UFES

Trabalho de Compiladores

Ciência da Computação
Prof. Rodrigo Freitas Silva

Um Compilador para Mini Java

1) Implemente um analisador léxico para a linguagem Mini Java cuja gramática é descrita no Anexo I. Siga as instruções e observações abaixo:

a) Data de entrega do trabalho: _____

b) Trabalho poderá ser feito em grupo de até 4 pessoas

i. Ao entregar o relatório final, mencionar a porcentagem de quanto cada integrante do grupo colaborou com o trabalho.

c) Você deverá criar um IDE para o desenvolvimento e compilação dessa linguagem

d) **Lembre-se** que o analisador léxico é um identificador de padrões, caracterizando os lexemas lidos em tokens da linguagem.

e) O programa deverá diferenciar letras maiúsculas de letras minúsculas

f) Os identificadores:

i. Identificadores com caracteres acentuados são diferentes dos não acentuados. Ex: História é diferente de Historia
ii. Não tem limite para o número de caracteres de um identificador
iii. Não pode ser uma palavra reservada da linguagem
iv. Não pode ser **true** nem **false**
v. Não pode ser **null**
vi. Deve obrigatoriamente começar com uma letra (a-z, A-Z), underline (_) ou cifrão ($) seguida por estes mesmos caracteres ou números

g) Os comentários deverão ser removidos do programa.

```
// Este é um comentário de uma linha

/*  Aqui é
um comentário
de múltiplas linhas  */
```

h) Os espaços em branco também são automaticamente removidos pelo compilador

i) Quando for encontrado um erro léxico, seu analisador deverá indicar a linha no programa no qual o erro foi encontrado, assim como sublinhar de vermelho o erro no programa fonte

   i. Você deve TENTAR encontrar esses erros léxicos não somente após o usuário mandar compilar o programa, mas também enquanto ele estiver implementando

j) Lembre-se de incluir todos os tokens conhecidos antecipadamente na tabela de símbolos (palavras reservadas, sinais de pontuação, operadores). Inclua também os tokens encontrados no programa objeto (identificadores)

k) Você deverá escolher pelo menos uma estratégia de recuperação de erros

l) Ao final (na data marcada com o professor) , **você deverá apresentar seu trabalho e entregar um relatório** descrevendo o funcionamento do seu IDE e do seu analisador léxico, como foi o desenvolvimento do seu trabalho, o diagrama relacionando os módulos do seu programa, uma tabela de (tokens, lexemas, expressões regulares e atributos) para cada token encontrado na linguagem

2) Observações:

- Algumas instruções do Java estão ausentes como: Vetores, laço for, switch, operadores de incremento (++ e --), interfaces, ...
- Qualquer uma dessas estruturas que forem incorporadas valerá ponto extra
- Para análise Sintática: Pode haver instruções com somente um **ID ;** ou somente uma instrução com somente o **; .** Para casos como esse:
  - Avisar com '*Warning*' que a instrução não tem efeito nenhum na instrução durante a Análise Sintática
- Palavras reservadas da linguagem:

abstract | boolean | break | byte | case | catch | char | class | const | continue | default | do | double | else | extends | final | finally | float | for | goto | if | implements | import | instanceof | int | interface | long| native | new | package | private | protected | public | return | short | static | super | switch | synchronized | this | throw | throws | transient | try | void | volatile | while

ANEXO I

- <*program*> → <*packageDeclaration*>? <*importDeclaration*>? <*classDeclaration*>?
- <*packageDeclaration*> → **package** <*identifier*> **;**
- <*importDeclaration*> → **import** < *identifier*> **;** <*importDeclaration*> | E
- <*classDeclaration*> → <*classModifier*>? **class** <*identifier*> <*classBody*> {permitido somente uma definião de classe por unit}
- <*classModifier*> → **public** | **abstract** | **final** | E
- <*classBody*> → **{** <*classBodyDeclaration*>? **}**
- <*classBodyDeclaration*> → <*Modifiers*>?<*Type*><*identifier*><*fieldMethodDeclaration*> <*classBodyDeclaration*> | E
- <*fieldMethodDeclaration*> → <*variableDeclarators*> **;** | <*methodDeclaration*>
- <*Modifier*> → **public** | **protected** | **private** | **static** | **final** **{ pode tern a LP somente um destes modifier ou nehum deles}**
- <*variableDeclarators*> → **=** <*expression*> | **,** <*identifier*> <*variableDeclarators*> | E
- <*methodDeclaration*> → <*methodDeclarator*> <*block*>
- <*methodDeclarator*> → **(** <*formalParameter*>? **)**
- <*formalParameter*> → <*type*> <*identifier*> <*formalParameterList*> | E
- <*formalParameterList*> → , <*type*> <*identifier*> <*formalParameterList*> | E
- <*type*> → **byte** | **short** | **int** | **long** | **char** | **float** | **double** | **boolean | void**
- <*block*> → **{** <*blockStatement*>? **}**
- <*blockStatement*> → <*localVariableDeclaration*> **;** <*blockStatement*> | <*statement*> <*blockStatement*> | E
- <*localVariableDeclaration*> → <*type*><*identifier*> <*variableDeclarators*>
- <*statement*> → <*block*> | <*emptyStatement*> | <*identifier*><*statementExpression*> **;** | <*doStatement*> | <*breakStatement*> | <*continueStatement*> | <*returnStatement*> |<*ifSstatement*> | <*whileStatement*> | <*tryStatement*> | <*classInstanceCreationExpression*>
- <*emptyStatement*> → **;**
- <*labeledStatement*> → **:** <*statement*>

- <*statementExpression*> → <*assignment*> | <*methodInvocation*> | <*labeledStatement*>
- <*ifStatement*>→ **if (** <*expression*> **)** <*statement*> <*ifThenElseStatement*> ?
- <*ifThenElseStatement*>→ **else** <*statement*>
- <*whileStatement*> → **while (** <*expression*> **)** <*statement*>
- <*doStatement*> → **do** <*statement*> **while (** <*expression*> **) ;**
- <*breakStatement*> → **break** <*identifier*>? **;**
- <*continueStatement*> → **continue** <*identifier*>? **;**
- <*returnStatement*> → **return** <*expression*>? **;**
- <*tryStatement*> → **try** <*block*> <*catchesStatement*>
- <*catchesStatement*> → <*catchClause*> <*finally*> ? | <*finally*>
- <*catchClause*> → **catch (** <*formalParameter*> **)** <*block*> <*catchClause*> | E
- <*finally* > → **finally** <*block*>
- <*assignment*> → <*assignmentOperator*> <*expression*>
- <*assignmentOperator*> → = | ***=** | **/=** | **%=** | **+=** | **-=** | **<<=** | **>>=** | **>>>=** | **&=** | **^=**
- <*expression*> → <*comparationExpression*> <*expression2*> *//Faltou inserir parenteses*
- <*expression2*> → || <*comparationExpression*> <*expression2*> | *&&* <*comparationExpression*> <*expression2*> | ϵ
- <*comparationExpression*> → <*operationalExpression*> <*relationalExpression*>
- <*relationalExpression*> → *==* <*operationalExpression*> <*relationalExpression*> | *!=* <*operationalExpression*> <*relationalExpression*>

  | < <*operationalExpression*> <*relationalExpression*>

  | <= <*operationalExpression*> <*relationalExpression*>

  | > <*operationalExpression*> <*relationalExpression*>

  | >= <*operationalExpression*> <*relationalExpression*>

  | ϵ

- <*operationalExpression*> → <*term*> <*additiveExpression*>
- <*additiveExpression*>→ + <*term*> <*additiveExpression*>| - <*term*> <*additiveExpression*> | ϵ
- <*term*> → <*unaryExpression*> <*multiplicativeExpression*>
- <*multiplicativeExpression*> → * <*unaryExpression*> <*multiplicativeExpression*> | / <*unaryExpression*> <*multiplicativeExpression*> | % <*unaryExpression*> <*multiplicativeExpression*> | ϵ
- <*unaryExpression*> → + <*identifier1*> <*methodInvocation*>? | - <*identifier1*><*methodInvocation*>? | <*identifier1*><*methodInvocation*>?
- <*methodInvocation*> → **(** <*argumentList*>? **)** | **super .** <*identifier*> **(** <*argumentList*>? **)**
- <*classInstanceCreationExpression*> → **new** <*classType*> **(** <*argumentList*>? **)**
- <*argumentList*> → <*expression*> | <*argumentList*> **,** <*expression*>

<Identifier > aceita nome

<Identifier1 > aceita nome, int, real