1a. Tool for computation.
b. readability: characterizes the understandability of program written in the language.
writeability: characterizes the easiness to express computations.
reliability: characterizes the stability of programs from execution to execution.
c. Example using Java;
Readability: + : use of classes for modularity, facilitates the understanding of programs.
- : some syntactic rules make programs hard to read by the use of ‘{’, ‘}’, ‘=’, the dangling else,
try-catch syntax, the need to use or not use blocks in some constructs. etc.

Writability: + Writing composed queries such as obj.f1().f2(), the existence of interfaces, and
abstract classes.
- : the need to catch unexpected exceptions that have been declared in the server code. The need to
be aware of the existence of aliases. Separation between primitive values and objects.

Reliability: + : exceptions.
- : aliases proliferation, no automatic check pre- and post- conditions.

2. procedural: FORTRAN90, Pascal, C,
Object Oriented, C++, Eiffel, Java
Functional: Lisp, Scheme, Haskell.
Logical: Prolog, VeryLog, Mercury

3.
a. i. Syntax: the rules of the constructs that dictate the external appearance of the language, and
the symbols used to write in the language.
ii. Semantics: the meaning of the legal constructs.
b. BNF for syntax, A natural language (English) for semantics.
c. BNF can only describe context free languages, while programming languages are context sensi-
tive.
The use of a natural language give rise to ambiguities and incompleteness.
d. Use Attribute grammars for the syntax description, and Denotational semantics or other formal
method for the semantics.

4. Note: a low score is better.
program training: low; but training for OO can be high. 20.
program writing: because of OO its supports program decomposition that can be done by teams.
cost here is low. 20.
compilation: files are not large and compilation is time is low. cost: low. 15.
execution: its interpreted so much slower; can be improved with JIT compiler; cost can be very
high. Memory usage is monitored by a garbage collector. No extra burden on memory usage. 40.
Implementation system: no special requirements outside from installing a JRE. 5.
maintenance: supporting OO makes programs much more amenable for maintenance which will
increase with well designed programs. Cost here is rather low. It will not get higher than any other
language if implementation structure is poor. 20.
reliability: Because of exceptions it can increase the detection of unsafe programs. So cost its as good all languages with exceptions. But the use of aliases and no support pre- and post conditions checking can hinder it. 30.

5. Expression support:
   a. a rich set of primitive expressions.
   b. a way to combine primitive expressions to form more complex expressions.
   c. a way to abstract expressions.

6a. Parsing: A tool to recognize language sentences; sentences could be statements in particular and constructs in general.
   b. abstract syntax tree and it removes all syntactic sugar.
   c. A construct in a programming language which may have more than one meaning. It is detected by the production of more than one parse tree.

7a.

b. Revise the grammar to include that every “if” construct must be finished with an “end if”.
   <ConditionalSt> ::= if <ConditionalForm> endIf
   <ConditionalForm> ::= (<Boolean Exp>) <Stm> | (<Boolean Exp) <Stm1> else <Stm2> |

   c. For any given construct the parser should only save all the components in the construct sufficient for its meaning; thus all unnecessary tokens are removed. Those include punctuation marks, and most keywords.

8a. compiler: it produces files in secondary storage during translation. execution of those files is faster than using interpreters, as they code may be optimized during translation, but it does not impose any extra requirements on memory usage outside what is needed by the run-time system and the program.
   interpreters: it does not produce files to secondary storage; but at run time source code translation, execution and memory management requires much more time and memory outside from what the program per-se requires. In other words, compilers have a large footprint for secondary storage, while interpreters have a large footprint in memory and execution speed.
b. Procedural or OO: variables also have the assignment operator which can change its contents and produce side-effects. Functional programs do not have assignment statement.

c. Alias: a new name to an already existing memory location allocated to program.

d. It’s introduced by mechanism that create references to existing memory locations allocated to program.

e. In procedural or OO: aliases is a headache; programmers or languages provide mechanisms to reduce it. Assume a language such as Java, or C with pointers:
If x and y are variables being references to the same memory location; by changing the contents of x, it will change it for y.
In functional languages: no because even having aliases, you cannot change memory’s contents.

9.


Assembly language: code is written using mnemonics for memory addresses and operator codes. Examples the assembler for any existing processor: Pentium 7800 assembler, Motorola 1600 assembler.

High Level languages: code is english-like structured around statements and expressions. FORTRAN90, C++, Java, Haskell.

b. mnemonics for memory locations, memory management for allocation, diagnostics after a program crash.

c. Data types, rich expression formation, rich control structure set, abstraction for blocks of code under subprograms, parameter passing mechanisms.

d. Abstraction.

e. FORTRAN II and several of its successors. it abstracted the concept of compilation, making the code more portable than previous languages. It abstracted memory storage into INTEGER, REAL, and CHARACTER. It abstracted several forms to deal with conditionals.
Pascal: It abstracted data type, it provided for recursive subprograms, it abstracted memory management.

10. a. Immutable memory.

b. function composition. More explicitly, a program is decomposed into a set of functions definitions and the actual program results from a sequence of function compositions.

c. Although there is no direct mechanism for side effects, you can simulated via copying, parameter passing and recursion.

d. See next page. It is written in Haskell, the language of choice by every one who took the test. The actual program is written using literate programming. You can cut it and paste it to a file with extension *.lhs. Althoug I wrote processOne function, you were only required to write pair-Counter function.
We will assume Student type is String.

```haskell
> type Student = String
> type StudentList = [String]
> type NumberedStudentList = [(String, Int)]

> processOne :: Student -> Int
> processOne name = (length name) + ord(head name) +
>                    ord (last name)

> pairCounter :: StudentList -> NumberedStudentList
> pairCounter [] = []
> pairCounter (student:students) =
>   (student, 100) :  processRest students (processOne student)

> processRest:: StudentList -> Int -> NumberedStudentList
> processRest [] _ = []
> processRest (student:students) number =
>  (student, number): processRest students (processOne student)
```

{-
Hugs interactions:

pairCounter []
[]

Main> pairCounter ["Peter"]
["Peter",100]

Main> pairCounter ["Peter", "Paul"]
["Peter",100],("Paul",199)]

Main> pairCounter ["Peter", "Paul","Mario", "Jose", "David"]
["Peter",100],("Paul",199),("Mario",192),("Jose",193),
  ("David",179)]

-}