UNIVERSIDADE FEDERAL DE MATO GROSSO DO SUL
FACOM - FACULDADE DE COMPUTAÇÃO

COMPILADORES I - 2021/1

PROFA. BIANCA DE ALMEIDA DANTAS

Trabalho Prático
Descrição da 2^a Etapa – Análise Sintática

1 DESCRIÇÃO

A segunda parte do trabalho prático de nossa disciplina consiste na implementação do analisador sintático descendente preditivo para a linguagem *X++*, descrita no livro "Como Construir um Compilador Utilizando Ferramentas Java" de Márcio Eduardo Delamaro, com algumas alterações, cujos tokens já foram descritos na primeira etapa do trabalho e cuja gramática encontra-se no final deste arquivo.

O analisador sintático deve ser capaz de percorrer o programa fonte, detectar e reportar erros. **Não é necessário implementar nenhuma técnica de recuperação de erros sintáticos**.

Os códigos fontes do programa desenvolvido serão compilados usando o compilador g++.

2 EXECUÇÃO E ENTRADA

O seu programa deve ser capaz de realizar a compilação de um arquivo de texto com a extensão **.xpp**, cujo nome será fornecido na linha de comando do terminal logo após o nome do executável de seu compilador. Por exemplo, se seu executável possuir o nome **xpp_compiler** e o arquivo de entrada for **teste1.xpp**, a seguinte instrução será digitada no terminal:

```
./xpp_compiler teste1.xpp
```

3 SAÍDA

O compilador deve emitir mensagens de erros, caso encontre algum, informando claramente o erro e a linha de ocorrência. Caso não sejam encontrados erros, o compilador deve imprimir que a compilação foi encerrada com sucesso. Todas as mensagens devem ser mostradas no terminal. Sugere-se usar como inspiração mensagens geradas por compiladores reais (como o próprio g++).

4 AVALIAÇÃO

O seu programa será compilado usando o comando:

```
g++ *.cpp -o xpp\_compiler
```

Caso a compilação gere erros e o executável não seja gerado, o trabalho receberá nota zero.

O programa será executado com n arquivos fontes, podendo conter erros ou não, e a nota atribuída será proporcional ao número de testes cuja execução de seu compilador conseguir detectar

os erros (ou a falta deles) corretamente. Testes em que a execução não gerar o resultado esperado serão zerados.

O programa deve receber a entrada e gerar a saída **exatamente** como especificado nas descrições das etapas, caso isso não ocorra, a nota será penalizada.

5 ESPECIFICAÇÕES

- O trabalho prático poderá ser realizado em grupos de, no máximo, 3 alunos **sem exceções**.
- A linguagem C++ deverá ser utilizada na implementação do trabalho.
- Se qualquer parte do trabalho for copiada ou "fortemente inspirada" nos trabalhos de outros grupos, o trabalho receberá nota **zero**.
- Entrevistas podem ser realizadas com todos os grupos.
- A entrega de todas as etapas deve ser realizada até o dia: **25/06/2021**.

6 GRAMÁTICA DA LINGUAGEM X++

1. *Program* → *ClassList*
   | ϵ
2. *ClassList* → *ClassDecl* *ClassList*
   | *ClassDecl*
3. *ClassDecl* → **class ID** *ClassBody*
   | **class ID extends ID** *ClassBody*
4. *ClassBody* → **{** *VarDeclListOpt ConstructDeclListOpt MethodDeclListOpt* **}**
5. *VarDeclListOpt* → *VarDeclList*
   | ϵ
6. *VarDeclList* → *VarDeclList VarDecl*
   | *VarDecl*
7. *VarDecl* → *Type* **ID** `VarDeclOpt` **;**
   | *Type* **[] ID** `VarDeclOpt` **;**
8. *VarDeclOpt* → **, ID** *VarDeclOpt*
   | ϵ
9. *Type* → **int**
   | **string**
   | **ID**
10. *ConstructDeclListOpt* → *ConstructDeclList*
    | ϵ
11. *ConstructDeclList* → *ConstructDeclList ConstructDecl*
    | *ConstructDecl*
12. *ConstructDecl* → **constructor** *MethodBody*
13. *MethodDeclListOpt* → *MethodDeclList*
    | ϵ
14. *MethodDeclList* → *MethodDeclList MethodDecl*
    | *MethodDecl*
15. *MethodDecl* → *Type* **ID** *MethodBody*
    | *Type* **[] id** *MethodBody*
16. *MethodBody* → **(** *ParamListOpt* **) {** *StatementsOpt* **}**
17. *ParamListOpt* → *ParamList*
    | ϵ
18. *ParamList* → *ParamList* **,** *Param*
    | *Param*
19. *Param* → *Type* **ID**
    | *Type* **[] ID**
20. *StatementsOpt* → *Statements*
    | ϵ
21. *Statements* → *Statements* *Statement*
    | *Statement*
22. *Statement* → *VarDeclList*
    | *AtribStat* **;**
    | *PrintStat* **;**
    | *ReadStat* **;**
    | *ReturnStat* **;**
    | *SuperStat* **;**

| *IfStat*
| *ForStat*
| **break ;**
| **;**

23. *AtribStat* → *LValue* **=** *Expression*
| *LValue* **=** *AllocExpression*
24. *PrintStat* → **print** *Expression*
25. *ReadStat* → **read** *LValue*
26. *ReturnStat* → **return** *Expression*
27. *SuperStat* → **super (** *ArgListOpt* **)**
28. *IfStat* → **if (** *Expression* **) {** *Statements* **}**
| **if (** *Expression* **) {** *Statements* **} else {** *Statements* **}**
29. *ForStat* → **for (** *AtribStatOpt* **;** *ExpressionOpt* **;** *AtribStatOpt* **) {** *Statements* **}**
30. *AtribStatOpt* → *AtribStat*
| ϵ
31. *ExpressionOpt* → *Expression*
| ϵ
32. *LValue* → **ID** *LValueComp*
| **ID [** *Expression* **]** *LValueComp*
33. *LValueComp* → **. ID** *LValueComp*
| **. ID [** *Expression* **]** *LValueComp*
| ϵ
34. *Expression* → *NumExpression*
| *NumExpression* **RelOp** *NumExpression*
35. *AllocExpression* → **new ID (** *ArgListOpt* **)**
| *Type* **[** *Expression* **]**
36. *NumExpression* → *Term* **+** *Term*
| *Term* **-** *Term*
| *Term*
37. *Term* → *UnaryExpression* ***** *UnaryExpression*
| *UnaryExpression* **/** *UnaryExpression*
| *UnaryExpression* **%** *UnaryExpression*
| *UnaryExpression*
38. *UnaryExpression* → **+** *Factor*
| **-** *Factor*
39. *Factor* → **INTEGER_LITERAL**
| **STRING_LITERAL**
| *LValue*
| **(** *Expression* **)**
40. *ArgListOpt* → *ArgList*
| ϵ
41. *ArgList* → *ArgList* **,** *Expression*
| *Expression*