

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

COMPILADORES I - 2025/2  
PROFA. BIANCA DE ALMEIDA DANTAS

**Trabalho Prático**  
**Descrição da 2<sup>a</sup> 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 e, também, em um arquivo separado.

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.
- A entrega de todas as etapas deve ser realizada até o dia: **14/11/2025**.

## 6 GRAMÁTICA DA LINGUAGEM X++

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

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