INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA TRIÂNGULO MINEIRO	Curso: Bacharelado em Ciência da Computação
	Unidade Curricular: Compiladores Ano/Período: 2019 / 8
	Tipo de Atividade: Projeto Interpretador Portugol
	Professor: Getúlio de Morais Pereira

O objetivo deste projeto é a implementação de um Interpretador para a pseudo-linguagem Portugol[1].

Este interpretador será usado, via linha de comando, para interpretar um arquivo contendo o pseudo-código de um algoritmo, conforme ilustrado nos exemplos 1 e 2 abaixo:

ex. 1: algoritmo (em arquivo nomeado "HelloWorld.por")

Listing 1: Exemplo de algoritmo em pseudo-código

```
1 algoritmo "ola mundo"
2 inicio
3   escreval("Ola mundo!");
4 fimalgoritmo
```

ex. 2: interpretação do algoritmo, via linha de comando

$> interpretar helloworld.por

Olá mundo!

$>

Tendo em vista que a especificação da pseudo-linguagem Portugol não tem uma versão formalmente definida na literatura, especificamente para este projeto, será adotada a versão utilizada pela ferramenta didática VisuAlg (ver referências).

As seções a seguir descrevem as etapas que subdividem este projeto, sendo cada uma objeto de avaliação.

[1] Também conhecida na literatura como Português Estruturado

1 Analisador Léxico

Deadline:	18/10/2019 23:59:59
Pontuação:	15,0

Tabela 1: Calendário para entrega do *scanner*

O objetivo desta etapa é a criação de um **módulo** responsável pela análise léxica utilizada do interpretador.

1.1 Instruções

- deve ser implementado um *scanner*[2], responsável pelo reconhecimento dos lexemas da pseudo-linguagem
- deve ser utilizada a ferramenta **Flex**[3] e a linguagem C-ANSI.
- os materiais apontados na seção de Referências devem ser consultados para identificar os tokens e lexemas válidos para a pseudo-linguagem.
- caracteres não significativos deverão ser ignorados pelo *scanner*, tais como:
  - espaços
  - tabs
  - "enters"(ou seja, quebras de linha)
  - comentários de linha (identificados por //)
  - comentários de bloco (delimitados por /* e */, ou por { e }))
- qualquer outro símbolo estranho ao *scanner* deverá ser considerado desconhecido, e uma mensagem de erro apropriada deve ser emitida.
  - elabore um dicionário de dados, para documentar os códigos/mensagens de erro utilizados no projeto.

1.2 Avaliação

A avaliação desse módulo ocorrerá por meio de compilação dos códigos-fontes e, em seguida, execução do programa gerado sobre os arquivos de exemplo, contendo cada um, um algoritmo, da seguinte maneira:

$> scanner entrada.por

onde:

- **scanner**: nome do programa constituído pelo analisador léxico
- **entrada.por**: arquivo, com a extensão **.por**, que conterá um algoritmo escrito na pseudo-linguagem Portugol e que será analisado pelo *scanner*.

[2]Sinônimo para "analisador léxico"
[3]Disponível em: http://www.gnu.org/software/flex/

Exemplo:

- arquivo de entrada:

Listing 2: Exemplo de algoritmo em pseudo-código

```
1 algoritmo "ola mundo"
2 inicio
3   escreval("Ola mundo!");
4 fimalgoritmo
```

- execução do analisador léxico:

$> scanner olamundo.por

PALAVRARESERVADA algoritmo

IDENTIFICADOR "ola mundo"

PALAVRARESERVADA inicio

IDENTIFICADOR escreval

SIMBOLOESPECIAL (

LITERAL "Olá Mundo!"

SIMBOLOESPECIAL)

SIMBOLOESPECIAL ;

PALAVRARESERVADA fimalgoritmo

$>

1.3 Considerações importantes

1. **Obrigatoriamente**, deverá ser criado um diretório nomeado "exemplos"contendo 5 algoritmos escritos corretamente em Portugol (sugestão de nomes: algoritmo1.por, algoritmo2.por, algoritmo3.por, algoritmo4.por e algoritmo5.por).

2. os exemplos devem conter as seguintes instruções:

   - Declaração de variáveis
   - Comandos de seleção (tais como: SE-ENTAO, SE-ENTAO-SENAO, ESCOLHA-CASO)
   - Comandos de repetição (tais como: PARA-FACA, ENQUANTO, REPITA-ATE)
   - Declaração de funções
   - Chamadas de funções (pré-definidas e declaradas)
   - Atribuições
   - Expressões aritméticas
   - Expressões lógicas
   - Expressões relacionais
   - **dica**: use alguns dos algoritmos implementados na disciplina "*Algoritmos e Lógica de Programação*".

3. o módulo do analisador léxico deve ser compilado via make. Portanto, o arquivo Makefile devidamente configurado também deve estar disponibilizado na entrega.

4. um arquivo de documentação do módulo também deve estar disponível (pode ser em **.docx**)

5. o executável gerado na compilação deste módulo deve ter o nome **scanner.exe**

1.4 Entrega

A entrega do módulo deve ser realizada via *release* no repositório do git, até a data definida no quadro 1.

O conteúdo da entrega deve contemplar **todos** os códigos-fontes necessários para a compilação (sem erros) do programa a ser avaliado, assim como os algoritmos de exemplo.

2 Analisador Sintático

Deadline:	01/11/2019 23:59:59
Pontuação:	20,0

Tabela 2: Calendário de entrega do parser

O objetivo desta etapa é a criação de um analisador sintático que será incorporado, como um módulo, ao interpretador.

2.1 Instruções

- deve ser implementado um *parser*[4], responsável pelo reconhecimento das instruções da pseudo-linguagem
- deve ser utilizada a ferramenta **Bison**[5] e a linguagem C-ANSI.
- o módulo do *scanner*, desenvolvido na etapa anterior, deve ser corretamente incorporado a este módulo (consulte as atividades práticas executadas em sala de aula).
- os materiais apontados na seção de Referências devem ser consultados para identificar as regras gramaticais válidas para a pseudo-linguagem. Também considere pesquisas sobre a gramática da linguagem Pascal, como referência adicional.
- atualize o dicionário de dados, para documentar os códigos/mensagens de erro adicionados por este módulo.
- a tabela de símbolos deve ser implementada/atualizada nesse módulo

2.2 Avaliação

A avaliação desse módulo ocorrerá por meio de compilação dos códigos-fontes e, em seguida, execução do programa gerado sobre os arquivos de exemplo, contendo cada um, um algoritmo, da seguinte maneira:

$> parser entrada.por

onde:

- **parser**: nome do programa constituído pelo analisador sintático
- **entrada.por**: arquivo, com a extensão **.por**, que conterá um algoritmo escrito na pseudo-linguagem Portugol e que será analisado pelo parser.

Exemplo:

- arquivo de entrada:

Listing 3: Exemplo de algoritmo em pseudo-código

```
1 algoritmo "ola mundo"
2 inicio
3   escreval("Ola mundo!");
4 fimalgoritmo
```

- execução do analisador léxico:

  $> parser olamundo.por

  algoritmo reconhecido com sucesso

  $>

[4] Sinônimo para "analisador sintático"

[5] Disponível em: http://www.gnu.org/software/bison/bison.html

2.3 Considerações importantes

- na ocorrência de um erro sintático, o programa deve exibir uma mensagem informativa indicando a linha e a coluna de sua ocorrência.
- não devem ocorrer erros do tipo shift-reduce nem do tipo reduce-reduce
- o executável gerado na compilação deste módulo deve ter o nome **parser.exe**

2.4 Entrega

A entrega do módulo deve ser realizada via *release* no repositório do git, até a data definida no quadro 2.

O conteúdo da entrega deve contemplar **todos** os códigos-fontes necessários para a compilação (sem erros) do programa a ser avaliado, assim como os algoritmos de exemplo.

3 Analisador Semântico

Deadline:	15/11/2019 23:59:59
Pontuação:	20,0

Tabela 3: Calendário de entrega do analisador semântico

O objetivo desta etapa é a criação de um analisador semântico que será incorporado, como um módulo, ao interpretador.

3.1 Instruções

- os módulos *scanner* e *parser*, desenvolvidos nas etapas anteriores, devem ser corretamente incorporados a este módulo.
- deve-se implementar, adequadamente, as estruturas de dados utilizadas para montar a árvore semântica do algoritmo a ser interpretado (consulte as referências para mais detalhes).
- o módulo deve detectar erros semânticos, emitido as mensagens apropriadas, assim como a localização de sua ocorrência (se for o caso). Mais detalhes no item seguinte.
- o analisador semântico deve verificar:
  - se as variáveis utilizadas foram declaradas
    - nesse caso, apresente a mensagem:
      Erro semântico na linha **n**. Variável não declarada.
      (Onde **n** é o número da linha).
  - se há variáveis declaradas mais de uma vez
    - nesse caso, apresente a mensagem:
      Erro semântico na linha **n**. Variável redeclarada.
      (Onde **n** é o número da linha).
  - se há variáveis declaradas e não foram utilizadas no algoritmo
    - nesse caso, apresente a mensagem:
      Aviso: variável **xpto** declarada e não utilizada.
      (Onde **xpto** é o nome da variável).
  - se os tipos associados às variáveis e ao valor associado a elas são compatíveis
    - no caso de incompatibilidade, apresente a mensagem:
      Erro semântico na linha **n**. Tipo inválido associado à variável **xpto**.
      (Onde **n** é o numero da linha e **xpto** é o nome da variável).
  - se o número de argumentos utilizados na chamada de uma função ou de um procedimento está correto
    - em caso incorreto, apresente a mensagem:
      Erro semântico na linha **n**. Número de argumentos não condizem com a declaração da função/procedimento **xpto**.
      (Onde **n** é o numero da linha e **xpto** é o nome da função/procedimento).

  - se os tipos dos argumentos passados na chamada de uma função ou de um procedimento estão corretos
    * em caso incorreto, apresente a mensagem:
      Erro semântico na linha **n**. Tipo inválido associado a parâmetro da função/procedimento **xpto**.
      (Onde **n** é o numero da linha e **xpto** é o nome da função/procedimento.
  - se o tipo associado ao valor de retorno de uma função está correto
    * em caso incorreto, apresente a mensagem:
      Erro semântico na linha **n**. Tipo inválido associado ao retorno da função **xpto**.
      (Onde **n** é o numero da linha e **xpto** é o nome da função).
  - se uma função possui retorno
    * em caso incorreto, apresente a mensagem:
      Erro semântico na linha **n**. Função **xpto** deve ter um retorno explícito.
      (Onde **n** é o numero da linha e **xpto** é o nome da função).

- a documentação do projeto deve ser atualizada adequadamente

3.2 Avaliação

A avaliação desse módulo ocorrerá por meio de compilação dos códigos-fontes e, em seguida, execução do programa gerado sobre os arquivos de exemplo, contendo cada um, um algoritmo, da seguinte maneira:

$> semantico entrada.por

onde:

- **semantico**: nome do programa constituído pelo analisador semântico
- **entrada.por**: arquivo, com a extensão **.por**, que conterá um algoritmo escrito na pseudo-linguagem Portugol e que será analisado.

3.3 Considerações importantes

- encontrado o primeiro erro, **saia do programa**.
- obrigatoriamente, o diretório de exemplos deve conter, além de 5 algoritmos corretamente escritos, e outros 5 algoritmos contendo erros semânticos para fins de testes.
- o executável gerado na compilação deste módulo deve ter o nome **semantico.exe**.

3.4 Entrega

A entrega do módulo deve ser realizada via *release* no repositório do git, até a data definida no quadro 3.

O conteúdo da entrega deve contemplar **todos** os códigos-fontes necessários para a compilação (sem erros) do programa a ser avaliado, assim como os algoritmos de exemplo.

4 Interpretador

Deadline:	06/12/2019 23:59:59
Pontuação:	20,0

Tabela 4: Calendário de entrega do interpretador

O objetivo desta etapa é a integração de todos os módulos desenvolvidos nas etapas anteriores, gerando o interpretador para o Portugol.

4.1 Instruções

- os módulos desenvolvidos nas etapas anteriores, devem ser corretamente incorporados a este módulo final.
- o interpretador deve receber como entrada o algoritmo a ser interpretado. Para esse fim, utilize o tratamento adequado de parâmetros obtidos pela função *main(...)*.
- todos os erros tratáveis devem estar devidamente documentados (por exemplo, no dicionário de dados)
- a documentação final do projeto deve estar devidamente atualizada e formatada

4.2 Avaliação

A avaliação desse módulo ocorrerá por meio de compilação dos códigos-fontes de todos os módulo e, em seguida, execução do programa gerado sobre os arquivos de exemplo, contendo cada um, um algoritmo, da seguinte maneira:

$> interpretar entrada.por

onde:

- **interpretar**: nome do interpretador constituído
- **entrada.por**: arquivo, com a extensão **.por**, que conterá um algoritmo escrito na pseudo-linguagem Portugol e que será interpretado.

4.3 Considerações importantes

- encontrado o primeiro erro, **saia do programa**.
- obrigatoriamente, o diretório de exemplos deve conter, além de 5 algoritmos corretamente escritos, e outros 5 algoritmos contendo erros para fins de testes.
- o executável gerado na compilação final do projeto deve ter o nome **interpretar.exe**.

4.4 Entrega

A entrega final deve ser realizada via *release* no repositório do git, até a data definida no quadro 4.

O conteúdo da entrega deve contemplar **todos** os códigos-fontes necessários para a compilação (sem erros) do programa a ser avaliado, assim como os algoritmos de exemplo.

5 Referências

1. **Flex & Bison**: consultar o livro para entendimento de
   - uso do Flex (ver: capítulo 1)
   - uso do Bison (ver: capítulo 2)
   - exemplos de implementação de **árvore semântica** (ver: capítulo 3)
2. **Manual do G-Portugol**: consultar o Apêndice A do manual de G-Portugol
3. **Utilização do Flex**: consultar as atividades práticas executadas em sala, usando Flex