

Universidade Federal de Juiz de Fora
Departamento de Ciência da Computação
DCC045 - Teoria dos compiladores

Geração de Código

Grupo

André Dias Nunes — MAT: 201665570C

Guilherme Barbosa — MAT: 201435031

Leonardo Reis

Relatório do segundo trabalho da
disciplina DCC045 — Teoria dos
Compiladores. Desenvolvimento de
um Gerador de Código.

Juiz de Fora

Julho de 2022

Sumário

1	Introdução	2
2	Desenvolvimento	2
2.1	Geração Código	2
2.2	Estrutura	2
2.3	Construção do Template	3
2.3.1	Função	3
2.3.2	Data	4
2.3.3	Iterate	5
2.3.4	Chamada de Retorno	5
2.3.5	Chamada de Comando	6
3	Execução	7

1 Introdução

Neste relatório será apresentado a estratégia e a ferramenta utilizada para a criação do gerador de código para a linguagem **JavaScript**, o qual foi construído com o objetivo de contemplar a execução do quinto trabalho de Teoria dos Compiladores.

2 Desenvolvimento

2.1 Geração Código

Para a presente atividade foi escolhido criar uma geração de código em JavaScript. Esta linguagem foi escolhida pois a equipe trabalha em uma empresa que tem como principal ferramenta de desenvolvimento a linguagem mencionada acima.

Para a geração de código em de JavaScript foi utilizado a ferramenta StringTemplate(desenvolvida pelo mesmo criador da Antlr). Sendo assim foi criado um novo package denominado “template”. Dentro deste novo packge temos o arquivo “js.stg” que possui as definições das expressões em JavaScript que equivalem aos comandos da linguagem lang.

Para a utilização do template foi necessário criar a classe JavaVisitor. As principais aspecto da classe em questão, é a criação da variável groupTemplat, que recebe o template “js.stg” e é usada para integrar as funções do template de acordo com a respectiva expressão do arquivo lang.

2.2 Estrutura

- A escrita da linguagem foi desenvolvida baseada na compilação do Node v10.19.0.
- Para receber um input via console, é necessário possuir a biblioteca "prompt-sync"que pode ser instalada pelo comando `npm install prompt-sync`.
- Como foi usado uma linguagem não tipada a declaração dos tipos de variáveis não foi necessária.
- As funções foram geradas a partir do comando “function” e para seus retornos, é construído um array de elementos a serem retornados.
- O tipo Data foi criado como sendo um objeto com n chaves com o valor inicial igual a null.
- A adição de valores em um variável Data é feita mediante o comando `nome_variavel.nome_chave = valor_adicionado`

- Para imprimir os resultado em tela foi utilizado o comando `process.stdout.write`, visto que o comando `console.log` possui quebra de linha por padrão.
- Os valores impressos tiveram que ser transformado em comando do tipo `string` através do comando `toString()`.
- A criação de arrays foi feita mediante uma simples delação `"x = []"`, sendo que graças à estrutura da linguagem JS não foi necessário informar o tamanho do array.
- Para representar o comando `iterate`, é construído um `for`, onde sua variável de contagem é denominada a partir de sua linha e coluna para evitar conflitos de declaração.
- O comando `new` não é necessário para o Js.
- Quando acontece um retorno do resultado da divisão de dois inteiros, o resultado é um `float`, não tratamos essa condição.

2.3 Construção do Template

2.3.1 Função

O comando de função no formulário foi denominado “func”. Ele espera receber os seguinte parâmetros:

- **name:** nome da função
- **params:** parâmetros da função
- **stmt :** comandos da função

```
func(name, params, stmt) ::= <<
function <name>(<params; separator=", ">) {
|   <stmt; separator="\n">
|
}
```

Figura 1: Função

Ao identificar uma função a classe `LangVisitor` chama a “func” do template passando os dados informados acima. Como regra os parâmetros de entrada serão separados pelo caracteres “,”, sendo que esta ação é realizada através do comando “separator”.

Os comandos da função serão adicionadas na mesma a partir da variável “params” que é definida como sendo uma lista do ST(StringTemplate)

2.3.2 Data

O comando de data no formulário foi denominado “data”. Ele espera receber os seguinte parâmetros:

- **name:** nome do data
- **decl:** variáveis do data

```
data(name, decl) ::= <<
function <name> () {
    return{
        <decl; separator="\n">
    }
}
```

Figura 2: Data

Ao identificar um data a classe LangVisitor chama a “data” do template passando os dados informados acima. As variáveis do data serão adicionadas na mesma a partir da variável “decl” que é definida como sendo uma lista de ST(StringTemplate). Vale ressaltar que as variáveis informada serão adicionadas como valor padrão null (Figura 6)

```
decl(name) ::= <<
'<name>': null,
```

Figura 3: Decl

2.3.3 Iterate

O comando Iterate formulário foi denominado “iterate”. Ele espera receber os seguinte parâmetros:

- **expr:** critério de inicialização do iterate
- **decl:** parâmetros do iterate
- **lc:** número da linha e coluna

Ao identificar um Iterate a classe LangVisitor chama a “iterate” do template passando os dados informados acima. Os parâmetros do Iterate serão adicionadas na mesma a partir da variável “decl” que é definida como sendo uma lista de ST(StringTemplate). As variáveis do usadas na regra do Iterate são definidas com o prefixo “it” concatenadas com o número da linha e coluna na qual foi identificado a expressão no arquivo de entrada. Vale ressaltar que o “iterate” foi criado de maneira decrescente, ou seja, partindo do maior valor (informação recebida no expr) para o primeiro valor maior que 0.

```
iterate(expr, stmt, lc) ::= <<
for(let it<lc> = <expr>; it<lc> > 0; it<lc>--) {
|   <stmt>
|
}
```

Figura 4: Iterate

2.3.4 Chamada de Retorno

A chamada de expressão foi denominada "call_expr". Ele espera receber os seguinte parâmetros:

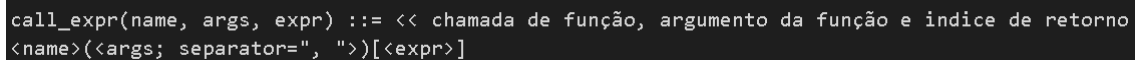
- **name:** nome da função
- **args:** argumento da função
- **expr:** índice de retorno

Ao identificar a chamada a uma função com a especificação de qual retorno se que ter acesso (Figura 5) a classe LangVisitor chama a "call_expr" do template passando os dados informados acima.



```
w = divMod(n, q)[1];
```

Figura 5: Exemplo de Chamada de Expressão da Linguagem Lang



```
call_expr(name, args, expr) ::= << chamada de função, argumento da função e índice de retorno  
<name>(<args; separator=", ">)[<expr>]
```

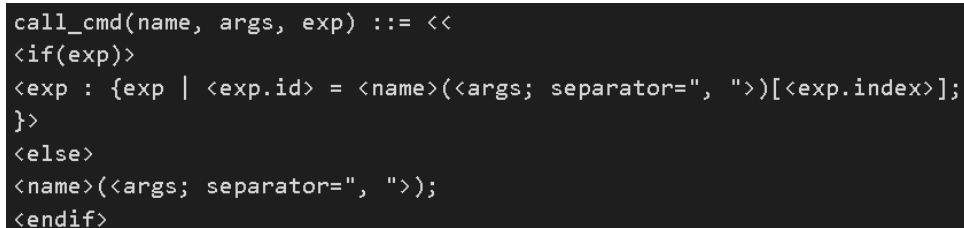
Figura 6: Chamada de Expressão

2.3.5 Chamada de Comando

A chamada de expressão foi denominada "call_cmd". Ele espera receber os seguintes parâmetros:

- **name:** nome da função
- **args:** argumento da função
- **expr:** nome das variáveis que irão receber o retorno

Ao identificar a chamada a uma função com a criação de variáveis que irão receber o retorno (Figura 8) a classe LangVisitor chama a "call_cmd" do template passando os dados informados acima.



```
call_cmd(name, args, exp) ::= <<  
<if(exp)>  
<exp : {exp | <exp.id> = <name>(<args; separator=", ">)[<exp.index>];  
>  
<else>  
<name>(<args; separator=", ">);  
<endif>
```

Figura 7: Chamada de Comando



```
divMod(n, q) < quo, res >;
```

Figura 8: Exemplo de Chamada de Comando da Linguagem Lang

3 Execução

Para o trabalho final, o código gerado foi feito em JavaScript, para sua compilação foi utilizado o Node v10.19.0 e uma biblioteca para ler o input do console, chamada "prompt-sync". Para tanto, certifique de realizar a instalação dos mesmos para testá-los, utilizando os comandos:

- `sudo apt install nodejs`
- `sudo apt install npm`
- A seguir execute a instalação da biblioteca no diretório "code/lang/code_js".
- `npm install prompt-sync`

Para a execução do programa foi criado um arquivo makefile, certifique-se de estar dentro do diretório lang para chamá-lo. Use os seguinte comandos:

- **make:** utilizado para realizar a criação das classes do ANTLR, a partir da gramática definida no arquivo lang.g4 (localizada no diretório parser), além da compilação das classes do projeto.
- **make run cmd=-bs** Executa uma bateria de testes sintáticos, referentes ao trabalho 2.
- **make run cmd=-bsm** Executa uma bateria de testes no interpretador, referentes ao trabalho 3, mas agora a verificação do sistemas de tipos foi acrescentada.
- **make run cmd=-bty** Executa uma bateria de testes no sistemas de tipos, referentes ao trabalho 4.
- **make run cmd=-bgc** Executa todos os testes anteriores e gera os códigos em JavaScript, referentes ao trabalho 5. Não havia uma sigla para a execução do trabalho 5 no arquivo "LangCompiler.java", então adicionamos este novo comando.
- **make clean** utilizado para realizar a limpeza dos arquivos gerados pelo ANTLR e dos .class gerados.
- **make all cmd=-bgc** Compila, executa, faz a limpeza e executa os códigos gerados em JavaScript. Para isso é necessário possuir o Node instalado.