ESCOLA SUPERIOR DE TECNOLOGIA IPCA

# Trabalho Prático II

Processamento de Linguagens — 2019/2020

Licenciatura em Engenharia de Sistemas Informáticos

Versão 0.1 — November 12, 2019— Avaliação Contínua

**Instruções**

- Estes dois enunciados correspondem ao **segundo** trabalho prático de Processamento de Linguagens, regimes diurno e pós-laboral, no ano letivo 2019/2020, para a **avaliação contínua**;
- O trabalho prático será realizado em Grupo com um **máximo de 3 alunos**;
- A data limite para a entrega do segundo trabalho prático é o **dia 16 de Dezembro**, e a entrega será aceite apenas usando o formulário correspondente disponível no Moodle;
- As apresentações dos trabalhos práticos é feita por **todos os elementos do grupo**;
- Para além da implementação em C + flex + bison do projeto, deverá ser preparado um pequeno relatório que explique de que forma o enunciado foi interpretado, e quais as decisões tomadas na sua implementação.
- Qualquer detalhe que não esteja claro no enunciado deve ser extrapolado pelos alunos, optando pela interpretação que lhes parecer mais lógica/funcional.

## 1 Interpretador de Comandos

Na disciplina de Sistemas Operativos e Sistemas Distribuídos (SOSD) foram desenvolvidos vários comandos: `mostra`, `conta`, `apaga`, `informa`, `acrescenta` e `lista`. Pretende-se integrar estes comandos numa linguagem de *scripting* que permita fazer várias operações sobre ficheiros. Esta linguagem deverá permitir a execução direta dos comandos, tal como indicado no enunciado de SOSD. Para além disso:

- Todas as linhas que iniciem com `#` deverão ser ignoradas, e consideradas como comentários.
- Todos os comandos deverão permitir a especificação de mais do que um ficheiro. Implemente essa funcionalidade sem alterar a implementação original dos comandos realizados em SOSD. Exemplos de uso:

```
# apresenta no ecra os dois ficheiros indicados
mostra /etc/passwd /etc/group

# acrescenta em todos.txt os outros dois ficheiros
acrescenta todos.txt /etc/passwd /etc/group
```

```
# apaga os ficheiros ficheiro1.txt ficheiro2.txt
apaga ficheiro1.txt ficheiro2.txt
```

- Será possível definir variáveis com nomes de ficheiros:

  ```
  $texto = /etc/passwd
  ```

  Estas variáveis poderão posteriormente ser utilizadas em futuros comandos, como por exemplo:

  ```
  apaga $texto
  ```

  Ou ainda:

  ```
  $fich1 = ficheiro1.txt
  acrescenta $fich1 "ficheiro2"
  ```

- O comando `print` imprime uma string, substituindo todas as variáveis pelo seu respetivo valor. No exemplo seguinte, a string `=== /etc/hosts` seria impressa no ecrã:

  ```
  $hosts = /etc/hosts
  print "=== $hosts"
  ```

- Deverá ser possível implementar ciclos, sobre ficheiros de uma pasta, usando a seguinte sintaxe:

  ```
  para-cada $fich da-pasta /etc
     print "Conteúdo de: $fich"
     mostra $fich
  fim-para
  ```

- Note que será também necessário que seja possível a implementação de ciclos aninhados, tal como a seguir se descreve, para listar todos os ficheiros que se encontram dentro de qualquer pasta que se encontre em `/etc`:

  ```
  para-cada $fich1 da-pasta /etc
     para-cada $fich2 da-pasta /etc/$fich1
        mostra $fich1/$fich2
     fim-para
  fim-para
  ```

# 2 Linguagem de Desenho de Imagens Raster

Propõe-se a implementação de uma linguagem para a descrição (ou programação) de imagens.

A aplicação implementada deverá ser capaz de ler um documento de texto com um programa escrito usando a linguagem descrita na secção 3.2, e deverá executar os comandos nele contidos. Os comandos podem indicar para carregar uma imagem, desenhar rectas, rectângulos e círculos, escrever texto ou para gravar a imagem com um novo nome.

Antes de apresentar a linguagem de desenho, a secção que se segue explica o formato de imagem PNM que se sugere que os alunos usem. Este formato tem a vantagem de ser simples de ler e escrever, e a imagem pode ser armazenada em memória usando uma ou mais matrizes.

## 2.1 Formatos de Imagens

Existem centenas de formatos para armazenar imagens em computador. Alguns exemplos são o Portable Network Graphics (PNG), o Joint Photographic Experts Group (JPEG ou JPG), o Graphics Interchange Format (GIF), entre outros. Alguns destes, embora com um único nome, correspondem a famílias completas de formatos (em que varia a compressão, número de côr, suporte para transparências, etc).

Os formatos mais conhecidos são, tipicamente, formatos comprimidos. Isso significa que para além de saber converter cada ponto de uma imagem para uma côr, e armazenar no ficheiro, o programador também tem de saber como proceder à compactação do formato resultante.

No sentido de facilitar essa tarefa, sugere-se que os alunos usem um de dois formatos da família de formatos Portable aNyMap format (PNM). A família de formatos PNM tem a desvantagem dos seus ficheiros não serem compactados, e portanto, ocuparem mais espaço, mas a vantagem de serem simples de ler e/ou escrever. Cada um dos formatos da família PNM é designado por $Pn$ com $1 \leq n \leq 6$. As suas diferenças são apresentadas na tabela que se segue:

| **Nome** | **P1** | **P2** | **P3** | **P4** | **P5** | **P6** |
|---|---|---|---|---|---|---|
| **Cores** | B&W | 256 cinzentos | 16M cores | B&W | 256 cinzentos | 16M cores |
| **Codificação** | ASCII | ASCII | ASCII | Binário | Binário | Binário |

Para o trabalho em causa sugerimos que escolham o formato $P3$ ou $P6$ ou ambos. De seguida descrevem-se os formatos $P1$, $P3$ e $P6$.

Embora nem todas as aplicações suportem este formato, é relativamente simples converter uma imagem em PNM para um PNG, por exemplo:

```
convert foo.pnm foo.png
```

### 2.1.1 Formato P1

O formato $P1$ usa apenas duas cores (preto e branco, ou preto e ausência de cor). É usado o 1 para representar o preto, e o 0 para representar o branco.

Uma imagem $P1$ começa numa linha que contém apenas a string "`P1`." A linha seguinte indica o número de colunas e o número de linhas da imagem (número de pixeis na horizontal e na vertical). São dois valores inteiros, separados por um espaço. Seguem-se as várias linhas da imagem, representada por zeros e uns. No final de cada linha da imagem, é feita uma mudança de linha no ficheiro.

Segue-se o conteúdo de uma imagem em formato $P1$, que representa um **J**[1].

```
P1
6 10
0 0 0 0 1 0
0 0 0 0 1 0
0 0 0 0 1 0
0 0 0 0 1 0
0 0 0 0 1 0
0 0 0 0 1 0
1 0 0 0 1 0
0 1 1 1 0 0
0 0 0 0 0 0
0 0 0 0 0 0
```

### 2.1.2 Formato P3

Este formato é completamente textual e embora de leitura razoavelmente mais complicada para o ser humano (comparando-o com $P1$) ainda é possível a sua interpretação.

Tal como em $P1$, a primeira linha identifica o formato em causa, e tem apenas os caracteres `P3`. Segue-se uma linha com as dimensões da imagem (número de pixeis na horizontal e na vertical), separadas por um espaço. A terceira

[1] No caso do formato P1 (e apenas neste formato) os zeros e uns da imagem podem estar juntos, sem a existência do espaço a separá-los.

linha contém a *profundidade* ou *nível de cor* da imagem, que é indicado por um inteiro (habitualmente com o valor 255, para 256 níveis de cor).

Na quarta linha começam a ser descritos os pixeis da imagem, começando pelo canto superior esquerdo da imagem, linha a linha. Cada pixel é identificado por três valores inteiros que variam entre 0 e a *profundidade* da imagem definida. Estes três valores inteiros correspondem à intensidade das cores Vermelha, Verde e Azul. Habitualmente este modelo de cores é representado por RGB (Red, Green, Blue). Em cada uma das componentes, 0 indica a ausência dessa cor, enquanto que um valor positivo, quanto mais próximo da *profundidade* da imagem, indica maior luminosidade.

Segue-se um exemplo (minúsculo) de uma imagem neste formato:

```
P3
3 2
255
255   0 0     0 255   0   0 0 255
255 255 0   255 255 255   0 0   0
```

### 2.1.3 Formato P6

O formato **P6** é semelhante ao formato **P3**, mas é gravado em formato binário. A primeira linha deverá conter os caracteres `P6`, e a segunda e terceira linha são semelhantes ao formato **P3**. No entanto, a imagem propriamente dita (os pixeis da imagem) é codificada de forma binária, usando um byte para cada cor.

## 2.2 Linguagem de Desenho

A linguagem de desenho é definida por um conjunto de instruções. Cada instrução deve terminar num ponto e vírgula, tal como a linguagem `C`.

Os comandos são escritos em maiúsculas, as variáveis em minúsculas. Só existem valores do tipo inteiro, ponto (dois inteiros separados por uma vírgula), área (dois inteiros separados por um `x`), cor (três inteiros separados por dois pontos) ou *string* (delimitadas entre aspas), tal como veremos de seguida.

### 2.2.1 Gestão de Imagens

Existem três comandos para a gestão de imagens:

```
NEW 800x600 255:0:0;
```

Este comando prepara a área de desenho, definindo o seu tamanho (800 pixeis na horizontal, 600 pixeis na vertical) e a cor de fundo. As cores serão representadas por triplos, com valores entre 0 e 255, que indicam o peso de cada uma das componentes RGB na definição da cor.

A cor pode ser omitida, como em `NEW 800x600;`, o que corresponde a preparar a área de desenho com a cor branca.

```
LOAD "ficheiro.pnm";
```

O comando *load* carrega um ficheiro de imagem para memória. Se existir alguma imagem em memória, ela será destruída (e a memória que ocupava será descartada). Este comando deverá ser capaz de ler pelo menos um dos formatos P3 ou P6.

```
SAVE "ficheiro.pnm";
```

O comando *save* guarda a imagem actualmente em memória num ficheiro de imagem (P3 ou P6, à escolha do grupo).

### 2.2.2 Desenho

Todas as primitivas de desenho podem aceitar um último parâmetro com a cor que deve ser usada para desenhar. Caso este parâmetro não esteja definido, será usada a cor "por omissão", definida pelo comando `COLOR`.

```
COLOR 128:128:128;
```

Define a cor para ser usada por omissão. A partir do momento em que este comando é emitido, todos os comandos que não incluam informação de cor usam esta cor.

```
POINT 5,10 255:128:0;
```

```
POINT 4,100;
```

O comando *point* desenha um ponto nas coordenadas $x, y$ indicadas. Opcionalmente pode receber a cor que deve ser usada para o ponto (caso contrário será usada a cor por omissão).

```
LINE 5,10 10,20 255:128:0;
```

```
LINE 4,100 4,200;
```

As linhas são desenhadas pelo comando *line* que recebe dois pontos $p_1 = (x_1, y_1)$ e $p_2 = (x_2, y_2)$ e desenha uma recta entre eles (usando a cor indicada ou a cor por omissão).

```
RECT 5,10 10,20 255:128:0;
```

```
RECT 5,10 20x20 255:128:0;
```

```
RECT 4,100 40,200;
```

```
RECT 4,100 30x40;
```

Os rectângulos podem ser desenhados explicitando ou não a cor, e indicando dois pontos opostos $p_1$ e $p_2$, ou um ponto $p_1$ (posição do canto superior esquerdo) e as dimensões do rectângulo.

```
RECTFILL 5,10 10,20 255:128:0;
```

```
RECTFILL 5,10 20x20 255:128:0;
```

```
RECTFILL 4,100 40,200;
```

```
RECTFILL 4,100 30x40;
```

O comando *rectfill* é semelhante ao comando *rect*, mas o seu conteúdo é pintado (ou seja, o comando *rect* desenha apenas o rectângulo e não preenche o seu conteúdo).

```
CIRC 5,10 100 80:40:20;
```

```
CIRC 5,10 100;
```

O comando *circ* desenha um círculo centrado no ponto $p_1$ indicado, e com o raio apresentado.

```
POLYLINE 5,10 10,10 10,15 20,30 128:0:0;
```

```
POLYLINE 5,10 10,15 10,20;
```

O *polyline* desenha várias linha entre os pontos indicados. Ou seja, para os pontos $p_1$, $p_2$, $p_3$ e $p_4$, o comando desenha uma linha de $p_1$ para $p_2$, outra de $p_2$ para $p_3$ e uma outra de $p_3$ para $p_4$. Note que o comando *polyline* apenas com dois argumentos é equivalente ao comando *line*.

### 2.2.3 Variáveis e Expressões

Qualquer sequência de caracteres e números (que inicie por uma letra diferente de `x`) em minúsculas define uma variável. Uma variável pode ser usada em qualquer sítio onde um valor inteiro é usado. Por exemplo, o seguinte comando seria correcto:

```
RECT a1, a2  t1 x t2  c1 : c2 : c3;
```

```
var = 10;
```

```
var2 = var1;
```

As variáveis podem ser atribuídas tal como em C, usando o sinal `=`, e o valor pode ser um inteiro ou uma outra variável.

```
a = RAND 10;
```

Existe um caso especial, apresentado acima, que coloca na variável em causa um número aleatório entre 0 e o número indicado (no caso apresentado, 10).

### 2.2.4 Ciclos

A linguagem de desenho também suporta ciclos *for*, como qualquer linguagem imperativa, mas a sua sintaxe é ligeiramente diferente.

```
FOR i IN [10..20] DO .... END FOR;
```

Este comando irá executar 11 vezes o código colocado no lugar das reticências, sendo que em cada iteração o valor da variável $i$ irá variar, começando em 10 e incrementando até 20[2]. Note que podem-se usar ciclos dentro de ciclos.

## 2.3 Exemplos

Seguem-se alguns pequenos exemplos de programas escritos nesta linguagem.

**Exemplo 1**

```
NEW 101x101;
COLOR 0:0:0;
```

[2]Equivalente a um ciclo `for(i=10;i<=20;i++)` em C.

```
center = 51;
iradius = RAND 25;
CIRC center,center 50;
COLOR 150:0:100;
CIRCFILL center,center iradius;
COLOR 255:0:0;
POLYLINE 10,10 50,10 50,90 90,90;
SAVE "foo.pnm";
```

**Exemplo 2**

```
NEW 100x100 255:255:255;
COLOR 0:0:0;
RECT 1,1 100x100;
FOR t IN [1..10] DO
   p1 = RAND 50;
   p2 = RAND 50;
   t1 = RAND 50;
   t2 = RAND 50;
   c1 = RAND 255;
   c2 = RAND 255;
   c3 = RAND 255;
   RECTFILL p1,p2 t1 x t2 c1:c2:c3;
END FOR;
SAVE "bar.pnm";
```