INSTITUTO MAUÁ DE TECNOLOGIA

MAUÁ

ECM253 – Linguagens Formais, Autômatos e Compiladores

Atividade
Construção de analisadores sintáticos com CUPS

Prof. Marco Furlan

Outubro/2022

Esta atividade possui peso 5.

1 A atividade

Alterar o projeto do **interpretador** de expressões simples com **CUP**, apresentado em aula, para **atender os requisitos a seguir** (implementar e testar na sequência apresentada, que já está em uma ordem lógica):

i. **Manipular números reais** (positivos e negativos). **Sugestão**: no analisador léxico do JFlex, utilizar a seguinte expressão regular:

```
\d+(\.\d+)?(["E""e"]["+""-"]?\d+)?
```

Onde `\d` é um atalho definido no JFlex que casa com um dígito. Analisar nos códigos quais serão as outras modificações que deverão ser realizadas.

ii. **Adicionar** na **gramática** atual uma **regra** de **atribuição** de **expressão** à uma **variável**.

```
expr ::= ID ASSIGN expr
         ;
```

Onde:

- `ID` representa uma **expressão regular** que **casa com qualquer cadeia iniciada** por **letra maiúscula** ou **minúscula seguida** de **zero ou mais letras ou dígitos**. Será necessário **alterar o analisador léxico** para **retornar este símbolo** e, **no CUP**, este símbolo deve ter **tipo** `String`; ASSIGN
- O **símbolo** ~~ATRIB~~, de **atribuição**, é o **terminal** '`=`'. Alterar o analisador léxico para retornar este símbolo. O **operador de atribuição** é um operador cuja **precedência é a mais baixa de todos** os operadores. e sua **associatividade é da direita para a esquerda** (`right`). Com a associatividade da direita para esquerda, será possível realizar atribuições assim (testar) e não gerará erros de sintaxe:

```
x = y = 10;
```

O interpretador não deverá acusar erro léxico ou sintático (mas não executará ação nenhuma).

iii. **Implementar** um **sistema** de **ambiente** para o **interpretador**. Ambiente é o nome dado à parte do programa onde serão armazenados os dados e as variáveis durante a execução de um programa. Neste caso, deseja-se armazenar os valores atribuídos às variáveis em algum lugar da memória para que, depois, possam ser recuperadas e utilizadas em outras expressões.

Uma **forma eficiente** para **armazenar variáveis e seus valores** correspondentes é utilizar uma **tabela de hash**. Em Java, a classe `HashMap` permite criar uma **tabela de variáveis** de modo a armazenar as variáveis durante a execução do programa.

Para fazer isso no **CUP**, modificar `Parser.cup` assim:

- Adicionar a linha a seguir na parte de importações:

```
import java.util.HashMap;
```

- Adicionar uma **seção de código** para declarar um objeto do tipo `HashMap` na classe `Parser`, que será utilizado como **tabela de símbolos**. Ele deve **mapear** nomes de variáveis à números. Adicionar esta seção logo após as importações de pacotes:

```
parser code {:
    // symbolTable é a tabela de símbolos
    private HashMap<String, Double> symbolTable = new HashMap<>();
:}
```

Depois, **alterar a regra de atribuição definida no item anterior** para que, quando houver uma atribuição, seja **adicionado à tabela de símbolos** o **texto com o nome do identificador** e o **número** sendo atribuído. Será necessário:

- **Anotar** com **rótulos** (exemplos: `ID:ide`, `expr:e`) o nome do identificador e valor que se deseja armazenar na tabela de símbolos e depois armazená-los com a operação `put()`. Exemplo:

```
    symbolTable.put(id, e);
```

- **Retornar** em `RESULT` o valor sendo atribuído à variável. Desse modo, uma expressão como `x=y=10;` funcionará corretamente. Testar o código para certificar que está sem erros. Ainda não será possível usar variáveis em expressões, que será tratado no item a seguir.

iv. **Adicionar** uma **regra** à **gramática** para **recuperar valores** de variáveis **e permitir** que elas **sejam utilizadas em expressões**. Basta **adicionar uma regra** como apresentado a seguir:

```
expr ::= ID
        ;
```

Depois, **pensar em uma solução** para **retornar seu valor à expressão** (RESULT) ou **gerar um erro**, caso a variável não esteja presente no ambiente. Utilizar rótulo para obter o texto com o nome da variável. Usar a operação `get()` de `HashMap`. Ela retorna `null` caso o símbolo não esteja na tabela (testar!).

```
Double value = symbolTable.get(id);
```

v. **Adicionar** as **funções matemáticas** `sin(x)` e `cos(x)`, onde `x` é **qualquer expressão** da gramática. Usar as funções Java da classe `Math`. Será necessário alterar tanto o analisador léxico quanto o sintático (pense!).

vi. **Adicionar** na **gramática** atual um **operador** de **exponenciação**, usando '`**`' como símbolo. Exemplo:

```
y = x ** 3;
```

Será necessário alterar tanto o analisador léxico e sintático (pense!). O operador de exponenciação deverá ter uma **precedência maior** que **multiplicação ou divisão**, mas **menor que inversão de sinal**. Sua **associatividade** deve ser da **direita para a esquerda**.

vii. **Adicionar o símbolo** `PI` (π), de modo que, quando usado, retornará o valor de π. **Dica**: usar a constante Java `Math.PI`. Será necessário alterar tanto o analisador léxico quanto o sintático (pense!). Exemplo:

```
y = sin(PI/3);
```

2 O que enviar?

O projeto modificado e compactado no formato ZIP.