

# Introdução ao Flex e Bison

Lucas Begnini Costa // André Moreira // George Lucas Gomes // Luiz Sergio // Walderi Filho

Universidade do Estado do Amazonas

May 25, 2013



# Introdução ao Flex e Bison

Lucas Begnini Costa // André Moreira // George Lucas Gomes // Luiz Sergio // Walderi Filho

Universidade do Estado do Amazonas

May 25, 2013

# Sumário

## 1 Introdução

## 2 FLEX

- Instalando Flex no Linux
- Criando um arquivo para o Flex

## 3 Bison

- Instalando Bison no Linux
- Criando um arquivo para o Bison

## 4 Calculadora

- Arquivo Flex da Calculadora
- Arquivo Bison da Calculadora
- Árvore Sintática da Calculadora

# Introdução

Este trabalho tem como intuito apresentar a construção de uma calculadora que será utilizada através de linha de comando e interpretada através de:

- Flex - Analisador léxico

Para apresentar a calculadora será necessário uma breve apresentação a respeito de Flex e Bison

# Introdução

Este trabalho tem como intuito apresentar a construção de uma calculadora que será utilizada através de linha de comando e interpretada através de:

- Flex - Analisador léxico
- Bison - Analisador semântico

Para apresentar a calculadora será necessário uma breve apresentação a respeito de Flex e Bison

# FLEX

Como falado anteriormente o Flex é uma ferramenta para criação de analisadores Léxicos que procura padrões de escrita através de expressões regulares definidas dentro dos analisadores léxicos. Como resultado, depois da compilação do arquivo flex feito, é gerado um arquivo em C que valida todas as expressões regulares definidas no arquivo que passará pelo Flex

# Instalando Flex no Linux

Antes de fazer qualquer coisa para criação do arquivo flex é necessário inicialmente a instalação do compilador flex no sistema operacional, no caso utilizado foi o Linux Ubuntu 13.04. Para instalação é necessário abrir o terminal e por linha de comando digitar:

```
$ sudo apt-get install flex
```

# Criando um arquivo para o Flex

Para a criação de um arquivo para o flex não é necessário nenhuma extensão específica, podendo ser o que o desenvolvedor quiser, mas como um ato de boa pratica de programação, padronizou-se como `.lex` ou `.l` para melhor distinguir de outros arquivos.



# Criando um arquivo para o Flex

Dentro do programa há duas partes, separado por "%%" dividindo o programa em duas partes: A primeira para delimitação das expressões regulares que serão consideradas, e a segunda as ações que serão tomadas caso encontre as expressões regulares definidas anteriormente. Como podemos ver no exemplo seguinte: **Exemplo de Flex**

# Criando um arquivo para o Flex

Para compilação de um arquivo flex é necessário 2 etapas:

- Compilação em flex primeiro do seguinte modo, pela linha de comando do linux:

Na primeira compilação ele pega arquivo de flex e gera um arquivo .C que vai validar as expressões regulares O parametro "-lfl" passado na compilação do gcc serve pra identificar que foi usado o flex pra gerar o .C

# Criando um arquivo para o Flex

Para compilação de um arquivo flex é necessário 2 etapas:

- Compilação em flex primeiro do seguinte modo, pela linha de comando do linux:
- `flex -o nome-do-programa.lex.c nome-do-programa.lex`

Na primeira compilação ele pega arquivo de flex e gera um arquivo .C que vai validar as expressões regulares O parametro "-lfl" passado na compilação do gcc serve pra identificar que foi usado o flex pra gerar o .C

# Criando um arquivo para o Flex

Para compilação de um arquivo flex é necessário 2 etapas:

- Compilação em flex primeiro do seguinte modo, pela linha de comando do linux:
- `flex -o nome-do-programa.lex.c nome-do-programa.lex`
- Em seguida é necessária a compilação em GCC com o seguinte comando

Na primeira compilação ele pega arquivo de flex e gera um arquivo .C que vai validar as expressões regulares O parametro "-lfl" passado na compilação do gcc serve pra identificar que foi usado o flex pra gerar o .C

# Criando um arquivo para o Flex

Para compilação de um arquivo flex é necessário 2 etapas:

- Compilação em flex primeiro do seguinte modo, pela linha de comando do linux:
- `flex -o nome-do-programa.lex.c nome-do-programa.lex`
- Em seguida é necessária a compilação em GCC com o seguinte comando
- `gcc -o nome-do-programa nome-do-programa.lex.c -lfl`

Na primeira compilação ele pega arquivo de flex e gera um arquivo .C que vai validar as expressões regulares O parametro "-lfl" passado na compilação do gcc serve pra identificar que foi usado o flex pra gerar o .C

# Bison

Bison é um gerador de interpretadores para fins gerais que converte uma gramática livre de contexto anotado em uma linguagem regular aplicando um parser nas tabelas do analisador lexico.

# Instalando o Bison no Linux

Antes de fazer qualquer coisa para criação do arquivo Bison é necessário inicialmente a instalação do compilador Bison no sistema operacional, no caso utilizado foi o Linux Ubuntu 13.04. Para instalação é necessário abrir o terminal e por linha de comando digitar:

```
$ sudo apt-get install bison
```

# Criando um arquivo para o Bison

- Compilação em Bison primeiro do seguinte modo, pela linha de comando do linux:



# Criando um arquivo para o Bison

- Compilação em Bison primeiro do seguinte modo, pela linha de comando do linux:
- `bison -d nome-do-programa.y`

# Criando um arquivo para o Bison

- Compilação em Bison primeiro do seguinte modo, pela linha de comando do linux:
- `bison -d nome-do-programa.y`
- Em seguida é necessária a compilação do arquivo lexico no caso o flex com o seguinte comando

# Criando um arquivo para o Bison

- Compilação em Bison primeiro do seguinte modo, pela linha de comando do linux:
- `bison -d nome-do-programa.y`
- Em seguida é necessária a compilação do arquivo lexico no caso o flex com o seguinte comando
- `flex -o nome-do-programa.lex.c nome-do-programa.lex`

# Criando um arquivo para o Bison

- Compilação em Bison primeiro do seguinte modo, pela linha de comando do linux:
- `bison -d nome-do-programa.y`
- Em seguida é necessária a compilação do arquivo lexico no caso o flex com o seguinte comando
- `flex -o nome-do-programa.lex.c nome-do-programa.lex`
- e em seguida a compilação do GCC com o seguinte comando

# Criando um arquivo para o Bison

- Compilação em Bison primeiro do seguinte modo, pela linha de comando do linux:
- `bison -d nome-do-programa.y`
- Em seguida é necessária a compilação do arquivo lexico no caso o flex com o seguinte comando
- `flex -o nome-do-programa.lex.c nome-do-programa.lex`
- e em seguida a compilação do GCC com o seguinte comando
- `gcc -o nome-do-programa nome-do-programa.lex.c nome-do-programa.tab.c -lfl -lm`

# Criando um arquivo para o Bison

Todos os arquivos de bison são terminados em .y

- Depois de definida a BNF o arquivo tem de ser compilado para gerar uma tabela de execucao

# Criando um arquivo para o Bison

Todos os arquivos de bison são terminados em `.y`

- Depois de definida a BNF o arquivo tem de ser compilado para gerar uma tabela de execucao
- o arquivo gerado tem a extensao `.tab.c` para entao ser compilado no gcc

# Calculadora

Como exemplo foi feito uma calculadora.

A calculadora e simples fazendo somente as operacoes basicas de soma, subtracao, multiplicacao e divisao

Ela tambem pode fazer operacoes de potenciacao



# Arquivo Flex da Calculadora

No arquivo flex da calculadora foi dividido em:

- \* caracteres da linguagem

# Arquivo Flex da Calculadora

No arquivo flex da calculadora foi dividido em:

- \* caracteres da linguagem
- \* significado dos lexemas

# Arquivo Bison da Calculadora

No bison foi feito primeiro uma definicao dos tokens da linguagem  
Depois a precedencia das operacoes indicando quais sao os nos que ficaram na raiz da arvore E tambem qual o metodo de derivacao da arvore  
Logo em seguida e definida a BNF da calculadora

# Árvore Sintática da Calculadora

\*Árvore sintática é uma abstração de como é feita a verificação da derivação das expressões de acordo com a BNF.

\*A árvore sintática cria todas as possíveis combinações de expressões que são válidas na linguagem

# Árvore Sintática da Calculadora

Esta é a EBNF da Calculadora.

```
Input:
    | Input Line
;

Line:
    END
    | Expression END { printf("Result: %f\n", $1); }
;

Expression:
    NUMBER { $$=$1; }
    | Expression PLUS Expression { $$=$1+$3; }
    | Expression MINUS Expression { $$=$1-$3; }
    | Expression TIMES Expression { $$=$1*$3; }
    | Expression DIVIDE Expression { $$=$1/$3; }
    | MINUS Expression %prec NEG { $$=-$2; }
    | Expression POWER Expression { $$=pow($1,$3); }
    | LEFT Expression RIGHT { $$=$2; }
;
```

# Árvore Sintática da Calculadora

Exemplos:

1

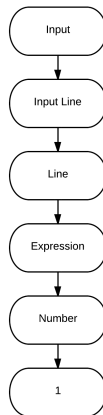
1+3

1+3\*4

1+3\*4-2

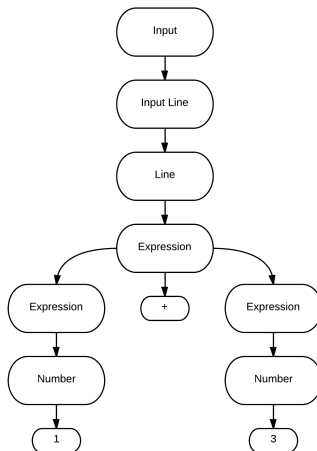
# Árvore Sintática da Calculadora

## Derivando 1



# Árvore Sintática da Calculadora

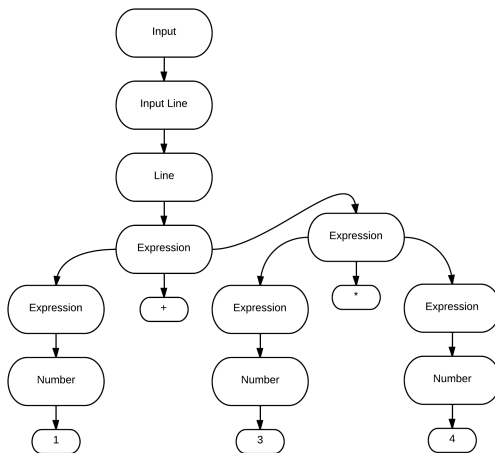
Derivando 1+3





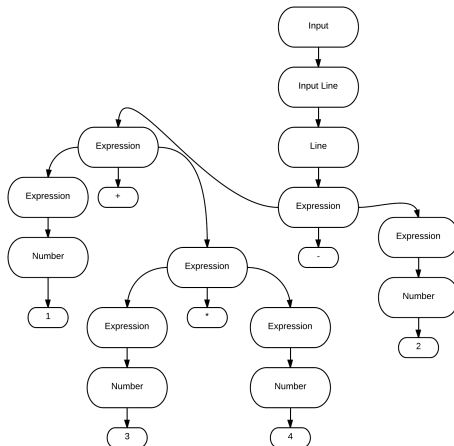
# Árvore Sintática da Calculadora

Derivando  $1+3*4$



# Árvore Sintática da Calculadora

Derivando  $1+3*4-2$



# Árvore Sintática da Calculadora

Expressões com parênteses

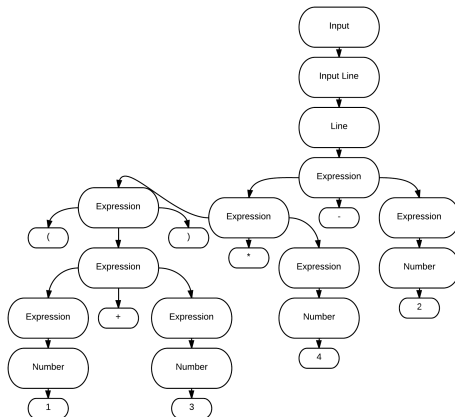
$(1+3)*4-2$

$(1+3)*(4-2)$

$(1+3)*4(-2)$

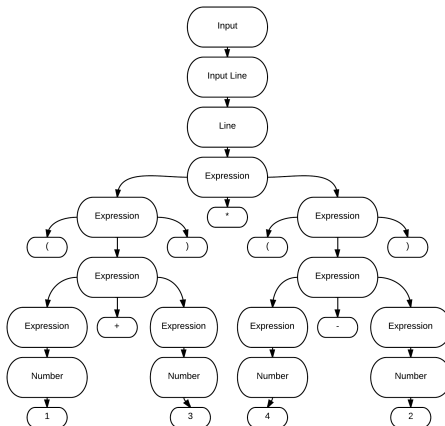
# Árvore Sintática da Calculadora

Derivando  $(1+3)*4-2$



# Árvore Sintática da Calculadora

Derivando  $(1+3)*(4-2)$



# Árvore Sintática da Calculadora

Derivando  $(1+3)*4(-2)$  - ERRO DE SINTAXE

