Atividade Prática 02
"Simulador de Compilador"

Universidade Tecnológica Federal do Paraná (UTFPR), campus Dois Vizinhos
Curso de Engenharia de Computação
Disciplina de Estruturas de Dados - ED23S
Prof. Dr. Rafael Gomes Mantovani

Instruções:

- Leia todas as instruções corretamente para poder desenvolver sua atividade/programa;
- Evite plágio (será verificado por meio de ferramentas automatizadas). Faça seu programa com os seus nomes de variáveis e lógica de solução. Plágios identificados anularão as atividades entregues de todos os envolvidos.
- Adicione comentários nos códigos explicando seu raciocínio e sua tomada de decisão. Porém, não exagere nos comentários, pois a própria estrutura do programa deve ser auto-explicativa.
- Salve sua atividade em um arquivo único, com todas as funções e procedimentos desenvolvidos. É esse **arquivo único** que deverá ser enviado ao professor.

1 Descrição da atividade

Um compilador é um programa de computador que, a partir de um código fonte, cria um programa semanticamente equivalente, porém escrito em outra linguagem, o código objeto. Classicamente, um compilador traduz um programa de uma linguagem textual facilmente entendida por um ser humano para uma linguagem de máquina, específica para um processador e sistema operacional.

Existem algumas etapas no processo de compilação: na análise léxica o compilador verifica se escrevemos algum símbolo (caractere) que não pertence à linguagem; na análise sintática verifica-se se a estrutura do programa segue a sintaxe da linguagem; e na análise semântica são checadas se os comandos podem ser executados. No final destas análises é gerado um código intermediário, mais próximo da linguagem objeto e que ainda permite uma manipulação mais fácil do que o código de máquina.

Um tipo popular de linguagem intermediária é conhecido como **código de três endereços**. Neste tipo de código uma sentença típica tem a forma `X := A op B`, onde `X`, `A` e `B` são operandos e `op` uma operação qualquer. Uma forma prática de representar sentenças de três

endereços é através do uso de quádruplas (operador, argumento 1, argumento 2 e, resultado). Este esquema de representação de código intermediário é preferido por diversos compiladores, principalmente aqueles que executam extensivas otimizações de código, uma vez que o código intermediário pode ser rearranjado de uma maneira conveniente com facilidade.

2 Objetivo

Sendo, assim, sua tarefa é implementar um programa que receba expressões alfa-numéricas no padrão de notação pós-fixada, valide essas expressões e gere o correspondente código intermediário em um arquivo de saída. O vocabulário permitido nas expressões de entrada é:

- Operadores: {+, -, *, /}
- Operandos: letras maiúsculas [A ... Z]

ou seja, seu programa, que simulará um compilador deverá validar expressões compostas apenas por esses caracteres. Além disso, as expressões estarão descritas na Notação Polonesa (reversa), também conhecida como **Notação Pós-fixa**[1]. O atrativo da Notação Pós-fixa é que ela dispensa o uso de parênteses. Por exemplo, as expressões

a*b+c
a*(b+c)
(a+b)*c
(a+b)*(c+d)

seriam representadas nesse tipo de notação respectivamente como:

a b * c +
a b c + *
a b + c *
a b + c d + *

Posteriormente, essa notação se mostrou apropriada para a representação dessas expressões em linguagens de programação. Essa notação foi, também, popularizada por ter sido adotada em diversas calculadoras científicas, notadamente as calculadoras da marca HP. Conforme o programa for lendo as expressões e realizando os cálculos, ele deve gerar instruções de código intermediário no arquivo de saída. Seis operações podem ser geradas:

- `LDR <OP>`: carregar/ler o valor de um operando `<OP>`;
- `STR <OP>`: grava/salva o valor de um operando `<OP>`;
- `ADD <OP1> <OP2>`: somar os valores de dois operandos;
- `SUB <OP1> <OP2>`: subtrair os valores de dois operandos;

[1] (Mais informações consultar os links no final deste documento)

- `MUL <OP1> <OP2>`: multiplicar os valores de dois operandos;
- `DIV <OP1> <OP2>`: dividir os valores de dois operandos.

Se necessário você pode criar variáveis temporárias que irão armazenar resultados parciais das operações. Por exemplo, depois de somados os valores de `A` e `B` você pode considerar o resultado como a variável `TEMP1`. No seu simulador, explore os valores de variáveis temporárias de `TEMP1` até `TEMP9`, se necessário. A estrutura usada para resolver o problema deve estar vazia no final da execução do programa, o que caracteriza uma expressão válida. Caso existam valores dentro dela, isso indica um erro na expressão de entrada, e portanto erros devem ser tratados pela lógica do programa.

3 Entradas e saídas do programa

O programa receberá dois arquivos texto como parâmetros de entrada:

- **arquivo de entrada:** um arquivo texto simples com uma única linha contendo uma expressão aritmética na notação pós-fixa (arquivos de `entrada01.txt` e `entrada02.txt` na Figura 1). É seu dever verificar se os símbolos contidos na expressão fazem parte do vocabulário válido do compilador;
- **arquivo de saída:** é o arquivo onde serão impressos os comandos na linguagem intermediária. Neste arquivo você deve imprimir todas as operações necessárias para que o computador consiga calcular o resultado da expressão com base no conjunto de operações descrito na seção anterior. Os arquivos `saida01.txt` e `saida02.txt` na Figura 1 mostram as correspondentes saídas gerada para as entradas acima.

4 Orientações gerais

Além da funcionalidade desejada, implementar também o controle de erros, para lidar com exceções que possam ocorrer, como por exemplo:

- problemas nas aberturas dos arquivos de entrada e saída;
- arquivo de entrada vazio (sem informação);
- arquivo de entrada fora do padrão esperado;
- etc.

Opcionalmente, para acompanhamento do desenvolvimento, pode-se criar um repositório individual no `github`.

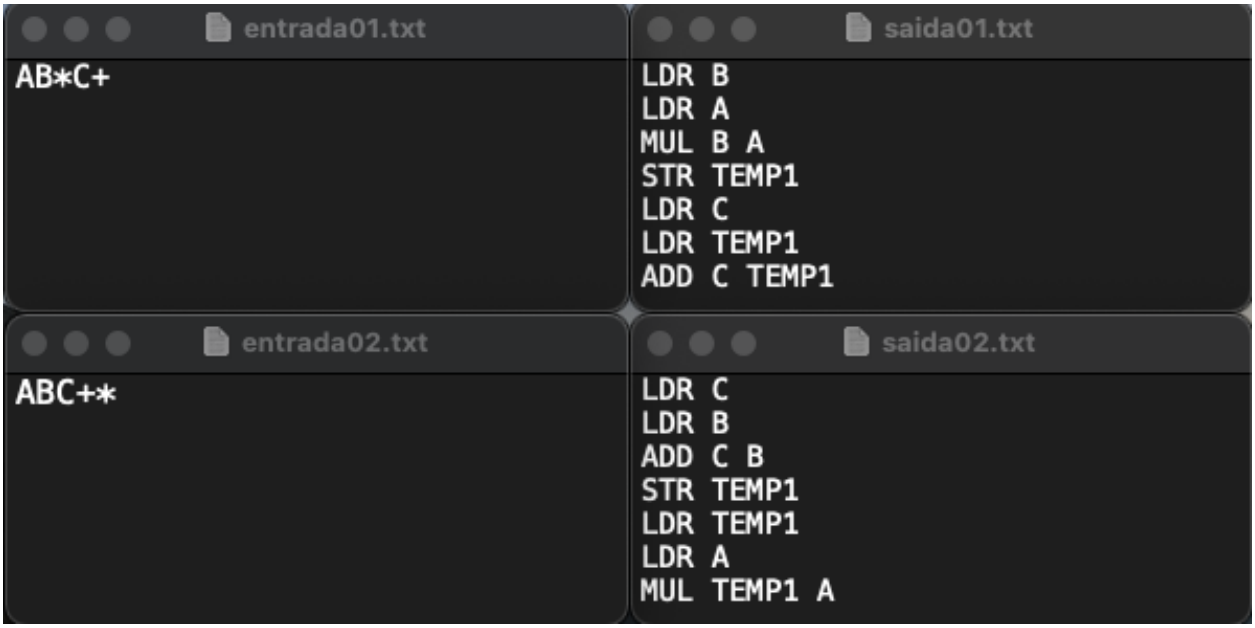

Figura 1: Valores de entrada e correspondentes arquivos de saída gerado pelo programa.

4.1 Critério de correção

A nota na atividade será contabilizada levando-se em consideração alguns critérios:

1. pontualidade na entrega;
2. não existir plágio;
3. completude da implementação (tudo foi feito);
4. o código compila e executa;
5. uso de `argc` e `argv` para controle dos arquivos de teste;
6. implementar o parser para entrada dos dados via arquivo texto;
7. implementação das estruturas necessárias;
8. legibilidade do código (identação, comentários nos blocos mais críticos);
9. implementação dos controles de erros (arquivos de entrada inválidos, e erros no programa principal);
10. controle de memória: chamar o destrutor e desalocar a memória de tudo se usar a estrutura dinâmica, fechar os arquivos, etc;
11. executar corretamente os casos de teste.

Em cada um desses critérios, haverá uma nota intermediária valorada por meio de conceitos:

- **Sim** - se a implementação entregue cumprir o que se esperava daquele critério;
- **Parcial** - se satisfizer parcialmente o tópico;
- e **Não** se o critério não foi atendido.

4.2 Dados para envio da atividade

Ao elaborar seu programa, crie um único arquivo fonte (.c) seguindo o padrão de nome especificado:

```
ED1-<ANO>-<SEMESTRE>-AT02-Compilador-<NOME>.c
```

Exemplo:

```
ED1-2022-1-AT02-Compilador-RafaelMantovani.c
```

A entrega da atividade será via Moodle: o link será disponibilizado na página da disciplina.

5 Links úteis

Arquivos em C:

- https://www.inf.pucrs.br/~pinho/LaproI/Arquivos/Arquivos.htm
- https://www.geeksforgeeks.org/basics-file-handling-c/
- https://www.programiz.com/c-programming/c-file-input-output

Argumentos de Linha de comando (argc e argv):

- https://www.tutorialspoint.com/cprogramming/c_command_line_arguments.htm
- http://linguagemc.com.br/argumentos-em-linha-de-comando/
- http://www.univasf.edu.br/~marcelo.linder/arquivos_pc/aulas/aula19.pdf
- http://www.inf.ufpr.br/cursos/ci067/Docs/NotasAula/notas-31_Argumentos_linha_comando.html
- http://www.dca.fee.unicamp.br/cursos/EA876/apostila/HTML/node145.html

Notação pós-fixa:

- https://www.dca.fee.unicamp.br/cursos/EA876/apostila/HTML/node75.html
- https://pt.wikipedia.org/wiki/Notaç~ao_polonesa_inversa
- https://panda.ime.usp.br/algoritmos/static/eps/ep6/parted/parted-polonesa.html

Referências

[1] Thomas H. Cormen,; Ronald Rivest; Charles E. Leiserson; Clifford Stein. Algoritmos - Teoria e Prática - 3ª Ed. Elsevier - Campus, 2012.

[2] Nivio Ziviani. Projeto de algoritmos com implementações: em Pascal e C. Pioneira, 1999.

[3] Adam Drozdek. Estrutura De Dados E Algoritmos Em C++. Cengage, 2010.