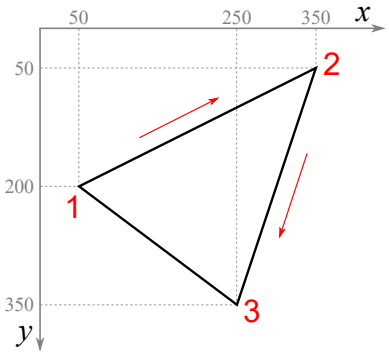
- Implemente um programa, em Prolog, para criar uma base de dados de pontos e deslocamentos que, a partir de suas ligações por segmentos de retas, na sequência em que ocorrem, possam formar desenhos. Para isto é definido um predicado `xy` com aridade 3, constituído pelos seguintes argumentos:
  - **1º:** identificador `Id` numérico (1, 2, ...) para que, em uma base de dados, possam ter diversos desenhos, cada um com um ponto inicial diferente;
  - **2º:** assume uma, entre duas funcionalidades possíveis:
    → coordenada `X` inicial (para a primeira ocorrência na base)
    → deslocamento em `X` (demais ocorrências na base)
  - **3º:** assume uma, entre duas funcionalidades possíveis:
    → coordenada `Y` inicial (para a primeira ocorrência na base)
    → deslocamento em `Y` (demais ocorrências na base)

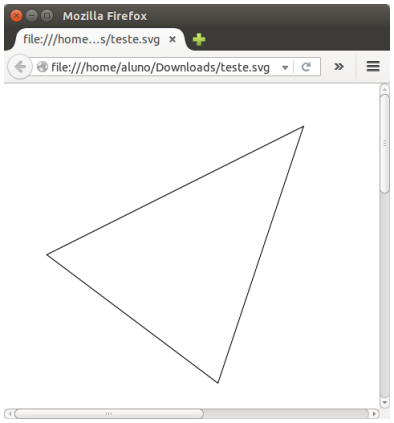
- Exemplo de banco de dados:

```
:- dynamic xy/3.

xy(1, 50.0, 200.0).
xy(1, 300.0, -150.0).
xy(1, -100.0, 300.0).
```

Neste exemplo, há apenas um desenho com identificador `'1'`. O ponto inicial é dado pela primeira ocorrência do predicado `xy` com argumento `'1'` no banco de dados, ou seja, a coordenada `(50, 200)`. O predicado `xy` seguinte desenha uma linha a partir de um deslocamento em relação a este primeiro ponto, ou seja, a coordenada `x` aumenta 300 e a coordenada y subtrai 150 para gerar o segundo ponto em `(350,50)`. A terceira ocorrência de `xy`, por sua vez, desenha uma linha a partir do deslocamento do ponto anterior em -100 para o `x` e 300 para o y para gera o ponto `(250,350)`, e assim por diante. A figura à esquerda ilustra o desenho formado (o último ponto se liga ao primeiro automaticamente). A figura à direita é uma exibição, no navegador, de uma versão em SVG (*Scalable Vector Graphics*) do banco de dados, convertido pelo programa `db2svg.pl` disponibilizado pelo professor.



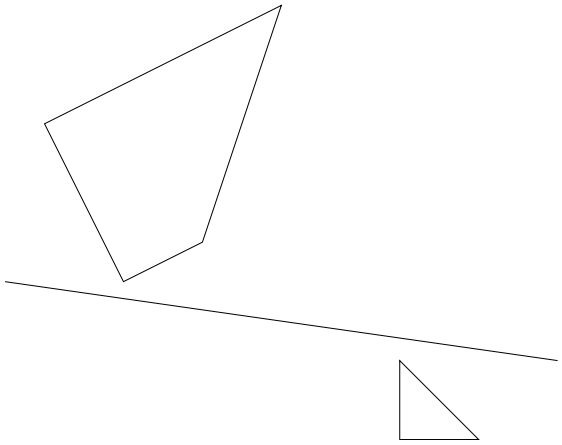


- Segue um exemplo com três identificadores, disponibilizado em `desenhos.pl`:

```
:- dynamic xy/3.

xy(3, 0, 400).
xy(2, 500, 500).
xy(1, 50.0, 200.0).
xy(1, 300.0, -150.0).
xy(1, -100.0, 300.0).
xy(1, -100, 50).
xy(2, 100, 100).
xy(2, -100, 0).
xy(3, 700, 100).
```

E os desenhos produzidos:



- Materiais:
  - `programa.pl`:
    `https://www.inf.ufsc.br/~alexandre.silva/courses/17s2/ine5416/exercicios/t2A/programa.pl`
  - `desenhos.pl` – arquivo fixo utilizado para a base de dados; exemplos para teste:
    → Exemplo 1:
    `https://www.inf.ufsc.br/~alexandre.silva/courses/17s2/ine5416/exercicios/t2A/ex1/desenhos.pl` (DB)
    `https://www.inf.ufsc.br/~alexandre.silva/courses/17s2/ine5416/exercicios/t2A/ex1/desenhos.svg` (SVG)

    → Exemplo 2:
    `https://www.inf.ufsc.br/~alexandre.silva/courses/17s2/ine5416/exercicios/t2A/ex2/desenhos.pl` (DB)
    `https://www.inf.ufsc.br/~alexandre.silva/courses/17s2/ine5416/exercicios/t2A/ex2/desenhos.svg` (SVG)

- `db2svg.pl` – conversão para `desenhos.svg`, possibilitando a visualização no navegador:
  `https://www.inf.ufsc.br/~alexandre.silva/courses/17s2/ine5416/exercicios/t2A/db2svg.pl`

  Modo de uso (em linha de comando):
  ```
  swipl -f db2svg.pl -t halt > desenhos.svg 2> stderr.txt
  ```

- O `programa.pl` já implementa os seguintes predicados:
  - `menu.` → Exibe menu principal
  - `load.` → Carrega os desenhos a partir do banco de dados `desenhos.pl`
  - `commit.` → Grava os desenhos da memoria no banco de dados `desenhos.pl`
  - `new(Id,X,Y).` → Insere deslocamento, se *Id* existente, ou ponto inicial, caso contrário
  - `search.` → Exibe opções de busca
  - `search(Id,L).` → Monta lista L com ponto inicial e todos os deslocamentos de *Id*
  - `remove.` → Exibe opções de remoção
  - `svg.` → Grava os desenhos, em SVG, no arquivo `desenhos.svg`

  Veja como efetuar inserções de novos pontos em desenhos existentes, criar novos desenhos, gravar o banco de dados, e gerar o SVG correspondente.

- Escreva predicado que:

1. Monta lista $< L >$ com ponto inicial e todos os deslocamentos de $< Id >$.

   `searchId(Id,L) :-` _____

   Exemplo:

   ```
   ?- searchId(1,L).
       L = [[50.0, 200.0], [300.0, -150.0], [-100.0, 300.0], [-100, 50]].
   ```

2. Monta lista $< L >$ com pontos iniciais de cada $< Id >$ (em ordem).

   `searchFirst(L) :-` _____

   Exemplo:

   ```
   ?- searchFirst(L).
   L = [[50.0, 200.0], [500, 500], [0, 400]].
   ```

3. Monta lista $< L >$ com pontos/deslocamentos finais de cada $< Id >$ (em ordem).

   `searchLast(L) :-` _____

   Exemplo:

```
?- searchLast(L).
L = [[-100, 50], [-100, 0], [700, 100]].
```

4. Remove todos os pontos/deslocamentos do último $< Id >$.

   `removeLast :-` _____

   Exemplo:

```
?- listing(xy).
:- dynamic xy/3.

xy(3, 0, 400).
xy(2, 500, 500).
xy(1, 50.0, 200.0).
xy(1, 300.0, -150.0).
xy(1, -100.0, 300.0).
xy(1, -100, 50).
xy(2, 100, 100).
xy(2, -100, 0).
xy(3, 700, 100).
```

```
?- removeLast.
```

```
?- listing(xy).
:- dynamic xy/3.

xy(2, 500, 500).
xy(1, 50.0, 200.0).
xy(1, 300.0, -150.0).
xy(1, -100.0, 300.0).
xy(1, -100, 50).
xy(2, 100, 100).
xy(2, -100, 0).
```

5. Remove o último ponto/deslocamento de $< Id >$.

   `removeLast(Id) :-` _____

   Exemplo:

```
?- listing(xy).
:- dynamic xy/3.

xy(3, 0, 400).
xy(2, 500, 500).
xy(1, 50.0, 200.0).
xy(1, 300.0, -150.0).
xy(1, -100.0, 300.0).
xy(1, -100, 50).
xy(2, 100, 100).
xy(2, -100, 0).
xy(3, 700, 100).
```

```
?- removeLast(1).
```

```
?- listing(xy).
:- dynamic xy/3.

xy(3, 0, 400).
xy(2, 500, 500).
xy(1, 50.0, 200.0).
xy(1, 300.0, -150.0).
xy(1, -100.0, 300.0).
xy(2, 100, 100).
xy(2, -100, 0).
xy(3, 700, 100).
```

6. Determina um novo $< Id >$ na sequência numérica existente.

   `newId(Id) :-` _____

   Exemplo:

```
?- newId(Id).
Id = 4.
```

7. Duplica a figura com $< Id >$ a partir de um nova posicao (X, Y). Deve ser criado um $< Id\_novo >$ conforme a sequência (questao 6).

```
cloneId(Id,X,Y) :- _____________________
```

Exemplo:

```
?- listing(xy).
:- dynamic xy/3.

xy(3, 0, 400).
xy(2, 500, 500).
xy(1, 50.0, 200.0).
xy(1, 300.0, -150.0).
xy(1, -100.0, 300.0).
xy(1, -100, 50).
xy(2, 100, 100).
xy(2, -100, 0).
xy(3, 700, 100).
```

```
?- cloneId(3, -10, 15).
```

```
?- listing(xy).
:- dynamic xy/3.

xy(3, 0, 400).
xy(2, 500, 500).
xy(1, 50.0, 200.0).
xy(1, 300.0, -150.0).
xy(1, -100.0, 300.0).
xy(1, -100, 50).
xy(2, 100, 100).
xy(2, -100, 0).
xy(3, 700, 100).
xy(4, -10, 415).
xy(4, 700, 100).
```

- **Entrega do T_2–parte A:**
  - **Prazo:** dia 19out2017 até 23h55
  - **Forma:** individual
  - **Submissão pelo VPL-Moodle:**
    1. Ao editar, salvar e executar o código-fonte, em "programa.pl" (no VPL), tem-se o registro de submissão
    2. A execução/avaliação pode ser feita, pelo navegador, quantas vezes forem necessárias
    3. Os exemplos de execução são produzidos pelo próprio VPL (os predicados dever ter os mesmos nomes indicados em cada exercício)