- Faça login no computador com a conta `sessao1` e password `1`.
- No directório *Desktop/exam* vai encontrar um conjunto de ficheiros úteis para o exame.
- Utilize o executável `./run-jar` como complemento na especificação do programa.
  Exemplo (num terminal aberto em `Desktop/exam`): `./run-jar p1.txt`
- Desenvolva a linguagem por forma a que todos os programas de exemplo da linguagem (`p?.txt`) sejam aceites (`p1.txt` para as alíneas a) e b), `p2.txt` para a alínea c), etc.).
- Pode consultar a documentação das classes `Java` usando o comando `view-javadoc`.
  Exemplo: `view-javadoc String`
- Tem à sua disposição os comandos de apoio à programação em `ANTLR4`:
  `antlr4`, `antlr4-test` (`a4-test`), `antlr4-visitor` (`visitor`), `antlr4-listener` (`listener`), `antlr4-main` (`main`), `antlr4-build` (`build`), `antlr4-run` (`run`), `antlr4-clean` (`clean`)
- Utilize o enunciado como rescunho, e no **fim entregue-o com o cabeçalho preenchido.**
- Caso o seu exame seja feito com uma pen, **entregue a pen juntamente com o enunciado.**
- Caso pretenda desistir deve indicar essa decisão no enunciado e executar o comando: `desisto`

Problema: Pretende-se implementar um interpretador para uma linguagem de manipulação de tuplos de números. Nesta linguagem, um tuplo de números é uma sequência, que pode ser vazia, de números reais (sintacticamente é delimitada por parêntesis rectos e separada por vírgulas). Todas as operações nesta linguagem operam sobre tuplos e têm como resultado um tuplo. Considere que os tuplos não são recursivos, pelo que não é permitida a inclusão de um tuplo dentro de outro tuplo.
Para representar tuplos sugere-se a utilização da classe `ArrayList` (instanciada para o tipo `Double`) da biblioteca java[1] (`java.util.ArrayList`).
Como exemplo inicial, considere o seguinte programa[2]:

```
--p1.txt
print [1,2,3];          --escreve tuplo de números na consola
a := [5];               --guarda tuplo de números na variável a
print a;                --escreve na consola a tuplo de números armazenado na variável a
print;                  --escreve linha em branco
a:=[5,4,3,2,1];         --guarda tuplo de números na variável a
print a,[1];            --escreve na consola os tuplos de números (a e [1]) separados por um espaço
```

Nota 1: partindo das instruções exemplificadas, tente tornar a linguagem o mais genérica possível.

Nota 2: os identificadores para variáveis contêm apenas letras (ASCII) e dígitos, não podendo começar com um dígito.

Nota 3: O carácter `,` na instrução `print` funciona como um separador de expressões mas apenas nesta instrução (i.e. não é um operador que deva fazer parte duma expressão).

Nota 4: Os números reais podem ser inteiros ou números reais com vírgula fixa (e.g. `4` e `3.1`).

Nota 5: Não se esqueça de ter em conta a ocorrência de erros. Existem ficheiros `err?.txt` para o ajudar nesse fim.

a) Implemente em `ANTLR4`, uma gramática `TupleNum` para esta linguagem (exemplificada em `p1.txt`). [4.5 valores]

b) Implemente um interpretador que faça a verificação semântica e execute as instruções desta linguagem[3]. [4.5 valores]

(Utilize o comando `antlr4-run` para testar o interpretador.)

[1] Ver documentação com: `view-javadoc ArrayList`.
[2] Na escrita do tuplo pode usar directamente a escrita (`toString`) do objecto do tipo `ArrayList`.
[3] Na criação do visitor use as pelicas (') para evitar uma interpretação errada dos símbolos `<>`.

c) Altere a gramática e o interpretador por forma a permitir a realização das seguintes operações sobre tuplos (ver programa `p2.txt`): [6 valores]

- extração de sub-tuplos com os operadores `head` (novo tuplo contendo apenas o primeiro elemento do tuplo ao qual está a ser aplicado) e `tail` (todos os elementos do tuplo excepto o primeiro); operadores `sum` (novo tuplo contendo um elemento que é a soma de todos os elementos do tuplo original) e `average` (novo tuplo contendo um elemento que é a média de todos os elementos do tuplo original).
- operadores unários `+` e `-`. Por exemplo: `-[1,2]` (resultado: `[-1,-2]`).
- soma e subtracção de tuplos com os operadores `+` e `-`.
  Por exemplo: `[3]+[2]` (resultado: `[5]`) ou `[1,2]+[2,3,4]` (resultado: `[3,5,4]`).
- parêntesis: este operador serve para impor prioridades na realização de operações aritméticas.

As prioridades nestes operadores devem ser, por ordem decrescente, as indicadas pelas alíneas (i.e. o mais prioritário é a extração de sub-tuplos/somatório/média, depois os operadores unários aritméticos, etc.).

```
--p2.txt
xx2:=[1,2,3,4];
print xx2;
print xx2 + [4,3,2,1];  --[5,5,5,5]
print xx2 - xx2;        --[0,0,0,0]
print +xx2;             --[1,2,3,4]
print -xx2;             --[-1,-2,-3,-4]
print head xx2;         --[1,2,3]
print tail xx2;         --[2,3,4]
print sum xx2;          --[10]
print average xx2;      --[2.5]
print head xx2 + tail xx2 - average xx2 + sum xx2;
print [], sum [];
print [3] + (head ([3]+[1])+tail [5,3] + sum (xx2+[1])+ average [5,3]);
```

d) Faça com que o interpretador leia o programa a partir de um, ou mais, ficheiros (cujos nomes são passados como argumentos do programa), e altere a gramática por forma a permitir a entrada de tuplos pelo utilizador (ver programa `p3.txt`). O tuplo à entrada deve ser uma sequência de números (a converter para `Double`), separados por vírgula, numa única linha de texto (leitura com `nextLine`). Um tuplo vazio corresponde a uma linha vazia. [3 valores]

```
--p3.txt
print read;  --pede um tuplo de números ao utilizador e escreve-a na consola
soma := (read + read);
print soma;
```

Mude para o comando `./run` para testar o interpretador. Pode também fazer esses testes utilizando os ficheiros de entrada (`input*`) fornecidos. Por exemplo:

```
./run p3.txt < input-p3-1.txt
```

e) Acrescente uma instrução iterativa em que a condição de manutenção do ciclo é falsa se o tuplo estiver vazio, ou se for composto apenas por zeros, e verdadeira em todos os outros casos (ver programa `p4.txt`). [2 valores]

Esta instrução deve poder aplicar-se a um qualquer número de instruções (incluindo, zero).