

**UNIVERSIDADE FEDERAL DE SÃO CARLOS
CAMPUS SÃO CARLOS
LUCAS OLIVEIRA DAVID – 407917**

**TRABALHO DE LINGUAGENS FORMAIS E AUTOMATOS
RELATÓRIO DA ATIVIDADE 1**

SÃO CARLOS – SP
2012

1. INTRODUÇÃO

Esta atividade tem como objetivo a implementação de um algoritmo capaz de identificar padrões inválidos sobre uma sequência de caracteres (ou *strings*). A formatação desejada é a dos números decimais. Isto é, todas as cadeias de caracteres que não obedecerem o padrão da representação decimal são considerados “inválidos”, enquanto os demais, “válidos”.

1.2. Exemplos de padrões válidos:

- +2;
- 102;
- 1.000.000;
- 10.000,20345

1.3. Exemplos de padrões inválidos:

- +-2;
- 1.0,00.000;
- ,100;
- 45.0+--+54,00

2. LINGUAGEM DE PROGRAMAÇÃO E CONFIGURAÇÃO

Como linguagem, o projeto foi implementado utilizando C++ através da IDE Netbeans na plataforma linux Ubuntu. Está anexado, juntamente com este relatório, o arquivo projeto.rar, contendo o projeto Netbeans para verificação do código-fonte. Um executável para sistemas linux já se encontra compilado na pasta ./disk/Debug/GNU-Linux-x86/atividade1vs2. Para recompilar o projeto, os únicos três arquivos necessários são o **main.cpp**, **Interpretador.h** e **Interpretador.cpp**.

main.cpp: instancia o Interpretador e cria a interface para a entrada das sequencias.

Interpretador.h: header da classe Interpretador. Declara métodos e atributos referentes a esta classe.

Interpretador.cpp: declaração dos métodos interiores à classe Interpretador.

3. DESCRIÇÃO DO ALGORITMO

Enquanto o código contido no arquivo **main.cpp** se preocupa unicamente com o loop de entradas das sequências, é de responsabilidade dos objetos pertencentes a classe **Interpretador** a verificação da validade destas mesmas.

3.1. DESCRIÇÃO DOS ATRIBUTOS DE INTERPRETADOR

3.1.1. char SeqOriginal[32]

Armazena a sequência de símbolos originalmente inserida no objeto através do método **AtualizarSeq()**.

3.1.2. char SeqMapeada[32]

Após a inserção de uma *string* em SeqOriginal[], esta mesma cadeia é codificada e enfim copiada para SeqMapeada[].

3.1.3. int NumSeqInvalidas

Armazena o número de sub-sequências declaradas em char ****SeqInvalidas** que, quando se apresentam na SeqMapeada[], invalidam a cadeia SeqOriginal[].

3.1.4. char ****SeqInvalidas**

Lista de sub-cadeias consideradas inválidas na representação dos decimais. Exemplo:

- “NS”: uma sequência S apresenta essa sub-sequência se, e somente se, ela é inválida para a representação decimal por apresentar um ou mais sinais entre a representação da magnitude (SeqMapeada igual a “NNSN”, SeqOriginal igual a “10+2”).

- “NVP”: igualmente ao exemplo acima. A sequência S também é necessariamente inválida caso apresente “NVP” como subcadeia (SeqMapeada igual a “NNVNNP”, SeqOriginal igual a “14,45.”).

3.2. DESCRIÇÃO DOS MÉTODOS DE INTERPRETADOR

3.2.1. bool Interpretador::AtualizarSeq(char *pSeq)

Descrição: atualiza o atributo **SeqOriginal** (char [32]) do objeto pertencente a classe Interpretador.

Possíveis erros: a cadeia ***pSeq** (que é a que se deseja copiar a sequência) possui mais de 31 dígitos. Dado o tamanho de **SeqOriginal** (32 bytes), é impossível copiá-la e a função retorna **falso**.

Conclusão: método **MapearSeq()** é chamado e, quando finalizado, a função retorna o valor **verdadeiro**.

3.2.2. void Interpretador::MapearSeq()

SeqOriginal[] é codificada e transposta para **SeqMapeada[]**. Essa codificação é descrita pela tabela abaixo:

ENTRADA	SÍMBOLO CODIFICADO
Números de zero a nove	N
Sinais de positivo e negativo	S
Vírgulas	V

Pontos	P
Qualquer outro caracter	?

3.2.3. void Interpretador::DefSeqInvalidas()

Define e aloca um numero dinâmico de sequências inválidas. Posteriormente, cada uma destas sequências são copiadas para a posição alocada referente através da função **strcpy(destino, origem)**. Exemplo:

- `strcpy(SeqInvalidas[0], "NNNN");`
- `strcpy(SeqInvalidas[1], "NS");`

A primeira sequência acima (“NNNN”) representa invalidades como “1203.201232.102”, enquanto a segunda representa as equivalentes a “12+23-32”.

Alguns caracteres especiais podem ser colocados na sequência para indicar algumas condições de invalidez adicionais:

- **!** Utilizado no início e no fim das sub-sequências. Indica que aquela sub-cadeia deve estar posicionado no inicio ou fim da cadeia original para invalidá-la. Exemplo:
 - **!VN**: neste exemplo, a cadeia original começará com uma vírgula e terminará com um número, sendo assim, considerada INVÁLIDA (,2).
 - **NNNP!**: a cadeia original terminará com um ponto, e portanto, é INVÁLIDA (100.)
- ***** Utilizado para especificar que uma sub-cadeia de tamanho indeterminada PODE estar alocada entre as duas cadeias inválidas. Mas Independentemente disto, a simples ocorrência das duas (em ordem), invalida a sequência como um todo. Exemplo:
 - **V*P**: aqui, a cadeia original é composta por uma virgula e, posteriormente, um ponto. O que é caracterizado como inválido (10,203.023).
 - **NNNN*V**: a cadeia original é composta por quatro números sem separação por ponto. Posteriormente a virgula é apresentada (100.000.0000.000,00000000).

3.2.4. void Interpretador::ExibirSeq(int pIdSeq) {

Exibe a sequência representada em **SeqOriginal[]** quando pIdSeq é igual a **zero**. Para todos os outros casos, imprime a sequência representada em **SeqMapeada[]**. Exemplo:

- { [1] [2] [,] [3] }
- { [N] [N] [V] [N] }

3.2.5. void Interpretador::ExibirResultado()

Chama a função **AnalisarSeq()** e, dependendo do valor booleano retornado por essa mesma, informa se a sequência é válida ou não.

3.2.6. bool Interpretador::AnalisarSeq()

Este método define a validade (ou invalidade) de uma dada sequência **SeqOriginal[]** com base em uma busca na sequência **SeqMapeada[]** por padrões contidos na lista **SeqInvalidas[][]**, caracterizados como INVÁLIDOS. Retorna verdadeiro caso esta mesma seja válida ou falso caso contrário.

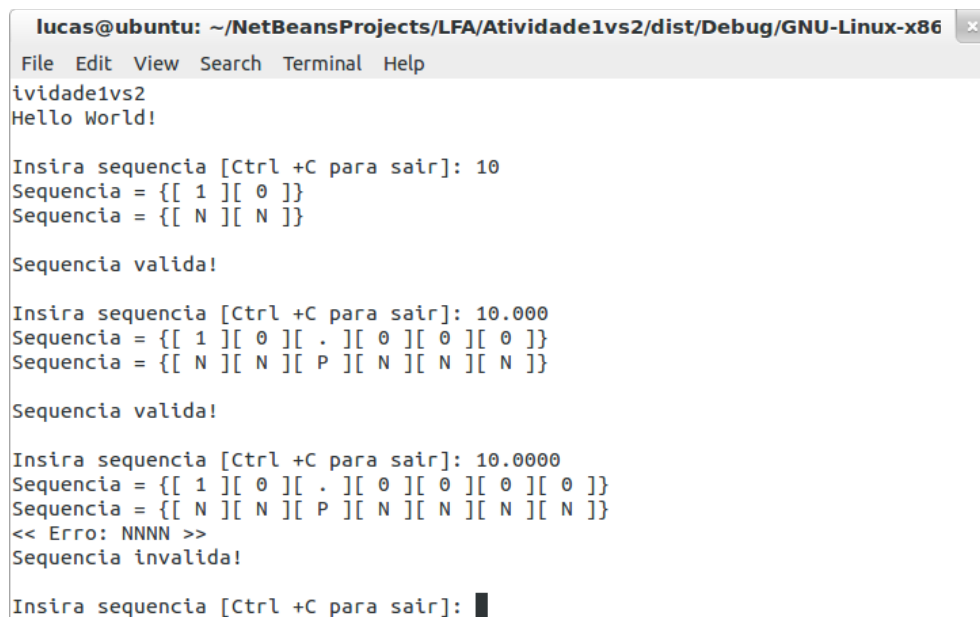
Primeiramente, é verificado se **SeqOriginal[]** é nula, caracterizando-a como inválida. Tal verificação é realizada pelo método **strlen()**, que retorna o número de elementos contidos em uma *string*. Este método é nativo no C++.

Em seguida, é verificada a ocorrência de cada uma das sub-sequências contidas na lista **SeqInvalidas[]**. Se ao menos uma destas sub-sequências ocorrer, a cadeia é inválida. Caso contrário este método retorna verdadeiro, classificando essa sequência como VÁLIDA. Esta verificação é realizada com o auxílio da função **strstr()**, que retorna o valor NULL caso não encontre nenhuma ocorrência.

Algumas sub-sequências verificadas (como descrito no parágrafo acima) possuem caracteres “especiais”, reservados para especificar o posicionamento da sequência (como, por exemplo, ! ou *). Nestes casos, operações sobre as variáveis que armazenam estas mesmas são realizadas a fim de analisá-las. Em outras palavras, sub-sequências como “V[..]P”, “[..]P”, “[..]PN”, “[..]PNN”, “[..]”, “[..]”, “[..]”, “[..]” tem “pedaços” cortados. A parte do vetor restante

constitui o começo, o início ou o meio da sequência original, mas agora, com uma posição fixa e não mais variável em relação a sequência completa. Desta forma, é possível analisar se determinada sub-cadeia inválida se apresenta o início ou fim, especificamente.

4. LISTA DE FIGURAS



```
lucas@ubuntu: ~/NetBeansProjects/LFA/Atividade1vs2/dist/Debug/GNU-Linux-x86
File Edit View Search Terminal Help
Atividade1vs2
Hello World!

Insira sequencia [Ctrl +C para sair]: 10
Sequencia = {[ 1 ][ 0 ]}
Sequencia = {[ N ][ N ]}

Sequencia valida!

Insira sequencia [Ctrl +C para sair]: 10.000
Sequencia = {[ 1 ][ 0 ][ . ][ 0 ][ 0 ][ 0 ]}
Sequencia = {[ N ][ N ][ P ][ N ][ N ][ N ]}

Sequencia valida!

Insira sequencia [Ctrl +C para sair]: 10.0000
Sequencia = {[ 1 ][ 0 ][ . ][ 0 ][ 0 ][ 0 ][ 0 ]}
Sequencia = {[ N ][ N ][ P ][ N ][ N ][ N ][ N ]}
<< Erro: NNNN >>
Sequencia invalida!

Insira sequencia [Ctrl +C para sair]:
```



```
lucas@ubuntu: ~/NetBeansProjects/LFA/Atividade1vs2/dist/Debug/GNU-Linux-x86
File Edit View Search Terminal Help
<< Erro: NNNN >>
Sequencia invalida!

Insira sequencia [Ctrl +C para sair]: 10.000,000
Sequencia = {[ 1 ][ 0 ][ . ][ 0 ][ 0 ][ 0 ][ , ][ 0 ][ 0 ][ 0 ]}
Sequencia = {[ N ][ N ][ P ][ N ][ N ][ N ][ V ][ N ][ N ][ N ]}

Sequencia valida!

Insira sequencia [Ctrl +C para sair]: 10,0
Sequencia = {[ 1 ][ 0 ][ , ][ 0 ]}
Sequencia = {[ N ][ N ][ V ][ N ]}

Sequencia valida!

Insira sequencia [Ctrl +C para sair]: 0,0
Sequencia = {[ 0 ][ , ][ 0 ]}
Sequencia = {[ N ][ V ][ N ]}

Sequencia valida!

Insira sequencia [Ctrl +C para sair]:
```

```
lucas@ubuntu: ~/NetBeansProjects/LFA/Atividade1vs2/dist/Debug/GNU-Linux-x86
File Edit View Search Terminal Help
lucas@ubuntu:~/NetBeansProjects/LFA/Atividade1vs2/dist/Debug/GNU-Linux-x86$ ./atividade1vs2
Hello World!

Insira sequencia [Ctrl +C para sair]: +203,12
Sequencia = {[ + ][ 2 ][ 0 ][ 3 ][ , ][ 1 ][ 2 ]}
Sequencia = {[ S ][ N ][ N ][ N ][ V ][ N ][ N ]}

Sequencia valida!

Insira sequencia [Ctrl +C para sair]: -+210
Sequencia = {[ - ][ + ][ 2 ][ 1 ][ 0 ]}
Sequencia = {[ S ][ S ][ N ][ N ][ N ]}
<< Erro: SS >>
Sequencia invalida!

Insira sequencia [Ctrl +C para sair]: 20-12
Sequencia = {[ 2 ][ 0 ][ - ][ 1 ][ 2 ]}
Sequencia = {[ N ][ N ][ S ][ N ][ N ]}
<< Erro: NS >>
Sequencia invalida!

Insira sequencia [Ctrl +C para sair]:
```

```
lucas@ubuntu: ~/NetBeansProjects/LFA/Atividade1vs2/dist/Debug/GNU-Linux-x86
File Edit View Search Terminal Help
lucas@ubuntu:~/NetBeansProjects/LFA/Atividade1vs2/dist/Debug/GNU-Linux-x86$ ./atividade1vs2
Hello World!

Insira sequencia [Ctrl +C para sair]: -203.201.32,203
Sequencia = {[ - ][ 2 ][ 0 ][ 3 ][ . ][ 2 ][ 0 ][ 1 ][ . ][ 3 ][ 2 ][ , ][ 2 ][ 0 ][ 3 ]}
Sequencia = {[ S ][ N ][ N ][ N ][ P ][ N ][ N ][ N ][ P ][ N ][ N ][ V ][ N ][ N ][ N ]}
<< Erro: PNNV >>
Sequencia invalida!

Insira sequencia [Ctrl +C para sair]: -203.201.322,203
Sequencia = {[ - ][ 2 ][ 0 ][ 3 ][ . ][ 2 ][ 0 ][ 1 ][ . ][ 3 ][ 2 ][ 2 ][ , ][ 2 ][ 0 ][ 3 ]}
Sequencia = {[ S ][ N ][ N ][ N ][ P ][ N ][ N ][ N ][ P ][ N ][ N ][ N ][ V ][ N ][ N ][ N ]}

Sequencia valida!

Insira sequencia [Ctrl +C para sair]: █
```