UNIVERSIDADE FEDERAL DE ALAGOAS-UFAL
CAMPUS A.C. SIMÕES
CIÊNCIA DA COMPUTAÇÃO

CAIO BARBOSA VIEIRA DA SILVA

GH - UMA LINGUAGEM DE EXPLORAÇÃO DE DADOS NO GITHUB

MACEIÓ
2018.1

Caio Barbosa Vieira da Silva

GH - Uma Linguagem de Exploração de Dados no GitHub

Trabalho desenvolvido sob a orientação do professor Alcino Dall'Igna, como requisito parcial para obtenção de nota na disciplina Compiladores do curso de Ciência da Computação da Universidade Federal de Alagoas.

Maceió
2018.1

SUMÁRIO

1 Informações Gerais

1.1 INTRODUÇÃO

1.1.1 Definição da Linguagem

A Linguagem GH tem como objetivo facilitar a vida de pesquisadores que fazem Mineração de Dados no GitHub. GH permite a fácil manipulação dos tipos da API, transformando-os em objetos manipuláveis em uma linguagem de programação.

Um programa na linguagem GH é composto por funções. A função inicial (mestre) do programa é a função ***main***.

```
int main(){
    comandos;
    return 0;
}
```

1.1.2 Variáveis

1.1.2.1 Declaração

Uma variável na Linguagem GH é declarada seguindo a sintaxe:

```
<tipo> <nome_da_variavel>;
```

Elas são nomeadas através de identificadores (nomes), iniciados por uma letra, maiúscula ou minúscula (visto que a linguagem não é *case-sensitive*), podendo ser seguidos de letras, números ou o símbolo *underscore* ('_') a partir do segundo caracter, podendo ter tamanho máximo de 16 caracteres.

O fim do comando de declaração deve ser feito com o símbolo terminal ';' ou através de uma atribuição (1.1.2.2).

Alguns exemplos de declaração são:

```
int num;
string nome;
commit commit1;
issue issue1;
date feriado;
repository meu_repositorio;
```

1.1.2.2 Inicialização

A inicialização das variáveis é feita utilizando o símbolo **<-**, como exemplo:

```
int num <- 2;
```

1.1.3 Escopo

Os escopos das variáveis na linguagem GH são locais, existindo apenas dentro da função em que foram declaradas.

O escopo das funções é delimitado pelos símbolos '{' e '}'. E o terminador de comandos é o símbolo ';'.

As palavras reservadas são nomes exclusivos para os Tipos (1.1.6), Operadores (2), Instruções (3) e Funções da Linguagem (4).

O acesso aos campos das variáveis (tipos da API (1.1.7)) é feito através do símbolo **->**, como exemplo:

```
commit1 -> id;
```

A Linguagem GH não admite mascaramento de escopo.

1.1.4 Declaração de Funções

As funções devem ser declaradas fora do escopo do ***main*** e tem escopo global, não podendo ser operação de nenhum tipo (os tipos são finais).

Sua sintaxe de declaração é:

```
<tipo> <nome>(<tipo2> <param>, <tipo3> <param2>, ..){
    comandos;
    return <variavel do tipo do retorno>
}
```

Alguns pontos importantes sobre a sintaxe:

1. A quantidade de parâmetros é definida pelo programador.
2. O retorno das funções é feito utilizando a palavra reservada ***return*** seguido da variável a ser retornada e o terminador de comando ';'.
3. O return não é obrigatório caso o retorno da função seja void.

Um exemplo de declaração de função:

```
bool check_equal(int x, int y){
    return x equal y;
}

int sum(int x, int y, int z){
    return x + y + z;
}
```

A sintaxe de chamada das funções declaradas pelo usuários:

1. Retorno Void

```
nome_da_func(param1, param2);
```

2. Retorno com tipo

```
<tipo1> var;
var <- nome_da_func(param1, param2);
```

1.1.5 Coerção

A linguagem GH provê coerção das seguintes formas:

1. Ao concatenar um tipo à uma string com o operador de concatenação **+=**, esse tipo é transformado no tipo string de maneira automática.

2. O tipo de variável alvo de uma expressão dita o tipo da expressão. Ex:

```
int x <- 5;
int y <- 2;
float num <- x / y;
```

No exemplo, a divisão irá ser feita de maneira com que o resultado gerado seja um float, pois é o tipo da variável alvo.

1.1.6 Tipos de Dados:

1.1.6.1 bool

Descrição:

1. Possui dois valores representados por true e false.

2. O valor *default*(null) do bool é false.

Operações:

1. Utilizado em operações de negação, conjunção ou disjunção.

1.1.6.2 int

Descrição:

1. Utilizado para armazenar números inteiros com tamanho máximo de 32 bits.
2. O valor *default*(null) do int é 0.

Operações:

1. Soma, Subtração, Multiplicação, Divisão e Resto.

1.1.6.3 float

Descrição:

1. Utilizado para armazenar números de ponto flutuante. A parte inteira é separada da parte fracionária por um '.'.
2. O valor *default*(null) do float é 0.0.

Operações:

1. Soma, Subtração, Multiplicação, Divisão e Resto.

1.1.6.4 string

Descrição:

1. Representação de um caractere ou conjunto de caracteres.
2. O valor *default*(null) da string é uma string vazia .

Operações:

1. Comparação através da função ***compare***.
2. Concatenação através do operador de concatenação (**+=**).
3. Converter de maiúscula para minúscula e vice-versa através das funções da linguagem ***upper*** e ***lower***.

4. Busca de substrings através da função da linguagem ***find***.

5. Split da string através da função da linguagem ***split***.

6. Tamanho da string através da função da linguagem ***length***.

1.1.7 Tipos Associados à GitHub API

Descrição:

1. Representação em forma de Objeto do JSON retornado pela GitHub API[1] para a endpoint do tipo respectivo.

Campos:

1. Os campos são informações que vem da API para serem manipuladas de maneira read-only. (Não é possível modificar a informação, apenas operações de leitura).

1.1.7.1 user

a) login (string)

b) id (int)

c) type (string)

d) name (string)

e) company (string)

f) email (string)

g) public-repos (int)

h) followers (int)

i) following (int)

j) created-at (date)

[1] https://developer.github.com/v3/

1.1.7.2 repository

a) owner (string)

b) id (int)

c) name (string)

d) full-name (string)

e) description (string)

f) private (boolean)

g) fork (boolean)

h) url (string)

i) language (strings)

j) commit-list (commits)

1.1.7.3 commit

a) sha (string)

b) author-name (string)

c) author-login (string)

d) author-name (string)

e) author-date (string)

f) commiter-name (string)

g) commiter-login (string)

h) commiter-email (string)

i) commiter-date (string)

j) additions (int)

k) deletions (int)

l) file-list (files)

1.1.7.4 file

a) filename (string)

b) additions (int)

c) deletions (int)

d) patch (string)

1.1.7.5 issue

a) id (int)

b) title (string)

c) state (string)

d) url (string)

e) number (int)

f) body (string)

g) author-login (string)

h) comments (comments)

i) created-at (date)

j) closed-at (date)

k) closed-by (string)

1.1.7.6 pullrequest

a) id (int)

b) title (string)

c) state (string)

d) url (string)

e) number (int)

f) body (string)

g) author-login (string)

h) comments (comments)

i) created-at (date)

j) merged-at (date)

k) merged-by (string)

l) merge-commit

m) commits (int)

n) additions (int)

o) deletions (int)

p) changed-files (int)

1.1.7.7 comment

a) url (string)

b) id (int)

c) body (string)

d) position (int)

e) commit-hash (string)

f) author-login (string)

g) created-at (date)

h) updated-at (date)

1.1.7.8 date

ISO 8601 timestamp com informação de fuso.

a) day (string)

b) month (string)

c) year (string)

d) hour (string)

e) minute (string)

f) second (string)

g) timezone (string)

h) date (string)

1.1.7.9 Operações dos Tipos da API:

1. Comparação através da função ***compare***.
2. Atualização (buscar dados na API novamente) através da função ***refresh***.
3. Subtração (para o tipo ***date*** através da função ***diff_dates***).

1.1.8 Listas

1. São representados em linguagem natural pelo plural do seu tipo (e.g. strings).
2. Nunca são nulos, ao ser declarado, ele é instanciado com tamanho zero, sem necessidade de atribuição.
3. Não possuem campos. Para conseguir informações sobre as Listas deve-se utilizar as funções da linguagem, como por exemplo: ***length***.

1.1.8.1 Operações:

As operações suportadas pela pelas Listas são:

1. Ordenação através das funções ***sortAsc*** e ***sortDesc***.
2. Acesso aos elementos de forma única através do ***foreach***

3. Adição de elementos através da função ***add***
4. Remoção de elementos através da função ***remove***
5. Retorno do tamanho através da função ***length***
6. Comparação, através da função ***compare***.

2 Operadores

2.1 INFORMAÇÕES SOBRE OS OPERADORES

1. A tabela 7 lista as regras de precedência e de associatividade dos operadores.
2. Operadores na mesma linha tem a mesma ordem de precedência.
3. A tabela 7 está em ordem decrescente de precedência (e.g. * e / tem a mesma ordem de precedência, porém precedência maior que + e -).
4. Os operadores '++' e '- -' não são associativos.

Tabela 1 – Aritméticos

Operador	**Significado**
+	Adição
-	Subtração
*	Multiplicação
/	Divisão
%	Resto

Tabela 2 – Relacionais

Operador	**Significado**
greater	Maior que
less	Menor que
is	Igual a
isnot	Diferente de
greater-equal	Maior ou Igual
less-equal	Menor ou Igual

Tabela 3 – Lógicos

Operador	**Significado**
not	Negação
and	Conjunção
or	Disjunção

Tabela 4 – Concatenação

Operador	**Significado**
+=	Concatenação

Tabela 5 – Atribuição

Operador	**Significado**
<-	Atribuição

Tabela 6 – Acesso aos Campos

Operador	**Significado**
->	Acesso aos Campos

Tabela 7 – Precedência

Operador	**Associatividade**
++ e - -	direita para esquerda
( )	esquerda para direita
* / %	esquerda para direita
+ -	esquerda para direita
greater greater-equal less less-equal	esquerda para direita
is isnot	esquerda para direita
and	esquerda para direita
or	esquerda para direita

3 Instruções

3.1 ESTRUTURA CONDICIONAL DE UMA VIA

```
if (<logic expr>) {
    comando;
}
```

3.2 ESTRUTURA CONDICIONAL DE DUAS VIAS

```
if (<logic expr>) {
    comando;
} else {
    comando;
}
```

3.3 ESTRUTURA ITERATIVA

```
foreach <element> in <list> {
    comando;
}
```

3.4 ESTRUTURA ITERATIVA DE CONTROLE LÓGICO

```
while (<logic expr>) {
    comando;
}
```

3.5 ATRIBUIÇÃO

A atribuição é um comando, não sendo permitido sua utilização no meio de expressões.

4 Funções da Linguagem

4.1 ADD

A função ***add*** é uma função para do tipo Lista, e adiciona um elemento à lista.

Sua sintaxe de uso é:

```
lista_de_commits -> add(meu_commit);
```

4.2 REMOVE

A função ***remove*** é uma função para do tipo Lista, e remove um elemento da lista, se o elemento existir.

Sua sintaxe de uso é:

```
lista_de_commits -> remove(meu_commit);
```

4.3 SORTASC

A função ***sortAsc*** é uma função para do tipo Lista, e ordena os elementos de maneira crescente. Caso a lista não seja de inteiros, floats ou dates, é necessário um parâmetro para explicitar o campo que deve ser utilizado na ordenação.

Sua sintaxe de uso é:

```
lista_de_commits -> sortAsc("id");
lista_de_strings -> sortAsc();
```

4.4 SORTDESC

A função ***sortDesc*** é uma função para do tipo Lista, e ordena os elementos de maneira decrescente. Caso a lista não seja de inteiros, floats ou dates, é necessário um parâmetro para explicitar o campo que deve ser utilizado na ordenação.

Sua sintaxe de uso é:

```
lista_de_comments -> sortAsc("position");
lista_de_strings -> sortDesc();
```

4.5 FIND

A função ***find*** é uma função para do tipo string, e verifica se existe a substring (passada como parâmetro) dentro da string. O retorno da função é um bool.

Sua sintaxe de uso é:

```
minha_string -> find("-");
```

4.6 UPPER

A função ***upper*** é uma função para do tipo string, e passa todos os caracteres da string para maiúsculo.

Sua sintaxe de uso é:

```
minha_string -> upper();
```

4.7 LOWER

A função ***lower*** é uma função para do tipo string, e passa todos os caracteres da string para minúsculo.

Sua sintaxe de uso é:

```
minha_string -> lower();
```

4.8 COLLECT_DATA

A função ***collect_data*** é uma função para do tipo repository, ela é responsável por coletar informações do repositório na Github API.

Sua sintaxe de uso é:

```
repository meu_projeto -> collect_data("project_name");
```

4.9 REFRESH

A função ***refresh*** é uma função dos tipos associados à API, ela é responsável por atualizar os dados (fazer uma nova consulta à API) do tipo que chamar essa função.

Sua sintaxe de uso é:

```
meu_dado_da_api -> refresh();
```

4.10 DIFF_DATES

A função ***diff_dates*** é do tipo dates, e contabiliza a quantidade de dias entre a date atual (variável chamando a função) e a data passada como parâmetro, retornando um inteiro representando a quantidade de dias.

Sua sintaxe de uso é:

```
date hoje <- "24/10/2018";
date amanha <- "25/10/2018";

int days <- hoje -> diff_dates(amanha);
```

4.11 LENGTH

A função ***length*** é dos tipos Lista e string, e retorna o tamanho da Lista ou da string.

Sua sintaxe de uso é:

```
int tam <- minha_string -> length();
int tam2 <- minha_lista -> length();
```

4.12 COMPARE

A função ***compare*** serve para todos os tipos. Ela verifica se os elementos comparados (ou listas) são idênticos. No caso das listas, ele compara elemento à elemento, com a ordem importando. Retornando um bool.

Sua sintaxe de uso é:

```
commits meus_commits1 <- meu_repositorio1 -> commit_list;

commits meus_commits2 <- meu_repositorio2 -> commit_list;

bool check <- meus_commits1 -> compare(meus_commits2);
```

4.13 SPLIT

A função ***refresh*** é uma função do tipo string, ela separa a string e transforma numa lista strings, a separação ocorre nos pontos onde existe determinada string (passado por parâmetro).

Sua sintaxe de uso é:

```
strings lista_de_strings;
lista_de_strings <- minha_string -> split(",");
```

4.14 REFRESH

A função ***refresh*** é uma função dos tipos associados à API, ela é responsável por atualizar os dados (fazer uma nova consulta à API) do tipo que chamar essa função.

Sua sintaxe de uso é:

```
meu_dado_da_api -> refresh();
```

4.15 READ

A função ***read*** é uma função global, não associada a nenhum tipo. Ela tem como objetivo receber a entrada do usuário e atribuir à uma variável.

Sua sintaxe de uso é:

```
string input <- read();
```

4.16 PRINT

A função ***print*** é uma função global, não associada a nenhum tipo. Ela tem como objetivo imprimir dados na tela.

Sua sintaxe de uso é:

```
print("Ola mundo!");
```

5 Exemplos

5.1 OLÁ MUNDO

```
int main() {
    print("Hello world");
}
```

5.2 VERIFICANDO CAMPOS DOS TIPOS.

```
int main() {

    string project_name <- "project_name";
    repository github_project -> collect_data(project_name);
    commits project_commits <- github_project -> commit_list;

    foreach commit_1 in project_commits {
        if (commit_1.id is 2) {
            print("This is the second commit
                of the project: " + project_name);
        }
    }
}
```

5.3 CHECANDO SE DUAS LISTAS POSSUEM ELEMENTOS EM COMUM

```
int main(){

    repository tomcat <- collect_data("tomcat");
    repository derby <- collect_data("derby");
    commits commits_project1;
    commits commits_project2;

    commits_project1 <- tomcat -> commit_list;
    commits_project2 <- derby -> commit_list;

    print("Existem " + commits_project1 -> length() +
        " commits no projeto Tomcat");
    print("Existem " + commits_project2 -> length() +
        " commits no projeto Derby");
```

```
    strings tomcat_authors;

    foreach commit_tomcat in commits_project1 {
        tomcat_authors -> add(commit_tomcat -> author_login);
    }

    strings tomcat_authors;

    foreach commit_derby in commits_project2 {
        derby_authors -> add(commit_derby -> author_login);
    }

    foreach author_tomcat in tomcat_authors {
        foreach author_derby in derby_authors {
            if (author_tomcat -> compare(author_derby)){
                print("O usuario " + author_tomcat +
                    " tem commit no Derby e no Tomcat.\n");
            }
        }
    }
}
```

6 Tokens

A linguagem Java será utilizada para a implementação dos analisadores léxicos e sintáticos da Linguagem GH.

6.1 ENUMERAÇÃO DOS TOKENS

```
public enum GHTokens{

    main, id, tVoid, tInt, tInts, tFloat,
    tFloats, tString, tStrings,
    tBool, tBools, tCommit, tCommits,
    tUser, tUsers, tRepository,
    tRepositories, tFile, tFiles, tIssue,
    tIssues, tPullRequest,
    tPullRequests, tComment, tComments,
    tDate, tDates, escStart,
    escEnd, paramStart, paramEnd, cmt,
    term, sep1, sep2, prRead,
    prWrite, prIf, prElse, prWhile, prFor,
    prReturn, opAtrib, opIncre,
    opLogicAnd, opLogicOr, opLogicNeg,
    opAritAd, opAritMul, opNegUn,
    opRel1, opRel2, opRel3, opConc;

}
```

6.2 EXPRESSÕES REGULARES AUXILIARES

```
letra = '[:alpha:]';

digito = '[:digit:]';
```

6.3 LEXEMAS:

6.3.1 Main:

```
main = 'main';
```

6.3.2 Identificador:

```
id = ['letra'] ['letra' | '_' | 'digito']*;
```

6.3.3 Tipos Primitivos:

```
tVoid = 'void';
tInt = 'int';
tInts = 'ints';
tFloat = 'float';
tFloats = 'floats';
tString = 'string';
tStrings = 'strings';
tBool = 'bool';
tBools = 'bools';
tCommit = 'commit';
tCommits = 'commits';
tUser = 'user';
tUsers = 'users';
tRepository = 'repository';
tRepositories = 'repositories';
tFile = 'file';
tFiles = 'files';
tIssue = 'issue';
tIssues = 'issues';
tPullRequest = 'pullrequest';
tPullRequests = 'pullrequests';
tComment = 'comment';
tComments = 'comments';
tDate = 'date';
tDates = 'dates';
```

6.3.4 Funções:

```
fAdd = 'add';
fRem = 'remove';
fSortAsc = 'sortAsc';
fSortDesc = 'sortDesc';
fUpper = 'upper';
fLower = 'lower';
fCollect = 'collect_data';
fRefresh = 'refresh';
fDiff = 'diff_dates';
fLength = 'length';
fCompare = 'compare';
```

```
fFind = 'find';
```

6.4 DELIMITADORES:

6.4.1 Escopo:

```
escStart = '{';
escEnd = '}';
```

6.4.2 Parâmetros:

```
paramStart = '(';
paramEnd = ')';
```

6.4.3 Comentários:

```
cmt = '/#';
```

6.4.4 Terminador:

```
term = ';'
```

6.4.5 Separador:

```
sepFloat = '.';
sepFor = 'in';
```

6.5 PALAVRAS RESERVADAS:

6.5.1 Entrada e Saida:

```
prRead = 'read';
prWrite = 'print';
```

6.5.2 Iteração ou Seleção:

```
prIf = 'if';
prElse = 'else';
prWhile = 'while';
prFor = 'foreach';
```

6.5.3 Retorno:

```
prReturn = 'return';
```

6.6 OPERADORES:

6.6.1 Atribuição:

```
opAtrib = '<-';
opIncre = '++' | '--';
```

6.6.2 Lógicos:

```
opLogicAnd = 'and';
opLogicOr = 'or';
opLogicNeg = 'not'
```

6.6.3 Aritméticos:

```
opAritAd = '+' | '-';
opAritMul = '*' | '/' | '%";
```

6.6.4 Relacionais:

```
opComp = 'greater'|'less';
opComE = 'greater-equal'|'less-equal';
opLogicExp = 'is' | 'isnot';
```

6.6.5 Concatenação:

```
opConc = '+=';
```