Conceitos de Linguagens de Programação
MAC5754
Relatório das fases 1 e 2

Alantiel Freire Marins
Bárbara Tieko Agena
Rayssa Küllian Martins
Yuri David Santos

10 de Abril de 2014

Resumo

Nas fases 1 e 2, iremos apresentar o relatório do andamento do projeto de uma linguagem de programação, aqui se encontram a descrição do domínio e os elementos essenciais da nossa linguagem.

A seguir iremos descrever a futura linguagem, seu nome será MarkIME, o motivo deste nome se explica pelo foco da linguagem, pois a linguagem, como será visto, é uma facilitadora para a escrita de LaTeX, que é uma linguagem de markup, linguagem muito utilizada no IME para apresentação de trabalhos e artigos.

Conteúdo

1 Fase 1 . . . 2
1.1 Descrição do Domínio . . . 2
1.2 Proposta da Linguagem de Programação . . . 2
Características da linguagem: . . . 2
1.3 Elementos essenciais à linguagem . . . 2
2 Fase 2 . . . 3
2.1 Tipos de Dados . . . 3
2.1.1 Numéricos . . . 3
2.1.2 Não numéricos . . . 3
2.2 Expressões . . . 3
2.2.1 Literais . . . 3
2.2.2 Aplicação de funções . . . 4
2.2.3 Valores associados a identificadores . . . 4
2.3 Avaliação de expressões . . . 4
2.3.1 Ordem de avaliação . . . 4
2.4 Comandos . . . 4
2.4.1 Comandos de atribuição . . . 4
2.4.2 Blocos . . . 4
2.4.3 Condicionais . . . 5
2.4.4 Comandos de iteração . . . 5
2.5 Forma e Tempo de Vinculação de Tipos às Variáveis . . . 5
2.5.1 Forma de Vinculação . . . 5
2.5.2 Tempo de Vinculação . . . 5
2.6 Sistemas de Tipos . . . 5
2.6.1 Verificação de Tipos . . . 5
2.6.2 Erros de Tipos . . . 6
2.7 Esboço de um artigo escrito em MarkIME . . . 7

1 Fase 1

Nessa primeira fase será feito o levantamento dos requisitos do domínio da aplicação. Na seção 1.1 será apresentada a descrição do domínio; na seção 1.2, a proposta da linguagem de programação; e na seção 1.3, os elementos essenciais da linguagem.

1.1 Descrição do Domínio

Escrever código em LaTeXgeralmente resulta em documentos de alta qualidade (em termos de formatação), poupando muito trabalho redundante e evitando conhecimentos específicos de operação de softwares processadores de texto cujas interfaces estão sujeitas a mudança frequente. Apesar das vantagens, essa pode ser uma tarefa árdua. LaTeXnão tem uma sintaxe amigável, impondo grande trabalho e a necessidade de uma gama de conhecimentos específicos para a implementação de tabelas, imagens e formatações específicas.

Entre os pesquisadores que desejam escrever artigos nem sempre existe o conhecimento de linguagens de programação, muitos alunos tem foco no seu campo de atuação, mas não em habilidades específicas de LaTeX.

Dessa forma, visamos facilitar o processo de escrita de artigos, relatórios e apresentações de slides em LateX em geral, evitando tarefas repetitivas, aumentando o poder de reutilização de código e criando um meio mais simples e direto de escrevê-los.

1.2 Proposta da Linguagem de Programação

Escrever uma linguagem que facilite a manipulação de artigos, relatórios e apresentações em LateX, adicionando flexibilidade ao processo de edição por meio de uma sintaxe mais simples e flexível, facilitando tarefas comuns ao contexto e embutindo novos recursos.

Características da linguagem:

- Facilitar tarefas como escape de caracteres especiais, acentuação, etc.
- Simplificar a construção de tabelas e inclusão de imagens, legendas, etc.
- Comando de repetições e condições inexistentes em LaTeX pura para fazer tratamentos úteis como condições sobre elementos do texto, em formatos iterativos em LaTeX.

1.3 Elementos essenciais à linguagem

1. **Operadores**:
    - **Operações sobre texto**: Concatenação, uppsercase, etc.
    - **Operações atitméticas**: Soma, multiplicação, etc.
    - **Operações lógicas**: E / OU
2. **Declarações**:
    - **Constantes**: a linguagem suportará constantes numéricas e de string

- **Variáveis**: a linguagem suportará variáveis numéricas e de string

3. **Funções**:
   - **Funções matemáticas**: para realizar operações sobre os números
   - **Funções para manipulação de texto**: para realizar operações sobre as strings, fazer tratamentos de case, busca, etc.

2 Fase 2

2.1 Tipos de Dados

A linguagem terá apenas tipo de dados primitivos, separamos em 2 grupos:

2.1.1 Numéricos

- **<número>**: 1, 2, -3, 42, 0.0, -0.5, etc... (Servirão tanto para inteiros como para reais)

2.1.2 Não numéricos

- **<string>**: , "Meu texto", etc...
- **<booleano>**: true, false
- **<data>** tipo para armazenar data.

2.2 Expressões

A linguagem terá 3 formas de expressões para os valores das variáveis:

2.2.1 Literais

Como o objetivo principal da linguagem é facilitar a construção de documentos em LaTeX, a maioria dos tipos terão representações literais. O tipo <string>é o mais importante, pois constitui a maior parte do conteúdo dos documentos.

Considerando essas necessidades, dois tipos de literal serão usados para <string>: delimitadas por aspas (“) ou ao estilo heredocs. As literais heredocs são usadas entre delimitadores especiais de início ({) e fim (}), que devem ser os únicos caracteres das linhas em que estiverem. Segue exemplo de uso de heredocs sendo passado para a função abstract:

```
abstract
{
  Lorem ipsum dolor sit amet, consectetur
  adipisicing elit, sed
  do eiusmod tempor incididunt ut labore et dolor
  e magna aliqua. Ut enim ad minim veniam,
  quis nostrud exercitation ullamco laboris nisi ut
  aliquip ex ea commodo consequat.
}
```

Dentro do escopo das heredocs poderão ser inseridas expressões da linguagem, que serão avaliadas e o resultado concatenado à <string>. Essas expressões deverão estar dentro de delimitadores especiais ($$ expressão $$). Outras literais serão usadas para o tipo <número>, por exemplo:

- 1
- 1.1
- 0.001
- -23.9

2.2.2 Aplicação de funções

- upper("minhaString") equivale a string "MINHASTRING"
- concat("minhaString", "minhaNovaString") equivale a string "minhaStringminhaNovaString"

2.2.3 Valores associados a identificadores

- var i = 1 equivale a definir o inteiro 1 em i
- var s = "texto"equivale a definir a string "texto"em s

2.3 Avaliação de expressões

2.3.1 Ordem de avaliação

Mesma ordem da avaliação matemática usual, isto é, da esquerda para a direita, com ordem de precedência: (), *, /, +, -

2.4 Comandos

A linguagem terá recursos extras para tratamento de string e números, esses recursos são providos na maioria dos casos por funções, tais como.

2.4.1 Comandos de atribuição

```
var x = y
```

2.4.2 Blocos

```
head [...]
var x
#document{
    var y = 2
    $$z = x + y$$
}
```

2.4.3 Condicionais

```
#head [...]
var x = 0
#document{
    var y = 2
    $$if(y < x){ y } else { x }$$  //mostra o menor
}
```

2.4.4 Comandos de iteração

```
#head [...]
var x = 5
#document{
    $$for(var y = 0; y < x; y++){ y }$$  //itera 5 vezes
}
```

2.5 Forma e Tempo de Vinculação de Tipos às Variáveis

Essa seção tratará da forma como é feita a vinculação de tipos, e do momento em que ela ocorre.

2.5.1 Forma de Vinculação

A vinculação será feita implicitamente, de acordo com o valor que for atribuído à variável. Por exemplo, dado o trecho de código:

```
x = 10
```

automaticamente será vinculado o tipo número à variável x.

2.5.2 Tempo de Vinculação

A vinculação de tipos é estática. A partir do momento que a primeira atribuição for feita à variável, o tipo é inferido, e a partir desse momento não será mais alterado. Atribuições de tipos diferentes são resolvidas através de conversão de tipos (type casting).

2.6 Sistemas de Tipos

Nesta seção serão descritas as regras sobre os tipos definidos dentro da linguagem de programação. É importante deixar claro as variáveis, expressões e funções.

O sistema de tipos auxilia na redução de erros permitindo uma execução mais consistente.

2.6.1 Verificação de Tipos

A linguagem será fracamente tipada, i.e., será permissiva quanto a operações entre tipos diferentes, fazendo type casting sempre que for possível, por exemplo, permitindo a concatenação com cast implícito entre <string> e <número>.

2.6.2 Erros de Tipos

A linguagem será permissiva quanto às atribuições de tipos. Sempre que uma atribuição entre tipos diferentes for feita, o processador tentará fazer um cast entre os tipos. Caso não seja possível, um erro será gerado.

Variáveis do tipo <string>poderão receber valores de quaisquer outros tipos, nesse caso será feito um cast e o valor resultante será uma representação textual do valor recebido (e.g. o número 10 será armazenado como a *string* "10").

Variáveis do tipo número poderão receber strings que contenham a representação textual de um número (e.g. a string "10"), caso no qual será feito um cast para obter o valor numérico representado. Caso recebam valores booleanos, o valor verdadeiro será convertido para 1, e $falso$ para 0. Outros tipos gerarão um erro de atribuição incorreta de tipo.

Variáveis do tipo <booleano>poderão receber quaisquer valores, sempre convertendo para *verdadeiro*, exceto nos casos em que receberem a string vazia ou o número 0, nos quais o valor final será $falso$.

2.7 Esboço de um artigo escrito em MarkIME

```
#head['Test article', 'Footnote',
      'http://test.com', 'Author1; Author2;Author3']

      var string = "string-test"
      var count = 1

#abstract{
   Nome do artigo   //just if want overhide the default
   ==============

   Article description, bla bla, bla
}

#document{
  Name of article    //just if want overhide the default
  ==============

  Section 1
  ---------

  Paragraph are separeted a blank line,
  a just break doesn't do anything.

  Text in *italic*,
  **bold**, 'monospaced'.

  Um [link](http://teste.com).

  Variables and methods
  $$string | concat(concat(string, string), "last")$$
  $count = $ $$count+1$$  //print count = 1
  $count = $ $$count+1$$  //print count = 1
  different of
  $$count+1$$             //print 2
  $$count = count+1$$     //print 2
  $$count = count+1$$     //print 3

  Unordered list:
    * list a
    * list g
    * list c

  Ordered list:
    1. list 1
    2. list 2
    3. list 3
}
```