INSTITUTO FEDERAL FLUMINENSE - IFF
Campus Itaperuna

Prof.: Leandro Fernandes dos Santos | ✉ leandro.f.santos@iff.edu.br
Disciplina: Estruturas de Dados

Atividade Avaliativa 2 - Módulo 2

Turma: Sistemas de Informação 3º Período

Objetivo: Esta atividade tem como objetivo consolidar o conceito de pilha implementada de maneira estática e suas principais operações.

Observações e regras de entrega:

- Deverá ser realizada de forma individual.
- Terá o valor de 4 pontos na composição da nota na A2.
- A solução deverá ser enviada por meio da plataforma run.codes até as 23h59min do dia 12/03/2023.
- Pontos importantes a serem seguidos:
  - **Conformidade com os requisitos** aqui apresentados.
  - Uso dos conceitos de modularização.
  - Códigos bem estruturados e bem indentados.
- Caso seja detectada a ocorrência de plágio, será atribuída nota zero a **todos os envolvidos**.
- Submissões que apresentarem **erros de compilação** ou **falhas de segmentação** no run.codes **NÃO** serão corrigidas.
- O não atendimento dos **requisitos especificados neste documento** acarretará em perda de pontos.

Especificação

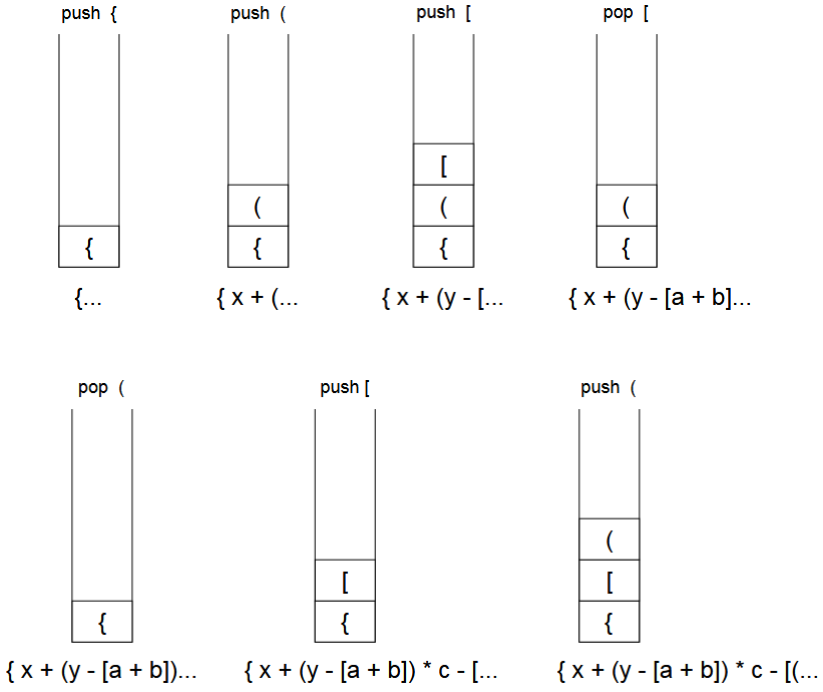
Este exercício trata-se da implementação de uma pilha para a solução de um problema que consiste basicamente na análise da validade de uma expressão quanto ao aninhamento de 3 delimitadores de escopo: parênteses ("()"), chaves ("{}") ou colchetes ("[]"). Desta forma, um finalizador de escopo deve ser do mesmo tipo de seu iniciador. Sendo assim expressões do tipo:

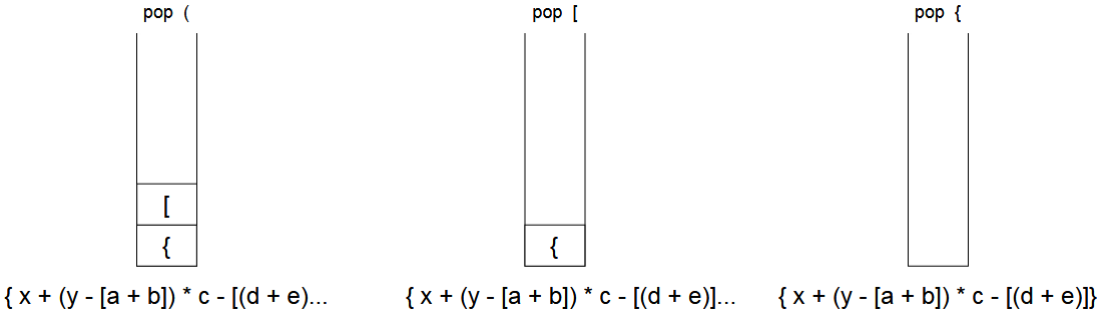
$(A + B]$, $[(A + B])$, $\{A - (B]\}$ são inválidas.

Assim torna-se necessário rastrear os escopos abertos e seus respectivos tipos. Estas informações são importantes pois toda vez que um finalizador de escopo for encontrado, então precisamos conhecer o símbolo com o qual o escopo foi aberto para assegurar que ele seja corretamente fechado.

Um pilha pode ser utilizada para rastrear os tipos de escopos encontrados. Sempre que um iniciador de escopo "(", "{"ou "["for encontrado, ele será empilhado (push). Sempre que um finalizador de escopo for encontrado, então a pilha será analisada. Se a pilha estiver vazia, então o finalizador de escopo não terá um inicializador correspondente e a string será, consequentemente inválida. De outro modo, se a pilha não estiver vazia, desempilharemos (pop) e verificaremos se o item desempilhado corresponde ao inicializador de escopo relacionado ao finalizador encontrado. Se ocorrer uma coincidência, então o processo de análise continua, caso contrário a string será inválida. Quando o final da string for alcançado, a pilha deverá estar vazia; caso contrário, existirá um ou mais escopos abertos sem terem sido fechados e a string também será inválida.

Abaixo seguem algumas imagens sobre o comportamento da pilha durante a análise da expressão:
$\{x + (y - [a + b]) * c - [(d + e)]\}$





Em seguida encontra-se o pseudo-algoritmo referente a função expression_analyze, responsável pela análise das expressões.

Algoritmo 1: expression_analyze(s, expr)

Entrada: Pilha vazia (s) e Vetor com expressão (expr) a ser analisada
Saída: **1** caso `expr` seja válida, **0** caso contrário

```
1  i ← 0;
2  valid ← 1;
3  symbol ← expr[0];
4  popped ←'\0';
5  //enquanto o final da string não for atingido
6  enquanto (symbol <> '\0') faça
7      se (symbol == '(' || symbol == '[' || symbol == '{') então
8          push(s, symbol) ;                                   // empilha
9      se (symbol == ')' || symbol == ']' || symbol == '}') então
10         se (empty(s)) então
11             valid ← 0;
12         senão
13             popped ← pop(s) ;                              // desempilha
14             se (popped não corresponde ao iniciador de symbol) então
15                 valid ← 0;
16     i ← i + 1;
17     symbol ← expr[i];
18 se (not empty(s)) então
19     valid ← 0;
20 retorna valid;
```

A solução deste problema requer o uso de uma pilha pois o último escopo a ser aberto dever ser o primeiro a ser fechado. Isto é simulado por uma pilha na qual o último elemento a entrar é o primeiro a sair (**L**ast **I**n **F**irst **O**ut - LIFO).

Para a realização deste trabalho você deverá construir um programa em C chamado expression_analyzer que conterá uma pilha internamente para analisar expressões passadas para a função expression_analyze acima descrita . Os requisitos a serem seguidos são:

- Sua pilha deverá ser implementada usando **listas encadeadas** como visto em aula.
- Deverá existir um tipo estruturado (`struct`) chamado Node contendo os seguintes campos:
  - `symbol`: Variável do tipo caractere que armazenará um símbolo delimitador de escopo
  - `next`: Ponteiro para o próximo elemento da pilha
- Deverá também existir um tipo estruturado com o nome `Stack` contendo o seguinte campo:
  - `top`: Ponteiro para estruturas do tipo `Node` referenciando elementos da pilha
- A operação de empilhar deverá ser implementada usando o seguinte protótipo `void push(Stack *s, char e)`, onde `s` representa a pilha e `e` o elemento a ser empilhado.
- A operação de desempilhar deverá ser implementada usando o seguinte protótipo `char pop(Stack *s)`, onde `s` representa a pilha. A função deverá retornar o elemento que foi desempilhado.
- Deverá existir uma operação para verificação de pilha vazia. Esta função deverá seguir o protótipo `short int isEmpty(stack *s)`, onde `s` representa a pilha. Sua função deverá retornar **1** caso a pilha esteja vazia e **0** caso contrário. Lembre-se que neste contexto, uma pilha vazia significa `top = NULL`.
- A função expression_analyze do Algoritmo 1 deverá possuir o seguinte protótipo:

  `short int expression_analyze(Stack *s, char *expr)`, onde `s` representa a pilha e `expr` a expressão a ser analisada. Esta função deverá retornar **1** caso a expressão seja válida e **0** caso contrário.
- Deverão existir funções para esvaziar a pilha e liberá-la como vistas em aula, pois em caso de expressão inválida, ainda podem restar elementos na pilha.
- Quando o programa for executado, este deverá solicitar a expressão ao usuário por meio de um simples `scanf` e deverá responder com a mensagem "Valida"em caso de validade da expressão, ou "Invalida"caso contrário. Abaixo seguem alguns exemplos de como o seu programa deverá se comportar.

```
./expression_analyzer
(a + b) + c * (d + e)
Valida
```

```
./expression_analyzer

[{x-3*(z+y)}^2+{57-q*(a+b)}]^3
Valida

./expression_analyzer
{([a*b] + c)
Invalida

./expression_analyzer
([a*b] + c)}
Invalida

./expression_analyzer
1
Valida
```

Comece a tentar resolver o exercício o quanto antes, enquanto os assuntos tratados estão frescos na memória e o prazo para terminá-lo está tão longe quanto jamais poderá estar.

Bom Trabalho!