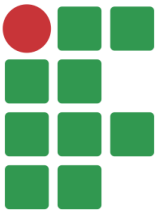

INSTITUTO FEDERAL DA PARAÍBA
CAMPUS CAMPINA GRANDE
BACHARELADO EM ENGENHARIA DA COMPUTAÇÃO
DISCIPLINA DE LABORATÓRIO DE ESTRUTURAS DE DADOS
PROF. VICTOR ANDRÉ PINHO DE OLIVEIRA

Lab. Estruturas de Dados

Atividade Prática 7 - Pilhas

Instruções

Responda às questões abaixo, desenvolvendo cada uma em um novo arquivo .cpp ou .c. Temos 3 questões, sendo que cada questão vale 2 pontos. Temos também duas questões bônus que oferecem pontos EXTRAS. A Bônus 1 vale 2 pontos e a Bônus 2, 3 pontos.

Questões

1. Para cada uma das versões de Pilha implementada no material crie uma função chamada getTopo que retorna o elemento do topo sem removê-lo da Pilha.
2. A forma como implementamos nossas Pilhas no material traz uma grande limitação: o nosso programa só pode manipular uma Pilha por vez, e isso não é bom. Podemos resolver isso criando um registro PILHA, que contém as variáveis particulares necessárias para o controle de cada Pilha. Desse modo, basta adicionarmos um novo parâmetro às funções para que elas operem em cima da Pilha passada como argumento. A partir do código abaixo, implemente uma versão melhorada de uma Pilha (baseada em Lista Sequencial).

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

struct sPILHA{
  int pos, *arr;
  int MAX;
};

typedef struct sPILHA PILHA;

void criar(PILHA *pi, int tam_MAX);
void apagar(PILHA *pi);

void push(PILHA *pi, int dado);
```

```
int pop(PILHA *pi);

int tamanho(PILHA *pi);
void imprimir(PILHA *pi);
```

Note que, da forma como criamos o registro, cada registro PILHA terá campos para o controle individual da Pilha. Veja um exemplo de como criar e usar a PILHA:

```
int main(){
  PILHA pilha1;
  criar(&pilha1, 10);

  push(&pilha1, 100);
  push(&pilha1, 50);
  push(&pilha1, 200);

  printf("Pop em Pilha 1 -> %d\n",pop(&pilha1));

  imprimir(&pilha1);

  apagar(&pilha1);

  return 0;
}
```

Considere:

- a função **criar** recebe, além da Pilha, o tamanho máximo. Esse tamanho deverá ser usado para alocar memória para o campo arr.
- a função **apagar**, por sua vez, deverá desalocar o array arr. Não esqueça de ajustar os demais campos do registro PILHA.
- as demais funções farão a mesma coisa conforme visto no material. Desta vez, no entanto, considerando o parâmetro PILHA *pi, que espera receber o ponteiro para a Pilha a ser manipulada.

Em essência, o código está praticamente pronto no material, à exceção da função criar. Você fará apenas as adequações necessárias para atender às novas especificações.

3. Que tal uma exercício prático que aplique o conceito de Pilha para resolvê-lo? Pois bem, usando Pilhas, crie um programa que receba uma entrada em formato de string (de tamanho máximo 100) e diga se os parênteses, colchetes e chaves presentes na string estão balanceados. Eles estarão balanceados quando, para cada parêntese abrindo, por exemplo, existir um parêntese fechando em qualquer parte da string. Idem para colchetes e chaves. A string poderá conter qualquer caractere ASCII. A saída do programa deverá ser “Balanceada” ou “Não Balanceada”.

BÔNUS 1 Escreva um programa que, usando uma Pilha, inverte as letras de cada palavra de uma string lida do usuário preservando a ordem das palavras. Por exemplo, dado o texto:

ESTRUTURA DE DADOS EH BOM DEMAIS

a saída deve ser:

ARUTURTSE ED SODAD HE MOB SIAMED

BÔNUS 2 Escreva um programa que utilize uma Pilha para verificar se uma string lida do usuário contém uma expressão com a parentização correta. O programa deve verificar se cada "abre parênteses" tem um "fecha parênteses" correspondente. Os operadores presentes serão sempre +, -, * e /, e os operandos serão sempre constantes de um dígito (1,2,3,4,5,6,7,8,9 ou 0). A string lida não contém espaços.

Desse modo, isto está correto: (2+3)-(9/9)

E isto também: ((2+3)-(9/9))

Ou isto: ((2+3-(9/9)))

Mas isto não:)2+3(-(9/9)

Nem isto: (2+3-(9/9)

O programa deve exibir como saída "OK" em caso de parentização correta e "Não OK" em caso de parentização incorreta.