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

COMPILADORES I (2024/2)
PROFA. BIANCA DE ALMEIDA DANTAS

Trabalho Prático
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 *C--*, cuja gramática se encontra ao final desse texto.

O analisador sintático deve ser capaz de percorrer o programa fonte, detectar e reportar erros. Não é necessário implementar estratégias de recuperação de erros, entretanto, caso o trabalho forneça alguma estratégia funcional para realizar essa atividade, o trabalho poderá receber um bônus de até (um) ponto em sua nota total.

2 EXECUÇÃO E ENTRADA

O seu programa deve ser capaz de realizar a compilação de um arquivo de texto com a extensão **.cmm**, 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 **mj_compiler** e o arquivo de entrada for **teste1.cmm**, a seguinte instrução será digitada no terminal:

```
./cmm teste1.cmm
```

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

A não ser que você especifique a utilização de outras opções de compilação ou forneça um makefile, o seu programa será compilado usando o comando seguinte:

```
g++ *.cpp -o cmm
```

Caso a compilação gere erros e o executável não seja gerado, o trabalho receberá nota zero. **Observe que o seu grupo deve garantir a compilação utilizando o compilador g++, o trabalho não será testado no sistema operacional Windows.**

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 terão atribuída a nota zero.

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: **11/11/2024**.

6 GRAMÁTICA

A gramática seguinte utiliza as notações **(N)*** para representar 0 ou mais repetições de **N** e a notação **(N)?** para representar 0 ou 1 repetição de **N**. Os tokens da linguagem são representados em vermelho e os não-terminais em *itálico*.

1. *Program* → (*Function*)* EOF

2. *Function* → *Type* ID(*ParamTypes*){(*Type* *VarDeclaration*(, *VarDeclaration*)*;)*
   (*Statement*)*}
   | void ID(*ParamTypes*){(*Type* *VarDeclaration*(, *VarDeclaration*)*;)* (*Statement*)*}

3. *VarDeclaration* → ID ([integerconstant])?

4. *Type* → char
   | int

5. *ParamTypes* → void
   | *Type* ID([])?(,*Type* ID([])?)*

6. *Statement* → if (*Expression*) *Statement* (else *Statement*)?
   | while (*Expression*) *Statement*
   | for ((*Assign*)?; (*Expression*)?; (*Assign*)?) *Statement*
   | return (Expression)?;
   | *Assign* ;
   | ID((*Expression*(, *Expression*)*)?);
   | { (*Statement*)* }
   | ;

7. *Assign* → ID ([*Expression*])? = *Expression*

8. *Expression* → - *Expression*
   | ! *Expression*
   | *Expression* BinOp *Expression*
   | *Expression* Relop *Expression*
   | *Expression* LogOp *Expression*
   | ID((*Expression*(, *Expression*)*)?)|[*Expression*])?
   | (*Expression*)
   | integerconstant
   | charconstant
   | stringconstant

9. *BinOp* → +
   | -
   | *

| /

10. *RelOp* → ==
| !=
| <=
| <
| >=
| >

11. *LogOp* → &&
| ||