Trabalho 06 - Verificador de Códigos

Data de entrega: 12/12/2021

Importante:

- **Não** olhe códigos de outros ou da internet, exceto o que é fornecido. Também não mostre ou publique o seu.
- Em caso de plágio, fraude ou tentativa de burlar o sistema será aplicado nota 0 na disciplina aos envolvidos.
- Alguns alunos podem ser solicitados para explicar com detalhes a implementação.
- Passar em todos os testes do run.codes não é garantia de tirar a nota máxima. Sua nota ainda depende do cumprimento das especificações do trabalho, qualidade do código, clareza dos comentários, boas práticas de programação e entendimento da matéria demonstrada em possível reunião.
- Você deverá submeter, até a data de entrega, o seu código na plataforma run.codes.

Neste trabalho você deverá implementar um verificador de parentização utilizando a TAD Pilha que **você** implementou no Trabalho 05.

Uma expressão matemática precisa estar corretamente parentizada, ou seja, a estrutura hierárquica de parênteses precisa ser respeitada. Por exemplo, a expressão ((x+2) * 2), está corretamente parentizada, enquanto (x+2(* 2) não está. O mesmo ocorre quando utilizamos múltiplos símbolos para realizar essa separação, [x + 2 * { y / (3 * z) }].

Com códigos em linguagens de programação acontece o mesmo. Por exemplo, o código:

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

int main(int argc, char * argv[]){
  printf("Ola Mundo\n");
}
```

Está balanceado em relação aos parênteses, colchetes e chaves que ele abre e fecha. Enquanto no código abaixo, provavelmente por erro do programador, há um parênteses que abre na linha 3 e nunca é fechado.

```
#include <stdio.h>
#include <stdlib.h>
(
int main(int argc, char * argv[]){
  printf("Ola Mundo\n");
}
```

Podemos utilizar uma pilha para verificar se a parentização está balanceada. Utilizando a seguinte ideia:

- Sempre que abrimos um símbolo ‘(‘, ‘[‘ ou ‘{‘, colocamos ele em uma pilha.
- Sempre que fechamos um símbolo ‘)’, ‘]’ ou ‘}’, retiramos o elemento do topo da pilha e verificamos se é o símbolo de abertura correspondente. Se não for, significa que detectamos um desbalanceamento.
- Ao final do código/expressão verificamos se a pilha está vazia. Se não estiver significa que abrimos um símbolo que nunca fechamos.
- Todos os outros tipos de caracteres são ignorados.

Nesse trabalho você deverá implementar um verificador de parentização simples. Que simplesmente lê o texto da entrada padrão do sistema e verifica se nesse texto, parentes, colchetes e chaves são abertos e fechados de maneira balanceada. Um código será fornecido e você pode usá-lo à vontade.

O verificador não vai verificar todos os casos. Por exemplo, se um símbolo aparece dentro de uma string, ele vai ser considerado um símbolo e acusará um erro. Por exemplo, a linha abaixo irá acusar um erro, embora ela esteja correta. Não iremos tratar desses casos.

```
printf(“Olá :]”);
```

Obviamente o nosso verificador também não garante que um código/expressão está corretamente escrito, mas pode sim ser usado como uma das etapas da verificação completa.

O seu programa deverá indicar qual o símbolo que originou o problema. Se o símbolo é de fechamento, ele deve dizer a linha que houve esse fechamento. E se for um símbolo de abertura que não foi fechado, o programa deve indicar o símbolo, mas não precisa indicar em qual linha ele foi aberto. No exemplo A:

```
#include <stdio.h>
#include <stdlib.h>
(
int main(int argc, char * argv[]){
  printf("Ola Mundo\n");
}
```

A saída do seu programa será:

```
Sobrou um '('
```

Já no exemplo B:

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

void imprime_mundo(){
  printf("Ola Mundo\n");
}

int main(int argc, char * argv[]){
  imprime_mundo();
}}}}
```

A saída do seu programa será:

```
Problema com '}' na linha 10
```

Se nenhum problema for detectado seu programa imprimirá apenas:

```
tudo ok
```

Você deverá entregar um .zip com 3 arquivos:

- main.c - Com a função main que lê os dados de entrada, funções e estruturas auxiliares.
- pilha.h - Uma interface para as funções de pilha. (trabalho 05)
- pilha.c - Com a implementação das funções e estrutura de pilhas. (trabalho 05)

Você não precisa colocar o Makefile no zip.