// Stack is full when top is equal to the last index

int isFull(Stack\* stack)

{ return stack->top == stack->capacity - 1; }

// Stack is empty when top is equal to -1

int isEmpty(Stack\* stack)

{ return stack->top == -1; }

// Function to add an item to stack.

// It increases top by 1

void push(Stack\* stack, char item)

{

if (isFull(stack))

return;

stack->array[++stack->top] = item;

}

// Function to remove an item from stack.

// It decreases top by 1

char pop(Stack\* stack)

{

if (isEmpty(stack))

return -1;

return stack->array[stack->top--];

}

// A stack based function to reverse a string

void reverse(char str[])

{

// Create a stack of capacity

//equal to length of string

int n = strlen(str);

Stack\* stack = createStack(n);

// Push all characters of string to stack

int i;

for (i = 0; i < n; i++)

push(stack, str[i]);

// Pop all characters of string and

// put them back to str

for (i = 0; i < n; i++)

str[i] = pop(stack);

}

int main()

{

char str[] = "When Chuck Norris falls in water, Chuck Norris doesn't get wet.";

reverse(str);

cout << "Reversed string is: " << str;

return 0;

}

**Output**

**Graphical user interface, text

Description automatically generated**

**Exercise 7**

***Use Stack to solve a maze***

#include <cstring>

#include <iostream>

#include <stack>

using namespace std;

#define N 4

#define M 5

class node {

public:

int x, y;

int way;

node(int i, int j)

{

x = i;

y = j;

// Initially way

// set to 0

way = 0;

}

};

// maze of n\*m matrix

int n = N, m = M;

// Coordinates of food

int wx, wy;

bool visited[N][M];

bool isReachable(int maze[N][M])

{

// Initially starting at (0, 0).

int i = 0, j = 0;

stack<node> s;

node temp(i, j);

s.push(temp);

while (!s.empty()) {

// Pop the top node and move to the

// left, right, top, down or retract

// back according the value of node's

// way variable.

temp = s.top();

int d = temp.way;

i = temp.x, j = temp.y;

// Increment the way and

// push the node in the stack again.

temp.way++;

s.pop();

s.push(temp);

// If we reach the way

// return true

if (i == wx and j == wy) {

return true;

}

// Checking the Up way.

if (d == 0) {

if (i - 1 >= 0 and maze[i - 1][j] and

visited[i - 1][j]) {

node temp1(i - 1, j);

visited[i - 1][j] = false;

s.push(temp1);

}

}

// Checking the left way

else if (d == 1) {

if (j - 1 >= 0 and maze[i][j - 1] and

visited[i][j - 1]) {

node temp1(i, j - 1);

visited[i][j - 1] = false;

s.push(temp1);

}

}

// Checking the down direction

else if (d == 2) {

if (i + 1 < n and maze[i + 1][j] and

visited[i + 1][j]) {

node temp1(i + 1, j);

visited[i + 1][j] = false;

s.push(temp1);

}

}

// Checking the right way

else if (d == 3) {

if (j + 1 < m and maze[i][j + 1] and

visited[i][j + 1]) {

node temp1(i, j + 1);

visited[i][j + 1] = false;

s.push(temp1);

}

}

// If none of the ways can take

// the person retract back

// to the way where the he or she came from.

else {

visited[temp.x][temp.y] = true;

s.pop();

}

}

// If the stack is empty and

// no exit is found return false.

return false;

}

// Driver code

int main()

{

// Initially setting the visited

// array to true (unvisited)

memset(visited, true, sizeof(visited));

// Maze matrix

int maze[N][M] = {

{ 1, 0, 1, 1, 0 },

{ 1, 1, 1, 0, 1 },

{ 0, 1, 0, 1, 1 },

{ 1, 1, 1, 1, 1 }

};

//ways to get the exit

wx = 2;

wy = 3;

if (isReachable(maze)) {

cout << "You found the exit." << '\n';

}

else

cout << "You are in the wrong way, no exit found." << '\n';

return 0;

}

**Output**

**Graphical user interface, text

Description automatically generated**