As soluções dos exercícios a seguir pressupõem o uso de TADs, principalmente o **TAD Pilha**.
Uma característica do ambiente *Sharif* é que a solução de cada problema esteja em um único arquivo **.c**, para que possa ser submetida (enviada) para ser avaliada no ambiente *Sharif*. Nesse sentido, ao trabalhar com TADs, antes de submeter uma solução, crie um único arquivo **.c**, cujo conteúdo segue o formato abaixo:

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

<TADs: conteúdo do(s) arquivo(s) .h>

<TADs: conteúdo do(s) arquivo(s) .c , sem #include "TAD.h">

int main () {
        <seu código>
}
```

O "uso do TAD" significa que a função **main** deve utilizar somente as operações presentes no arquivo **TAD.h**, ou seja, conhecer que operações estão previstas para o TAD.
Considere o TAD Pilha, conforme as operações abaixo:

```
typedef struct pilha Pilha;


Pilha * pilha_cria (int maxTamanho);
Pilha * pilha_copia (Pilha * p);
void pilha_libera (Pilha * p);
int pilha_insere (Pilha * p, char * elemento); // push
char * pilha_remove (Pilha * p); // pop
char * pilha_obtem_topo  (Pilha * p); // top
int pilha_obtem_tamanho  (Pilha * p);
int pilha_se_vazia (Pilha * p);
int pilha_se_cheia (Pilha * p);
char * pilha_imprime  (Pilha * p);
```

Problema 01) Expressões matemáticas
Considere que expressões matemáticas (cadeia de caracteres) são compostas por operandos (ex. 12 , -123 e 44) e operadores (ex. '+' , '/' e '*'), bem como caracteres para a definição de subexpressões (ex. '[' , '(' , '{' , ']' , ') e '}').
Uma cadeia de caracteres é dita bem definida (válida) se atende às seguintes propriedades:
1. Ela é uma cadeia de caracteres vazia (não contém nenhum caractere);
2. Ela é formada por uma cadeia bem definida envolvida por parênteses, colchetes ou chaves. Portanto, se a cadeia S é bem definida, então as cadeias '(S)', '[S]' e '{S}' também são bem definidas;
3. Ela é formada pela concatenação de duas cadeias bem definidas. Logo, se as cadeias X e Y são bem definidas, a cadeia XY é bem definida.

Escreva um programa que determina, especificamente, se uma expressão é bem definida com respeito ao uso de parênteses, colchetes e chaves. Aplique o TAD Pilha e implemente suas operações usando estrutura do tipo vetor (é preciso concluir a implementação de **Pilha_vetor.h**)

Entrada: uma única linha com uma expressão matemática, seguida por um espaço em branco e a string "fim". Os termos da expressão matemática são separados por um espaço em branco entre si.
Saída: uma única linha contendo "sim" se a expressão é bem definida com respeito ao uso de parênteses, colchetes e chaves, ou "nao" caso contrário.
Exemplos:

Entrada	Saída
(11 + { 2 * -33]) fim	**nao**
(11 + { 2 * -33 }) fim	**sim**
[11 + { 2 * -33 } [fim	**nao**
[11 + { 2 * -33 } fim	**nao**

Problema 02) Expressões matemáticas 2
Considere a Notação Polonesa Inversa (pós-fixada) proposta por Charles Hamblin em 1950, que utiliza operador após operandos. Por exemplo, se uma expressão na Notação Infixa é (1-2)*(4+5), então a mesma expressão na Notação Polonesa Inversa será 1 2 – 4 5 + *. Construa um programa que recebe uma expressão na Notação Infixa e a transforma na Notação Polonesa Inversa.

Breves dicas:

- expressões entre parênteses devem ser convertidas de tal forma que possam ser tratadas como um único operando;
- operadores são empilhados; operandos não são empilhados;
- abre parênteses é sempre empilhado;
- fecha parênteses nunca é empilhado; então todos os operadores são desempilhados até encontrar um '(', '[' ou '{';
- operadores desempilhados são colocados na expressão na forma pós-fixa.

Entrada: uma única linha com uma expressão na Notação Infixa, cujos *tokens* (elementos da expressão) estão separados por um espaço em branco, seguido por um espaço em branco e o *token* "fim".
Saída: a versão em Notação Polonesa Inversa da expressão de entrada.

Importante 1: os operadores são + , - , / e *, e não há precedência entre operadores.
Importante 2: é garantido que a expressão de entrada é válida.
Exemplos:

Entrada	Saída
`( 2 + 4 ) / ( 3 – 1 ) * 4 fim`	`2 4 + 3 1 – / 4 *`
`( 1 + ( 4 * 2 ) / 3 – 1 ) * 4 fim`	`1 4 2 * + 3 / 1 – 4 *`
`[ ( { 1 / 2 } – 3 ) * 5 ] – 1 fim`	`1 2 / 3 – 5 * 1 –`
`5 + 3 * 2 fim`	`5 3 + 2 *`

Problema 03) Expressões matemáticas 3
Repita o Exercício 02 (Expressões matemáticas 2), contudo usando a implementação da pilha por encadeamento simples. Noutras palavras, aplique o TAD Pilha e implemente suas operações usando estrutura do tipo encadeamento simples (é preciso concluir a implementação de **Pilha_encadeamento_simples.h**)

Problema 04) Expressões matemáticas 4
Repita o Exercício 03 (Expressões matemáticas 3), contudo considerando a precedência de operadores.
Por exemplo:
1) sem precedência: a entrada "3 + 5 * 2" é convertida em 3 5 + 2 *, com resultado 16;
2) com precedência: a entrada "3 + 5 * 2" é convertida em 3 5 2 * +, com resultado 13.

Aplique o TAD Pilha e implemente suas operações usando estrutura do tipo encadeamento simples (é preciso concluir a implementação de **Pilha_encadeamento_simples.h**)