UNIOESTE – Universidade Estadual do Oeste do Paraná
Curso de Ciência da Computação – 3º ano
Disciplina de Compiladores
Profª. Camile Bordini

TRABALHO 3 – Análise Léxica, Sintática e Semântica

O trabalho 3 da disciplina é composto pelas seguintes partes:

i. Incrementação do software criado no Trabalho 1 (fase **léxica**), adicionando a parte **sintática** e **semântica** de um compilador;

ii. Relatório técnico do sistema desenvolvido;

iii. Apresentação do sistema pela equipe através de seminário de curta duração.

A seguir, encontram-se as regras do trabalho, especificações, forma de entrega e critérios de avaliação.

Regras

- **Prazo de entrega: 26/06/2023**
- Trabalho **em duplas**
- Valor: 100 pontos (peso 2)
- Envio pelo Teams (comprimir numa pasta todos os arquivos). Apenas um da equipe envia.

Especificação do Trabalho

1. Implementar um software que simula um compilador capaz de realizar a três etapas de análise:

- **Léxica** (Trabalho 1): criar expressões regulares ou autômatos finitos para reconhecer **no mínimo 14 classes de *tokens*** para uma linguagem de programação procedural estruturada simplificada, não orientada a objetos.
- **Sintática**: criar uma gramática com **no mínimo 20 regras sintáticas** para reconhecer todas as classes de *tokens* criados na fase léxica.
- **Semântica**: definir uma gramática de atributos que deve calcular ao menos um atributo, como: valor, base ou tipo de dados. O(s) atributo(s) em questão deve(m) estar presente(s) em **ao menos 4 regras** da gramática de atributos com símbolos não terminais à esquerda do sinal de igual, exemplo:
  - `numero.val = num.val`
  - `digito.val = 0`

Obs: A definição das regras podem ser baseadas em outras linguagens, como C, Pascal, Basic, Fortran, etc.

Comandos (mínimos) que devem ser reconhecidos
-Declaração de variáveis com os tipos suportados pela linguagem -Atribuições -Operadores para expressões aritméticas, relacionais e lógicas combinadas, com suas precedências e associatividades -Estruturas condicionais (`if` "sozinho" e `if+else`) -Estruturas de repetição (pelo menos 2 tipos) -Leitura e escrita de dados -Retorno da função principal -Inclusão de bibliotecas *Obs*: Como a linguagem de programação deve ser estruturada, o compilador deve permitir a declaração de blocos internos uns aos outros. Exemplos: `if's` e laços aninhados, etc.

2. O software ao ser executado deve:

- Pedir para o usuário digitar ou selecionar o nome do arquivo fonte (Ex: `fonte1.txt`, `fonte2.txt`).
- Ler o código fonte a partir do arquivo selecionado:
  - Criar um arquivo correto ("sem erros de linguagem") de código fonte chamado `fonte1.txt`;
  - Criar um arquivo incorreto ("com **erros léxicos, sintáticos e semânticos**") de código fonte chamado `fonte2.txt`. É obrigatório constarem os 3 tipos de erros.
- Reconhecer classes de *tokens* no código fonte por meio da **análise léxica** e inserção dos mesmos em estruturas de dados apropriadas (tabela de símbolos, tabela de palavras reservadas) – Trabalho 1.
  - O projeto deve ser melhorado com as sugestões dadas durante os seminários do Trabalho 1.
- Reconhecer comandos (sequências de *tokens*) no código fonte por meio de **análise sintática**;
- Usar gramática de atributos como apoio para a **análise semântica**;
- Realizar **tratamento de erros léxicos, sintáticos e semânticos**, mostrando na tela qual o tipo de erro (léxico, sintático ou semântico) e apontando a posição do erro. Para ter uma maior cobertura de erros diferentes, é útil pesquisar os tipos de erros léxicos mais comuns em linguagens populares.

- Se o código fonte não contém nenhum erro de compilação, para pelo menos 5 comandos desse código deve-se **gerar a árvore sintática** correspondente aos comandos, de modo similar a uma árvore de diretórios. A equipe pode escolher se gera uma árvore só correspondente ao código do programa ou várias árvores, uma por comando. A(s) árvore(s) resultante(s) deve(m) ser escrita(s) em arquivo(s) texto(s). Por exemplo, para o arquivo `fonte1.txt`, sem erros de compilação, deve-se gravar a árvore correspondente no arquivo denominado `arvores_fonte1.txt`.

Obs: Fica a critério da equipe a linguagem de programação utilizada para implementar o software, e em utilizar ou não ferramentas de apoio para construção de compiladores, como Flex/Bison, Lex/Yacc ou JavaCC.

3. O **Relatório Técnico** do sistema deve conter no mínimo as seguintes informações:

- Nome dos integrantes da equipe;
- Nome do software/linguagem criada;
- Descrição do funcionamento do software que realiza a análise léxica, sintática e semântica, incluindo informações para orientar o usuário sobre como executar o software com autonomia;

- Sobre as etapas:
  - **Léxica**: descrição da linguagem, com todas as expressões regulares ou autômatos usados para reconhecer *tokens*, bem como citação dos trechos do software desenvolvido associados especificamente a cada expressão regular ou autômato (Trabalho 1);
  - **Sintática**: descrição da gramática para análise sintática, incluindo representação das regras em notação BNF ou EBNF;
  - **Semântica**: descrição da gramática de atributos, com a categorização dos atributos como “herdado” ou “sintetizado”, e a associação entre regras dessa gramática com regras da gramática para análise sintática.
    - *Obs*: Citação dos trechos do software desenvolvido associados às regras gramaticais;

- **Tabela de Símbolos**: descrição textual ou representação gráfica que ilustra o conteúdo da tabela de símbolos antes e depois do processamento de um comando de um código-fonte sem erros e menção às operações que foram realizadas nessa tabela -- a tabela deve conter todos os atributos indicados na gramática de atributos, além de cadeia, *token* e categoria;

- **Árvore sintática**: apresentação de uma das árvores sintáticas geradas pelo software;

- Documentar os principais métodos/funções do código fonte (ou as especificações Lex, Yacc, JavaCC,...) do software que realiza as análises léxica, sintática e semântica. Essa documentação deve incluir uma breve descrição da finalidade de cada método e dos seus parâmetros;
- Descrição do processo de construção do software desenvolvido, incluindo configurações, bibliotecas, ferramentas auxiliares e IDEs utilizadas.
- Referências bibliográficas utilizadas.

Forma de entrega e Apresentações

- Encaminhar todo o projeto (software, relatório técnico, bibliotecas necessárias para execução, os arquivos `fonte1.txt` e `fonte2.txt`), em um único arquivo comprimido pelo Teams.
- A apresentação dos trabalhos será no dia **28/06** ~~e (possivelmente) **05/07** dependendo do cronograma da disciplina~~. A ordem de apresentação das equipes será sorteada. Todos os membros da equipe devem participar da apresentação.
- A equipe deve demonstrar o software executando (o mesmo que foi entregue dentro do prazo) utilizando os arquivos com e sem erros, incluindo exibição de parte da árvore sintática para o código sem erro.
- Criar slides para a apresentação contendo uma descrição breve da linguagem criada e visão geral das gramáticas implementadas. Não é preciso enviar os slides no dia da entrega do trabalho.
- Tempo de duração da apresentação: **até 10 minutos**.

Critérios de avaliação

- Software conforme especificação do trabalho: atendimento aos requisitos, organização e legibilidade (clareza, indentação, modularização, comentários), execução sem erros – **60% da nota**.
- Relatório técnico: qualidade na escrita, estar compatível com o software – **30% da nota**.
- Apresentação (clareza, conhecimento demonstrado) – **10% da nota**.

- Observações:
  - Atrasos na entrega implicará em perda de pontos.
  - Perguntas individuais poderão ser feitas e consideradas para a nota individual do aluno.

- Em caso de identificação de trabalhos iguais, a nota 0 será atribuída para todos os envolvidos.
- ~~Caso, por questões de cronograma, não seja possível que todas as equipes apresentem nas datas estipuladas, algumas equipes podem ser solicitadas a realizarem a apresentação por meio de vídeo gravado e enviado à professora.~~
- Para as equipes que não der tempo de apresentar no dia 28/06, deverão agendar um horário fora de aula com a professora.

Referências indicadas

AHO, A.V; LAM, M.S.; SETHI, R.; ULLMAN, J.D. Compiladores: princípios, técnicas e ferramentas.. Pearson. 2008.

DELAMARO, M.E. Como construir um compilador utilizando Ferramentas Java. Novatec Editora. 2004.

LOUDEN, K.C. Compiladores: princípios e práticas. Pioneira. 2004.

Websites como os que seguem:

https://johnidm.gitbooks.io/compiladores-para-humanos/content/part1/lexical-analysis.html

https://johnidm.gitbooks.io/compiladores-para-humanos/content/part1/syntax-analysis.html

https://johnidm.gitbooks.io/compiladores-para-humanos/content/part1/semantic-analysis.html

http://www.cs.um.edu.mt/~sspi3/CSA2201_Lecture-2012.pdf

https://cse.iitkgp.ac.in/~bivasm/notes/LexAndYaccTutorial.pdf