# Trabalho Prático

## 1 Introdução

Com este projeto pretende-se que os alunos da UC de Processamento de Linguagens adquiram experiência na definição de analisadores léxicos e sintáticos, bem como na definição de ações semânticas que traduzem as linguagens implementadas.

O processo para a realização do trabalho deverá passar pelas seguintes fases:

1. especificar a gramática concreta da linguagem de entrada;
2. construir um reconhecedor léxico e sintático para essa linguagem com recurso ao par de ferramentas geradoras `flex`+`bison`;
3. desenvolver o gerador de código que produza a resposta solicitada, associando ações semânticas de tradução às produções da gramática.

Nesse sentido, o que se propõe é a implementação de uma linguagem para a edição de imagens. De realçar que o enunciado não envolve apenas a análise e reconhecimento da linguagem, mas também a tomada de ações, e a geração de uma representação textual de uma imagem.

A aplicação implementada deverá ser capaz de ler um ficheiro 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 em memória uma imagem, desenhar segmentos de retas, retângulos e circunferências, escrever texto ou para gravar a imagem com um novo nome.

Na secção seguinte são apresentadas as regras deste trabalho prático. Na secção 3 é apresentado o enunciado do problema proposto para este trabalho prático. O enunciado antes de apresentar a linguagem de edição de imagens, começa por explicar o formato de imagem PNM que se sugere que os alunos usem. Este formato tem as vantagens de ser simples de ler e escrever, e a imagem pode ser armazenada em memória usando uma ou mais matrizes.

## 2 Regras

- O trabalho prático deve ser realizado por grupos de dois alunos;
- O trabalho prático consiste em três componentes de avaliação: uma aplicação, um relatório e uma defesa oral;
- A aplicação deverá ser desenvolvida na linguagem de programação `C`, usando as ferramentas geradoras `flex` e `bison`. O código deve ser passível de ser compilado em Linux. Os alunos são convidados a desenvolverem o seu código de forma modular, e apresentar uma *makefile* que automatize a compilação da aplicação.
- O relatório deve introduzir o problema a ser resolvido, apresentar a abordagem seguida na sua resolução, quais os objectivos atingidos e quais os problemas encontrados. No relatório devem ser salientados todos os pontos que os alunos achem que poderão valorizar o seu trabalho em relação aos requisitos como, por exemplo, funcionalidades adicionais. Também deverá conter uma secção que descreva de que forma é que a aplicação foi testada.

  Devem ser colocadas todas as referências consultadas durante a elaboração do trabalho (bibliográficas ou mesmo consultas *online*);

  Os alunos são convidados a experimentarem a ferramenta LaTeX para a escrita do seu relatório. Na página da disciplina serão colocados apontadores para documentação para consulta pelos alunos interessados.
- Durante as aulas dedicadas aos trabalhos poderá ser solicitado aos alunos que apresentem o trabalho desenvolvido até esse momento.
- Cada grupo deverá defender pessoalmente o trabalho com o docente. Na data da apresentação (a definir atempadamente) devem comparecer à defesa com uma cópia do relatório impressa, e um portátil com a aplicação funcional.
- A data de entrega final é aquela que foi estabelecida no início do semestre.
- A falha de qualquer um dos três componentes de avaliação acima enumerados corresponderá a uma avaliação negativa na componente do trabalho prático.

# 3 Enunciado

O que se pretende neste trabalho prático é a implementação de uma ferramenta em `C`, usando `flex` e `bison`, que a partir de uma sequência de comandos para editar uma imagem, determine a representação da imagem que resulta da execução desses comandos.

A ferramenta a desenvolver deverá começar por ler um ficheiro de texto (com extensão *.ei*, para edição de imagem) contendo uma sequência de comandos de edição de imagens, aplique esses comandos de forma a calcular o conteúdo (cor) de cada pixel da imagem calculada.

Um exemplo de invocação da ferramenta poderá ser:

```
calcimagem   entrada.ei
```

O resultado de processar o ficheiro de texto `entrada.ei` é guardado durante a execução através de um comando *guardar*.

## 3.1 Formatos de Imagem PNM

No âmbito deste trabalho prático sugere-se que os alunos considerem a família dos formatos de imagem Portable aNyMap (PNM)[1]. 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 imagem da família PNM é $Pn$ com $1 \leq n \leq 6$. Para este trabalho sugere-se que escolham o formato P3 ou o formato P6 (ou ambos). De seguida descrevem-se os formatos P3 e P6.

### 3.1.1 formato P3

Uma imagem neste formato é guardada num ficheiro de texto no qual, 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 pixéis na horizontal e na vertical), separadas por um espaço. A terceira 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 pixéis 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:

```
1  P3
2  3 2
3  255
4  255   0 0     0 255   0   0   0 255
5  255 255 0   255 255 255   0 255   0
```

[1] `https://en.wikipedia.org/wiki/Netpbm_format`

### 3.1.2 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.

## 3.2 Linguagem para a Edição de Imagens

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, coordenada (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.

### 3.2.1 leitura e escrita de imagens

Existem três comandos para a gestão de imagens:

`NOVA 800x600 255:0:0;`
Este comando prepara a área de desenho, definindo o seu tamanho (800 pixéis na horizontal, 600 pixéis 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 `NOVA 800x600;`, o que corresponde a preparar a área de desenho com a cor branca.

`ABRIR "imagem.pnm";`
O comando *abrir* 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.

`GUARDAR "imagem.pnm";`
O comando save guarda a imagem actualmente em memória num ficheiro de imagem (P3 ou P6, à escolha do grupo).

### 3.2.2 edição de imagens

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 *cor*.

`COR 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.

```
PONTO 5,10 255:128:0;
PONTO 4,100;
```
Este comando desenha um ponto nas coordenadas x; y indicadas. Opcionalmente pode receber a cor que deve ser usada para esse ponto (caso contrário será usada a cor por omissão).

```
LINHA 5,10 10,20 255:128:0;
LINHA 4,100 4,200;
```
As linhas são desenhadas por este comando, o qual recebe as coordenadas de dois pontos $p1 = (x1, y1)$ e $p2 = (x2, y2)$ e desenha um segmento de reta entre eles (usando a cor indicada ou a cor por omissão).

```
RETANGULO 5,10 10,20 255:128:0;
RETANGULO 5,10 20x20 255:128:0;
RETANGULO 4,100 40,200;
RETANGULO 4,100 30x40;
```
Os retângulos podem ser desenhados explicitando ou não a cor, e indicando dois pontos opostos $p1$ e $p2$, ou um ponto $p1$ (posição do canto superior esquerdo) e as dimensões do retângulo.

```
RETFILL 5,10 10,20 255:128:0;
RETFILL 5,10 20x20 255:128:0;
RETFILL 4,100 40,200;
RETFILL 4,100 30x40;
```
O comando *retfill* é semelhante ao comando *retangulo*, mas o seu conteúdo é pintado (ou seja, o comando *retangulo* desenha apenas o rectângulo e não preenche o seu conteúdo).

```
CIRCUNF 5,10 100 80:40:20;
CIRCUNF 5,10 100;
```
O comando *circunf* desenha uma circunferência centrada no ponto indicado, e com o raio apresentado.

```
LINHAS 5,10 10,10 10,15 20,30 128:0:0;
LINHAS 5,10 10,15 10,20;
```
Este comando desenha várias linhas entre os pontos indicados. Ou seja, para os pontos $p1$, $p2$, $p3$ e $p4$, o comando desenha uma linha (segmento de reta) de $p1$ para $p2$, outra de $p2$ para $p3$ e uma outra de $p3$ para $p4$. Note que o comando *linhas* apenas com dois pontos nos argumentos é equivalente ao comando *linha*.

### 3.2.3 variáveis e expressões

Qualquer sequência de letras (minúsculas) e números (que inicie por uma letra diferente de x) pode ser utilizada como identificados de uma variável. Uma variável pode ser usada em qualquer local onde um valor inteiro é usado. Por exemplo, os seguintes comandos estariam corretos:

```
RETANGULO a1, a2 t1 x t2 c1 :  c2 :  c3;
CIRCUNF c,c raio 128 :  128:  azul ;
```

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

```
var1 = 10;
var2 = var1;
```

Existe um caso especial, conforme se segue, que coloca na variável em causa um número aleatório entre 0 e o número inteiro indicado (no caso apresentado, 10):

```
a = RAND 10;
```

Poderá ainda ser considerada a possibilidade de atribuição suportar operações aritméticas como por exemplo:

```
a = 10 + i * 2;
```

### 3.2.4 ciclos

A linguagem de edição de imagens também suporta ciclos, como qualquer linguagem imperativa, com seguinte sintaxe:

```
PARA i EM [10..20] FAZER ...  FIM PARA;
```

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. Note que podem-se usar ciclos dentro de ciclos.

### 3.2.5 funções

A linguagem de edição de imagens poderá também suportar a definição (e invocação) de funções, possibilitando a definição de um conjunto de comandos, os quais poderão ser invocados várias vezes utilizando parâmetros diferentes. Cada grupo deverá sugerir uma implementação e integração para esta funcionalidade da linguagem.

### 3.2.6 resultado

Os alunos deverão sugerir uma solução para situações de erro que poderão ocorrer durante o processamento de um ficheiro de entrada, tais como:

- O que acontece se o utilizador se enganar num comando?
- O que acontece se o utilizador usar coordenadas que estão fora dos limites da área de desenho?
- Será que só se poderá desenhar um circunferência (ou uma linha, porque não) se as coordenadas não estiverem fora dos limites da área de desenho?
- O que acontece se o utilizador tentar utilizar uma variável que ainda não foi declarada?

O programa deverá gerar uma representação da imagem (nos formatos indicados anteriormente) que resulta das ações realizadas pelos comandos indicados no texto de entrada, e guardar esse resultado no parâmetro do comando *guardar*.

## 3.3 Dúvidas

Quando o enunciado não for explícito em relação a um requisito específico, os alunos podem optar pela solução que parecer mais adequada, fazendo a sua apresentação e justificação no relatório. Dúvidas adicionais devem ser colocadas ao docente pessoalmente ou por *email*.