

## 1 Instruções Importantes

Nessa seção são apresentadas diversas informações relevantes referentes a entrega do trabalho e orientações a serem seguidas durante a implementação. Leia atentamente antes de começá-lo.

### 1.1 Equipe de Desenvolvimento

O trabalho será desenvolvido em grupos de até quatro alunos. Em hipótese alguma será permitido o trabalho ser realizado por cinco ou mais pessoas.

### 1.2 Escolha da Linguagem de Programação

O trabalho poderá ser desenvolvido em qualquer uma das linguagens a seguir: C/C++, Java e Python. Independente da linguagem a ser usada, todas as estruturas de dados referente a modelagem e desenvolvimento do trabalho deverão ser implementadas pelo grupo. Poderão ser usadas apenas bibliotecas de estruturas de dados elementares (listas, filas, pilhas, árvores, etc.) e leitura/escrita de strings.

### 1.3 Artefatos a Serem Entregues

Os artefatos a serem entregues são:

- código fonte do programa;
- arquivo de build para compilação do código fonte (makefile, ant, shell script, etc);
- documentação do trabalho em formato pdf.

Antes de enviar seu trabalho para avaliação, assegure-se que:

1. seu código compila utilizando a ferramenta de build e executa em ambiente Unix / Linux. Programas que não compilam receberão nota zero;
2. todos os arquivos fonte a serem enviados têm comentário no início do arquivo com nome e matrícula do(s) autor(es) do trabalho;
3. arquivo de documentação tenha a identificação do(s) autor(es) do trabalho;
4. arquivo compactado com os artefatos estão devidamente identificados com seu nome e matrícula.

### 1.4 Critérios de Avaliação

A avaliação será feita em duas etapas. A primeira entrega será avaliada em 10 pontos e a segunda etapa será avaliada em 20 pontos. As avaliações em ambas etapas serão feitas

mediante análise do código fonte, documentação e entrevista. Os seguintes fatores serão observados na avaliação do código fonte: corretude do programa, estrutura do código, redigibilidade e legibilidade. A corretude se refere à implementação correta de todas as funcionalidades especificadas, i.e., se o programa desenvolvido está funcionando corretamente e não apresenta erros. Os demais fatores avaliados no código fonte são referentes a organização e escrita do trabalho. Nesse quesito, serão observados os seguintes pontos:

- estruturas de dados bem planejadas;
- código modularizado em nível físico (separação em arquivos) e lógico;
- legibilidade do código, i.e., nomes mnemônicos de variáveis, funções e comentários úteis no código; e
- utilização adequada do GitHub para desenvolvimento do projeto, cada aluno deverá realizar os seus *commits*.

A documentação do código deve conter informações relevantes para compilar e executar o programa utilizando a ferramenta de build escolhida, e auxiliar no entendimento do código fonte. Ressalta-se que na documentação não deve conter cópias do código fonte - afinal o seu código fonte é um dos artefatos entregue, mas deve apresentar as decisões de projetos tomadas: estruturas de dados usadas na modelagem do problema, arquitetura do sistema, pseudocódigo dos algoritmos principais, dentre outras informações que o grupo julgar pertinente.

A entrevista tem por finalidade avaliar a confiabilidade e segurança do(s) autor(es) do código em explicar pontos relevantes do trabalho desenvolvido. Assim, a entrevista influenciará na avaliação dos artefatos entregues. Recomenda-se a criação de um roteiro de apresentação que mostre as funcionalidades implementadas em no máximo 20 minutos.

A nota final será dada a partir da avaliação do conjunto do código fonte, documentação e entrevista. É de responsabilidade do discente solicitar a marcação do dia e horário da entrevista com o professor da disciplina até a entrega da próxima etapa. Os trabalhos que não forem apresentados receberão nota zero.

Em caso de entrega após a data definida, o grupo perderá dois pontos por dia de atraso.

## 2 Especificação Técnica do Trabalho

O trabalho consiste em implementar um analisador de strings, o qual divide a entrada em uma sequência de subpalavras. Para tal, o usuário definirá um conjunto de “tags” (marcadores) que serão usados pelo programa para dividir a string de entrada. Por exemplo, na Tabela 1 há um conjunto de marcadores e uma breve descrição da linguagem representada por cada um deles. Supondo o texto de entrada “DCC146 = 1000 /\* Aspectos Teóricos da

Computação é nota 1000! \*/", o programa deverá apresentar a divisão desse texto conforme as tags da Tabela 1.

#### VAR SPACE EQUALS SPACE INT SPACE COMMENT

Tags	Linguagem que a tag descreve
INT	Número inteiros
SPACE	Sequência de espaços em branco
VAR	Sequência de símbolo alfanuméricos com o primeiro símbolo sendo uma letra
EQUALS	O símbolo "="
COMMENT	Qualquer sequência contida entre os símbolos "/*" e "*/"

Tabela 1. Definição das tags.

A classificação dar-se-á a partir do início da entrada, em sequência e sem retroceder às partes já classificadas. Isto é, a partir do início, a palavra CSI476 corresponde à tag VAR, assim o programa emite a saída correspondente e continua na entrada desse ponto, não retrocedendo na entrada. Como consequência, não será emitido a tag INT referente ao número 476. Portanto, na sequência, é emitido SPACE e assim por diante. Na Tabela 2 é apresentada a porção de texto correspondente a cada tag emitida. Vale ressaltar que a concatenação do texto referente a cada uma das tags emitidas é a string de entrada. O símbolo □ está sendo usado para representar um espaço em branco.

Teg emitida	Texto correspondente
VAR	DCC146
SPACE	□
EQUALS	=
SPACE	□
INT	1000
SPACE	□
COMMENT	/* Aspectos Teóricos da Computação é nota 1000! */

Tabela 2. Casamento das tags com o texto.

Nas seções seguintes são apresentados os detalhes de execução do programa e definição das tags.

## 2.1 Linguagem de Especificação das Tags

As tags são definidas, uma por linha, seguindo o padrão:

## NOME DA TAG: EXPRESSÃO DA TAG

Especifica-se o nome da tag, por exemplo VAR, seguido por dois pontos e após a definição da linguagem denotada pela tag. É válido ressaltar três pontos: (1) o nome da tag deve ser único, (2) entre o nome da tag e os dois pontos não é permitida a utilização de espaço e (3) existe apenas um espaço após os dois pontos que antecede a expressão da tag. A não observação dessas regras torna a definição da tag inválida.

A expressão da tag será definida usando expressões regulares em notação polonesa inversa, assim definida:

1.  $a \in \Sigma$ ,  $\lambda$  e  $\emptyset$  são expressões regulares que denotam, respectivamente, as linguagens  $\{a\}$ ,  $\{\lambda\}$  e  $\emptyset$ ;
2. se  $e_1$  e  $e_2$  são expressões regulares, então  $e_1e_2+$ ,  $e_1e_2.$  e  $e_1^*$  são expressões regulares que denotam, respectivamente, a união das linguagens denotadas por  $e_1$  e  $e_2$ , a concatenação da linguagem denotada por  $e_1$  com a linguagem denotada por  $e_2$  e o fecho de Kleene da linguagem denotada por  $e_1$ .

Por exemplo, a expressão regular em notação polonesa inversa  $ab.ba.++$  corresponde a expressão regular  $(ab + ba)^*$ , que denota a linguagem  $\{ab, ba\}^*$ . Assim, podemos especificar a tag INT, dessa forma:

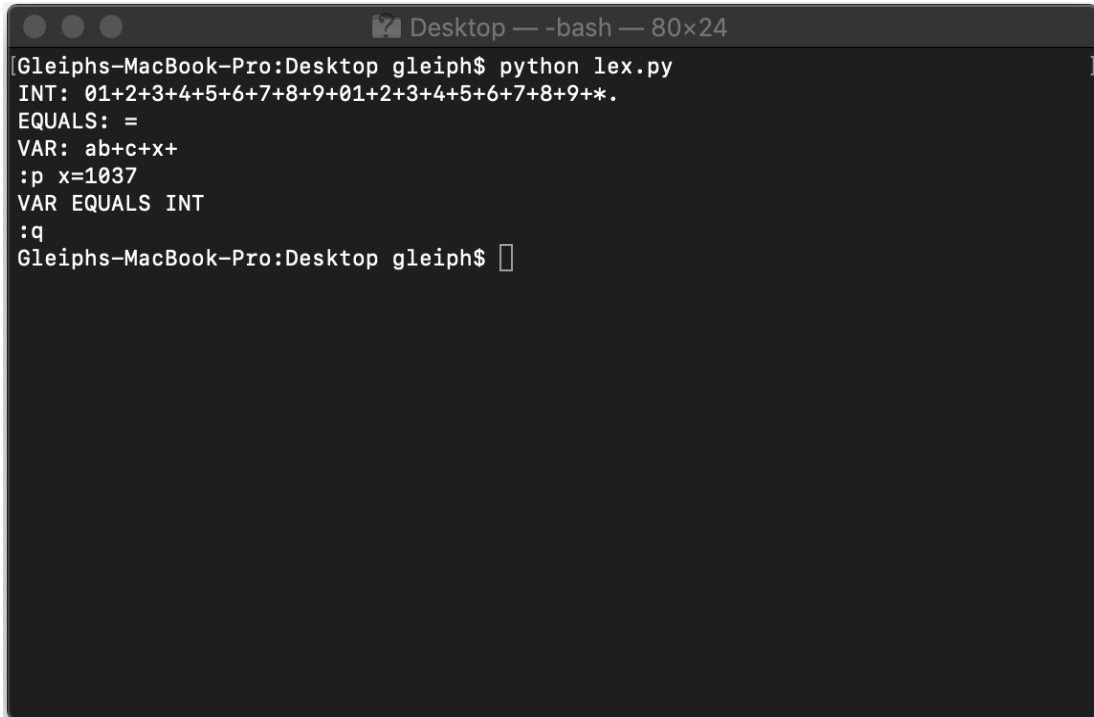
INT: 01+2+3+4+5+6+7+8+9+01+2+3+4+5+6+7+8+9+\*.

## 2.2 Execução do Programa

O programa deverá ser executado pelo terminal (linha de comando) e, quando executado, entrará em modo de interpretação. Toda interação entre o usuário e o programa dar-se-á no modo de interpretação, no qual o usuário poderá: especificar as tags conforme definido na Seção 2.1; executar o comando para “classificar” strings de entrada conforme as tags informadas; e executar demais comandos, como sair do programa. A Figura 1 mostra um exemplo de funcionamento do programa. Durante a execução do programa, o usuário definiu as tags INT, EQUALS e VAR, uma por linha. Em seguida usou o comando “:p” para realizar a classificação da entrada informada na mesma linha, “x=1037”. Após esse comando, o programa emitiu na tela a divisão da entrada, tendo como saída a linha “VAR EQUALS INT”. Na sequência, o usuário digitou o comando “:q” para encerrar o programa. Vale ressaltar que durante a execução do programa será dado apenas um comando por linha.

Em suma, o modo de interpretação aguarda definições de tags ou comandos. Os comandos começam com o símbolo de dois pontos seguido por um caractere e é sucedido pelo seu parâmetro, caso exista, separado por um espaço. Por outro lado, a definição de tag será realizada

como definida da Seção 2.1. A relação de comandos, descrição e exemplo de uso estão descritas na Tabela 3.



```
Desktop — -bash — 80x24
[Gleiphs-MacBook-Pro:Desktop gleiph$ python lex.py
INT: 01+2+3+4+5+6+7+8+9+01+2+3+4+5+6+7+8+9+*.
EQUALS: =
VAR: ab+c+x+
:p x=1037
VAR EQUALS INT
:q
Gleiphs-MacBook-Pro:Desktop gleiph$
```

Figura 1. Exemplo de execução do programa.

Comando	Descrição	Exemplo
:d	realiza a divisão em tags da string do arquivo informado	:d input.txt
:c	carrega um arquivo com definições de tags	:c tags.lex
:o	especifica o caminho do arquivo de saída para a divisão em tags	:o output.txt
:p	realiza a divisão em tags da entrada informada	:p x=1037
:a	Lista as definições formais dos autômatos em memória	:a
:l	Lista as definições de tag válidas	:l
:q	sair do programa	:q
:s	salvar as tags	:s file.txt

Tabela 3. Descrição dos comandos do programa.

O comando :d é utilizado para realizar a divisão do texto contido em um arquivo cujo caminho é passado por parâmetro. O funcionamento desse comando é semelhante ao do comando :p. A diferença entre ambos é que no comando :p o parâmetro é a entrada a ser dividida, enquanto no comando :d a entrada está contida em um arquivo, cujo caminho é

passado como parâmetro. O comando `:c` é utilizado para carregar definições de tags contidas em um arquivo texto. O parâmetro desse comando é o caminho do arquivo contendo as definições de tags que seguem a mesma sintaxe do modo de interpretação, com uma definição por linha. Ao finalizar a execução do `:c` apenas as tags válidas devem ser carregadas para a memória e um output ressaltando as tags válidas e inválidas deve ser apresentado. As saídas para os comandos `:d` e `:p` são feitas por padrão na tela.

Cada definição da tag válida será armazenada na memória e o será gerado um autômato para ela. Essas informações podem ser listadas com os comandos `:a` e `:l`. O comando `:a` é responsável por listar as definições formais de cada autômato, enquanto o `:l` lista todas as definições de tag armazenadas na memória.

A sequência das tags referente à divisão do texto de entrada é escrita na saída padrão (terminal), separadas por espaço. No entanto é possível alterar onde a saída será escrita com o comando `:o`, passando como parâmetro o caminho do arquivo no qual a saída deverá ser escrita. Finalmente, o programa é encerrado após executar o comando `:q`. Para salvar as tags no escopo da execução do programa é utilizado o comando `:s`, cujo parâmetro é o caminho do arquivo de texto no qual as definições serão escritas.

### 2.3 Emissões de Avisos, Erros e Informações ao Usuário

Mensagens de erros, avisos e informações deverão ser emitidas pelo programa. O início de cada mensagem terá uma das marcações [INFO], [ERROR] ou [WARNING] indicando se é uma informação, uma mensagem de erro ou mensagem de aviso, respectivamente. As mensagens de aviso serão emitidas após comandos que não têm saída para informar ao usuário que foi corretamente executado. Por exemplo, após executar o comando `“:l”` o programa emitirá na saída padrão a mensagem `“[INFO] As definições de tags foram carregadas”`.

As situações de erros são diversas, desde erros na entrada dos dados à erros na execução do programa. Destaca-se algumas situações que devem ser tratadas: erros na definição da tag; erros no comando; e erros na divisão da entrada. Deve-se verificar se há erros na especificação das tags, o qual pode ocorrer na definição do nome da tag ou com uma expressão regular malformada (errada). Erros nos comandos podem ocorrer por falta de parâmetros, parâmetros errôneos ou comandos inválidos. Erro na classificação acontecerá se não existir nenhuma tag para classificar alguma porção da entrada. Portanto, a divisão somente terminará com sucesso caso toda a entrada tenha sido dividida (classificada) com alguma tag.

O conjunto das palavras descritas por duas tags podem ter interseção. Por exemplos, suponha as definições das tags PAR e AB:

AB:  $a^*b^*$

PAR:  $ab.ba.+aa.+bb.+*$

Note que a entrada aabbbb, por exemplo, tem prefixo que pertence ao conjunto descrito por ambas as tags. O programa deverá identificar tais situações e emitir um aviso na saída padrão. Para esse exemplo, em particular, o programa deverá emitir a mensagem “[WARNING] Sobreposição na definição das tags PAR e AB”. Nessa situação de sobreposição, quando a classificação da entrada for realizada, o programa dará preferência para a tag que foi definida primeiro. Assim, nesse exemplo, a saída será “AB”, sendo que “aabbbb” casa com as tags AB e PAR.

Durante a divisão de tags será dada prioridade para a tag que casar com o maior prefixo. Por exemplo, dada com a entrada aabbbbbaa onde aabbbb e aa casam com AB e aabbbbbaa casa com PAR, daremos prioridade para a tag que casa com o maior prefixo comum: PAR.

## 2.4 Alfabeto de Entrada e Caracteres Especiais

O alfabeto da linguagem de expressão regular é o conjunto de caracteres especificado pela tabela ASCII (códigos 32 ao 126 da tabela ASCII), exceto os caracteres de controle, e os caracteres especiais para quebra de linha. A fim de definir linguagens (ou tags) com caracteres especiais, que usam os meta-símbolos da linguagem ou a cadeia vazia ( $\lambda$ ), serão usados símbolos de escape. Os símbolos de escape são especificados usando o símbolo  $\backslash$  seguido por um caractere. A tabela 4 apresenta todos os códigos de escape que devem ser tratados na definição das expressões regulares.

Código de escape	Caractere representado	Representação decimal do código ASCII
$\backslash n$	Quebra de linha	10
$\backslash \backslash$	$\backslash$	92
$\backslash *$	*	42
$\backslash \cdot$	.	46
$\backslash +$	+	187
$\backslash l$	$\lambda$	-

Tabela 4. Códigos de escape para a linguagem de expressões regulares.

## 3 Entrega do Trabalho

A entrega do trabalho será dividida em duas etapas: na primeira etapa será feita uma entrega parcial e, na segunda, o trabalho completo. Os artefatos que devem ser entregues e a forma de avaliação em ambas as etapas estão detalhadas nas Seções 1.3 e 1.4. A primeira entrega tem por finalidade avaliar o caminho que está sendo seguido no desenvolvimento do trabalho e dar orientações que devem ser observadas na segunda etapa. Portanto, as seguintes funcionalidades devem estar prontas na primeira etapa:

- interface de interação com usuário (comandos que ainda não foram implementados devem apresentar uma mensagem de *warning*);
- módulos para leitura e escrita em arquivo ou saída padrão;
- módulo de análise das definições de tags;
- emissões de avisos e erros em comandos e nas definições de tags; e
- projeto das estruturas de dados a serem usadas na solução do problema.

A divisão da entrada em tags, os erros na classificação e aviso de sobreposição de tags não precisam ser entregues na primeira etapa. Assim, a funcionalidade principal (divisão da entrada conforme as tags), seus respectivos erros e avisos deverão estar prontos para serem entregues na segunda etapa do trabalho. Todo o software com as funcionalidades corretamente implementadas será novamente avaliado na segunda etapa. Portanto, a segunda etapa consiste na entrega final do trabalho com todas as funcionalidades.

A data da entrega de ambas as etapas são:

- Primeira entrega: 13/07/2021;
- Entrega final: 31/08/2021.

A entrega será realizada via Google Classroom da disciplina. É de responsabilidade do aluno conferir se o trabalho foi corretamente enviado. Portanto, antes de enviar compile e execute o trabalho.