Escolha do Tema

Trabalho de Tradutores

Wladimir Gramacho - 15/0048718

Profª. Cláudia Nalon

Agosto de 2019

1 Objetivos

O curso de Tradutores ministrado na Universidade de Brasília tem como objetivo o estudo dos componentes de um tradutor assim como a implementação destes[1]. Sendo assim, o trabalho desta matéria consiste na construção de um tradutor para uma linguagem de programação simplificada e de propósito específico. O desenvolvimento do projeto se dará em 6 etapas diferentes, sendo elas: Escolha do Tema, Analisador Léxico, Analisador Sintático, Analisador Semântico, Gerador de Código Intermediário e Apresentação do Trabalho. Este relatório se trata da apresentação da primeira fase: Escolha do Tema.

2 Motivação

Estamos na Era da Informação [2], na qual a democratização do acesso à informação fez com que a criação e divulgação de informação seja constante e crescente [4]. Devido ao volume de produção e armazenamento desses conteúdos [5], mecanismos de busca e manipulação são necessários para que usuários comuns possam encontrar informações relevantes [3]. Um dos formatos mais utilizados para a difusão de informações é texto, de forma que conseguir lidar com tais dados pode auxiliar o trabalho não só de desenvolvedores mas também de produtores de conteúdo.

Manipulação de texto, ou manipulação de *strings*, é um conjunto de ferramentas para a execução de operações em cadeias de caracteres. Em várias linguagens de programação, o desenvolvedor tem uma série de funções nativas da linguagem que o ajudam nas tarefas com *strings*. Por exemplo, na linguagem `Ruby`[2], o programador pode facilmente fazer interpolação de *strings*, além de criar *strings* de tamanho dinâmico. Outra coisa que é possível fazer em `Ruby` é remover todas as ocorrências de uma *substring* de um *string*. Isso pode ser bastante útil na limpeza de conteúdos no formato `HTML`, por exemplo.

[1] https://matriculaweb.unb.br/graduacao/disciplina.aspx?cod=116459
[2] https://ruby-doc.org/core-2.6/String.html

Na linguagem de programação `C`, o desenvolvedor não tem todas essas facilidades. Para executar tais operações, é necessário que ele crie suas próprias funções. Seria interessante ter um tipo de dados para cadeias de caracteres de tamanho indefinido e essas funcionalidades implementadas nativamente na linguagem, pois permitiria operações de raspagem de dados em textos obtidos pela Web, seguindo o mesmo exemplo do parágrafo anterior.

3 Proposta de Projeto

A proposta deste projeto é a de implementar um tradutor para uma versão reduzida da linguagem de programação `C` com suporte para *strings* de tamanho dinâmico, fazendo as operações de interpolação, busca e remoção de *substrings*. Além dessas funções, o tradutor terá suporte para operações com inteiros e números de ponto flutuante, operações de controle condicional de fluxo e laços de repetição, comandos de leitura e escrita e chamadas de subrotinas. Um exemplo de um programa dessa linguagem seria:

```
string html_string;
string chapter_title;
string result_str;
int pages_book;
int pages_read;

html_string = "<h1>You still have to read</h1>";
pages_book = 120;
pages_read = 75;
chapter_title = "#{pages_book - pages_read} pages!";

strrmv(html_string, "<h1>");
strrmv(html_string, "</h1>");
result_str = "#{html_string} #{chapter_title}";
print(result_str);
```

```
>> You still have to read 45 pages!
```

Alguns pontos de atenção na implementação desse trabalho são: remoção *inplace* (otimizada) de *substrings*, busca de *substring* otimizada, interpolação de *strings* com expressões aritméticas e manipulação de memória dinâmica. Basicamente, as operações com o novo tipo de dados **string** devem se preocupar com desempenho em termos de tempo e de espaço.

4 Linguagem Formal

Uma forma muito utilizada de especificar sintaxe de uma linguagem é a de construir uma gramática livre de contexto [1]. A gramática da versão simplificada do `C` que será implementada neste trabalho tem a seguinte definição:

1. *program* → *declaration-list*

2. *declaration-list* → *declaration-list declaration* | *declaration*
3. *declaration* → *var-declaration* | *fun-declaration*
4. *var-declaration* → *type* **ID ;** | *type* **ID [NUM] ;**
5. *type* → **int** | **string** | **void**
6. *fun-declaration* → *type* **ID (** *params* **)** *compound-stmt*
7. *params* → *param-list* | **void**
8. *param-list* → *param-list* **,** *param* | *param*
9. *param* → *type* **ID** | *type* **ID [ ]**
10. *compound-stmt* → **{** *local-declarations stmt-list* **}**
11. *local-declarations* → *local-declarations var-declaration* | ε
12. *stmt-list* → *stmt-list stmt* | ε
13. *stmt* → *expression-stmt* | *conditional-stmt* | *iteration-stmt* | *return-stmt*
14. *expression-stmt* → *expression* **;**
15. *conditional-stmt* → **if (** *expression* **)** *compound-stmt* |
    if (*expression* **)** *compound-stmt* **else** *compound-stmt*
16. *iteration-stmt* → **while (** *expression* **)** *compound-stmt*
17. *return-stmt* → **return** *expression* **;** | **return ;**
18. *expression* → *var* **=** *expression* | *simple-expression*
19. *var* → **ID** | **ID [** *expression* **]**
20. *simple-expression* → *op-expression relop op-expression* | *op-expression*
21. *relop* → **<=** | **<** | **>** | **>=** | **==** | **!=**
22. *op-expression* → *op-expression addop term* | *term*
23. *addop* → **+** | **-**
24. *term* → *term mulop factor* | *factor*
25. *mulop* → ***** | **/**
26. *factor* → **(** *expression* **)** | *var* | *call* | **“** *string* **”** | **NUM**
27. *call* → **ID (** *args* **)**

28. $string \rightarrow string$ **STRING** | $string$ **#{** $expression$ **}** | ε

29. $args \rightarrow arg\text{-}list \mid \varepsilon$

30. $arg\text{-}list \rightarrow arg\text{-}list$ **,** $expression \mid expression$

ID = $letter\ (letter|digit)^*$

NUM = $digit\ digit^*$

STRING = $(letter|digit|punctuation|whitespace)^*$

$letter = a \mid ... \mid z \mid A \mid ... \mid Z$

$digit = 0 \mid ... \mid 9$

$punctuation =$, | . | ; | : | − | + | = | ∗ | ? | ! |
(|) | [|] | < | > | \ | /

$whitespace =$ \n | | \ t

Palavras reservadas: **int string void if else while return**
Símbolos especiais: + - * / < <= > >= == != = , ; ( ) [ ] { } # “ ”

Referências

[1] Alfred V Aho, Ravi Sethi, and Jeffrey D Ullman. Compilers, principles, techniques. *Addison wesley*, 7(8):9, 1986.

[2] Manuel Castells. The information age: Economy, society and culture (3 volumes). *Blackwell, Oxford*, 1997:1998, 1996.

[3] Michael Gordon and Praveen Pathak. Finding information on the world wide web: the retrieval effectiveness of search engines. *Information Processing & Management*, 35(2):141–180, 1999.

[4] Eszter Hargittai and Gina Walejko. The participation divide: Content creation and sharing in the digital age. *Information, Community and Society*, 11(2):239–256, 2008.

[5] Martin Hilbert and Priscila López. The world's technological capacity to store, communicate, and compute information. *science*, 332(6025):60–65, 2011.