**Week 11 : Data Structure and Algorithm Lab**

**Q1. You are given fully paranthesised expression. You have to evaluate the expression and give the result.**

**Code :**

def applyOp(a, b, op):

if op == '+': return a + b

if op == '-': return a - b

if op == '\*': return a \* b

if op == '/': return a // b

string = []

string = input().split(" ")

f1 = string.count("(")

f2 = string.count(")")

s1 = string.count("{")

s2 = string.count("}")

t1 = string.count("[")

t2 = string.count("]")

n = len(string)

if (f1!=f2 or s1 != s2 or t1 != t2):

print ("ERROR")

else :

for i in range (0,n):

if (string[i] == '(' and string[i+4]==')'):

x = applyOp (int(string[i+1]),int(string[i+3]),string[i+2])

string [i] = str(x)

del string [i+1:i+5]

break

for i in range (0,n):

if (string[i] == '{' and string[i+4]=='}'):

x = applyOp (int(string[i+1]),int(string[i+3]),string[i+2])

string [i] = str(x)

del string [i+1:i+5]

break

for i in range (0,n):

if (string[i] == '[' and string[i+4]==']'):

x = applyOp (int(string[i+1]),int(string[i+3]),string[i+2])

print (x)

string [i] = str(x)

del string [i+1:i+5]

break

**Q2.**

**Code :**

class Conversion:

# Constructor to initialize the class variables

def \_\_init\_\_(self, capacity):

self.top = -1

self.capacity = capacity

# This array is used a stack

self.array = []

# Precedence setting

self.output = []

self.precedence = {'+':1, '-':1, '\*':2, '/':2, '^':3}

# check if the stack is empty

def isEmpty(self):

return True if self.top == -1 else False

# Return the value of the top of the stack

def peek(self):

return self.array[-1]

# Pop the element from the stack

def pop(self):

if not self.isEmpty():

self.top -= 1

return self.array.pop()

else:

return "$"

# Push the element to the stack

def push(self, op):

self.top += 1

self.array.append(op)

# A utility function to check is the given character

# is operand

def isOperand(self, ch):

return ch.isalpha()

# Check if the precedence of operator is strictly

# less than top of stack or not

def notGreater(self, i):

try:

a = self.precedence[i]

b = self.precedence[self.peek()]

return True if a <= b else False

except KeyError:

return False

# The main function that converts given infix expression

# to postfix expression

def infixToPostfix(self, exp):

# Iterate over the expression for conversion

for i in exp:

# If the character is an operand,

# add it to output

if self.isOperand(i):

self.output.append(i)

# If the character is an '(', push it to stack

elif i == '(':

self.push(i)

# If the scanned character is an ')', pop and

# output from the stack until and '(' is found

elif i == ')':

while( (not self.isEmpty()) and self.peek() != '('):

a = self.pop()

self.output.append(a)

if (not self.isEmpty() and self.peek() != '('):

return -1

else:

self.pop()

# An operator is encountered

else:

while(not self.isEmpty() and self.notGreater(i)):

self.output.append(self.pop())

self.push(i)

# pop all the operator from the stack

while not self.isEmpty():

self.output.append(self.pop())

print ("".join(self.output))

test = int(input())

for z in range (0,test):

exp = input()

obj = Conversion(len(exp))

obj.infixToPostfix(exp)

**Q3. Balanced Brackets.**

**Code :**

def is\_matched(expression):

s = []

for e in expression:

if (e == '{'):

s.append('}')

elif (e == '['):

s.append(']')

elif (e == '('):

s.append(')')

else:

if (s == [] or e != s[-1]):

return False

s.pop();

return (s ==[])

t = int(input())

for a0 in range(0,t):

expression = input()

if (is\_matched(expression) == True):

print ("YES")

else:

print ("NO")

**Q4. Arithmetic Expressions.**

**Code :**

n = int(input())

a = [int(x) for x in input().strip().split()]

valid = [[False]\*101 for i in range(n)]

valid[0][a[0]] = True

for i in range(1, n):

for v in range(101):

if valid[i-1][v]:

valid[i][(v + a[i]) % 101] = True

valid[i][(v - a[i]) % 101] = True

valid[i][(v \* a[i]) % 101] = True

v = 0

for i in range(n-1,0,-1):

for w in range(101):

if valid[i-1][w]:

if (w + a[i]) % 101 == v:

a[i] = '+' + str(a[i])

v = w

break

if (w - a[i]) % 101 == v:

a[i] = '-' + str(a[i])

v = w

break

if (w \* a[i]) % 101 == v:

a[i] = '\*' + str(a[i])

v = w

break

print(\*a, sep='')

**Q5. Game of Two Stacks.**

**Code :**

#include <cmath>

#include <cstdio>

#include <cstdlib>

#include <climits>

#include <deque>

#include <iostream>

#include <list>

#include <limits>

#include <map>

#include <queue>

#include <set>

#include <stack>

#include <vector>

#define ll long long

#define MIN(a, b) a < b ? a : b

#define MAX(a, b) a > b ? a : b

using namespace std;

int readline(char \*str) {

int i = 0;

char ch;

while((ch = getchar()) != '\n') {

str[i++] = ch;

}

str[i] = '\0';

return i;

}

int arr1[100001];

int arr2[100001];

int foo(int n, int m, ll int x) {

ll int sum = 0;

int i = 0, j = 0;

int count = 0;

while (i < n) {

sum += (ll int) arr1[i];

i++;

if (sum > x) {

i--;

sum -= (ll int) arr1[i];

break;

}

}

count = i;

while (j < m) {

sum += (ll int) arr2[j];

j++;

while (sum > x && i > 0) {

i--;

sum -= (ll int) arr1[i];

}

count = (sum <= x) ? MAX( (count), (i + j) ) : count;

}

return count;

}

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

int t;

scanf("%d", &t);

while( t-- ) {

ll int x;

int n, m;

scanf("%d%d%lld", &n, &m, &x);

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

scanf("%d", &arr1[i]);

}

for (int j = 0 ; j < m; j++) {

scanf("%d", &arr2[j]);

}

printf("%d\n", foo(n, m, x));

}

return 0;

}

**Q6. Project Euler #20 : Factorial Digit Sum**

**Code :**

#include <cmath>

#include <cstdio>

#include <vector>

#include <iostream>

#include <algorithm>

using namespace std;

void mul(vector<int> &v, int x){

int carry = 0;

for(int i=0;i<v.size();i++){

carry += v[i]\*x;

v[i] = carry%10;

carry /= 10;

}

while(carry != 0){

v.push\_back(carry%10);

carry /= 10;

}

}

int main() {

int t; scanf("%d", &t);

while(t--){

int n; scanf("%d", &n);

vector<int> v (3000);

v.push\_back(1);

int sc = 1;

while(sc <= n){

mul(v, sc);

sc++;

}

int sum = 0;

for(int i=0;i<v.size();i++){

sum += v[i];

}

printf("%d\n", sum);

}

return 0;

}

**Q7. Equal Stacks**

**Code :**

#include <bits/stdc++.h>

using namespace std;

int arr1[100005];

int arr2[100005];

int arr3[100005];

void solution() {

int n1, n2, n3, h, sum1=0, sum2=0, sum3=0, d1, d2, d3;

cin >> n1 >> n2 >> n3;

for(int i=0; i<n1; i++) {

cin>>arr1[i];

sum1+=arr1[i];

}

d1 = 0;

for(int i=0; i<n2; i++) {

cin>>arr2[i];

sum2+=arr2[i];

}

d2 = 0;

for(int i=0; i<n3; i++) {

cin>>arr3[i];

sum3+=arr3[i];

}

d3 = 0;

while(1) {

if(d1 == n1 || d2 == n2 || d3 == n3) {

h = 0;

break;

}

if(sum1 == sum2 && sum2 == sum3) {

h = sum1;

break;

}

// If height op pile one is highest

if(sum1 >= sum2 && sum1 >= sum3) {

sum1 -= arr1[d1];

d1++;

}

// If height op pile two is highest

else if(sum2 >= sum1 && sum2>=sum3) {

sum2 -= arr2[d2];

d2++;

}

// If height op pile three is highest

else if(sum3 >= sum2 && sum3 >= sum1) {

sum3 -= arr3[d3];

d3++;

}

}

cout << h << endl;

}

int main () {

solution();

return 0;

}

**Q8. Crossword Puzzle**

**Code :**

#include <bits/stdc++.h>

using namespace std;

vector<string> grid(10);

vector<string> words;

bool f;

void call(int ind)

{

if(!f) {

return;

}

if(ind == words.size()) {

if(f) {

for(auto word: grid) {

cout<<word<<endl;

}

f=false;

}

return;

}

int i,j,p,q,k;

for(i=0;i<10;++i) {

for(j=0;j<10;++j) {

p=i,q=j;

for(k=0;k<words[ind].size() && p+k<10;++k) {

if(grid[p+k][q] != '-' && grid[p+k][q] != words[ind][k]) {

break;

}

}

if(k==words[ind].size()) {

vector<string> temp = grid;

for(k=0;k<words[ind].size();++k) {

grid[p+k][q] = words[ind][k];

}

call(ind+1);

grid = temp;

}

for(k=0;k<words[ind].size() && q+k<10;++k) {

if(grid[p][q+k] != '-' && grid[p][q+k] != words[ind][k]) {

break;

}

}

if(k==words[ind].size()) {

vector<string> temp = grid;

for(k=0;k<words[ind].size();++k) {

grid[p][q+k] = words[ind][k];

}

call(ind+1);

grid = temp;

}

}

}

}

int main()

{

f=true;

int i,j;

for(i=0;i<10;++i) {

cin>>grid[i];

}

string s,w;

cin>>w;

for(auto x: w) {

if(x==';') {

words.push\_back(s);

s="";

} else

s+=x;

}

words.push\_back(s);

call(0);

return 0;

}