Assignment 2: AP++

An Interpreter is a computer program which can read an Interpreted Language and after processing it, individually executes each statement and expression on the local machine. A simple Interpreted Language, AP++ is defined as follows:

Rule	Description
Type ::= τ ε {int, bool, alpha}	Allowed Data types are either int(integer), bool(boolean), or alpha(quoted string)
bool ::= t ε {tt,ff}	Bool refers to the set of tt(true) and ff(false), only
int ::= n ε ℤ	Int refers to the set of signed Integers(64 bit)
alpha ::= s ε /(["'])\w+\1/	Alpha refers to the set of quoted words
Id :== {a, ab, ab1,…}	Identifiers are the variable names, which always start with an alphabet and can contain alphanumeric and the special characters $,_, *, and #
E ::= k|Id| E bop E | uop E	Each expression can contain a Identifier, a value, binary operations on two expressions or a unary operator on one expression. While there is no explicit restriction in AP++, you may assume the operations are not valid for “alpha”.
k ::= t | n | s	The set of values can either be a bool, an integer, or a string.
bop ::= {+, *, and, or, /, ^, ==, >, <, ><}	The set of binary operators are +(add), *(product), and(binary AND), or (binary OR),

	/(difference), ^(binary XOR and powerOf), ==(comparison operator), >(greater than), <(less than), ><(not equal to)
uop ::= {-,not}	The set of unary operator contains – and not with both operators having the same effect, i.e. change the sign for Integers and convert the Bools from true to false and false to true.
D::=nil |const id: τ=E |var id:τ=E | var id:τ=y | var id:τ	*Declaration of Identifiers
C::=skip|id:=E|C0;C1|if E then C0 else C1 | while E do C	List of Commands
print E	Prints an expression on a new line

where τ, t, n, Id are meta variables.

This language is composed of both Expressions and Declarations.

All Expressions are separated by either a ";" (semi-colon) or a "/n" (line break)

*Identifiers are either Free or Bounded. In case of Bounded Identifiers an auxiliary function BI will return the bounded identifier from a declaration. Formally:

BI(*id:=E*) = BI(skip) = BI(nil) = BI(Exp)=∅

BI(const id:τ=e)=BI(var id:τ=e)={id}

BI(*var id:τ=y) = BI(var id:τ*) = id

BI(C0;C1)=BI(C0) ∪ BI(C1)

BI(if E then C0 else C1)=BI(C0) ∪ BI(C1)

BI(while E do C) = BI(C)

Similarly for free Identifiers, we define FI:{E,D}→Id, to find out the free Identifiers from an expression or declaration.

FI is automatically defined by FI=I\BI (Free Identifiers = Identifiers – Bounded Identifiers)

Formally:

FI(nil) = FI(k)=∅

FI(id)=id

FI(e0 bop e1)=FI(e0)∪FI(e1)

FI(uop e)=FI(e)

FI(const id:τ=e)=FI(var id:τ=e)=FI(e)

FI(*var id:τ=y*) = {y}

Using these rules a user can write a simple program which will be interpreted by a special Interpreter written by you in Scala.

Example

1. Correct Program

```
var x:int;

const y:int=2;

x = y+1

print x

print y
```

The output of this program is

```
3

2
```

2. Correct Program

```
var x:int = 10;

var y:int;

y = 0;

While x>y do y = y+1; print y;
```

The output of this program is

```
1

2

3

4

5

6

7

8

9

10
```

3. Correct Program

```
var x:int = 10;

const y:int = 9;

if x>y then y = y+1 else y=y-1;
```

The output of this program is

```

```

4. Correct Program

```
var x:bool = tt;

if x then print “true” else print “false”;
```

The output of this program is

```
true
```

5. Incorrect Program

```
Let 0=x;

Y=x;

Let x = (x==y)

x = x+1
```

This program uses incorrect assignment operator, assigns a value to Y, without declaring it and adds the integer 1 to a bool.

Task

- Create an Interpreter for AP++ imperative language using the functional paradigm in scala, which will produce a tool called APPP.
- Differentiate between functional and imperative Programming Paradigms
    - You must produce a tool named APPP, which can read a file containing a program written using AP++ and executes it, producing a result.
    - APPP should identify each expression and declaration.
    - APPP should be able to execute the program and should produce an appropriate output.
    - APPP should be able to properly cater for side-effects, encountered while changing the values of the variables.
    - APPP must be written using the Functional Paradigm in Scala.
- Apply various code management tools and techniques in C++ and Java.
    - APPP must output an error, if the program has any syntax errors, identifying the line number and the error.
    - Write unit tests for your program, producing APPP.
    - Write unit tests for APP testing each rule, independently.

- Write unit tests for APP testing the sample example above.
- Your program and APPP should be properly documented, which explicit, self-explanatory comments before each function/method.
- The result of APPP's memory and processor profiling should be included in the description document.

Deliverables

- Upload the full solution on GitHub in a public repo.
- Your upload will consist of the following files:
  - Compressed Source Code
  - APPP executioner
  - AP++ sample code files
  - Test Cases
  - Description document, with separate sections on Introduction, Novelty in your approach (what did you learn?) and How to run your application? Screen shots of sample runs full filling the requirements, profiling data..
- All your files should be named as follows: **Filename_YourName_Registration No._Class_Section**

Grading Criteria

This Assignment will be graded on the following rubric:

Criteria	Weight	0	1	2	3	4
Analysis and approach	0.5	Unable to plan and set objectives for the realization of the lab task.	In between	Adequate analysis of the project. Objectives have been set, but strategies to follow are not clearly stated.	In between	Complete analysis of the task has been done. Objectives have been set. Strategies to follow have been defined.
Clarity	1	Student has no knowledge of both problem and solution. Cannot answer basic questions	Student does not have grasp of information; student cannot answer questions about subject matter	Student is uncomfortable with information and is able to answer only rudimentary questions	Student has competent knowledge and is at ease with information. Can answer questions but fails to elaborate.	Student has presented full knowledge of both problem and solution. Answers to questions are strengthen by rationalization and explanation
Completeness and Accuracy	2	The system failed to produce	The system execution led to inaccurate or incomplete	In between	The system was correctly functional	The system was correctly functional and all of the

		the right accurate results	results. It was not correctly functional or not all the features were implemented.		and most of the features were implemente d	features were implemented
Coding Standards(Packa ging, Unit Tests, Profiling, Debugging, Testing)	0.5	Coding standards, best programming practices are not followed. Students cannot understand the code.	Coding standards, best programming practices are not followed.	Coding standards, best programmin g practices are rarely followed.	Coding standards, best programmin g practices are followed appropriatel y	Coding standards, best programming practices are followed extensively
Originality	0.5	Most part of the working product is copied	Working product is uninspired and straightforwar d work with little to no creative potential.	Working product has some potential for making a creative contribution.	Working product has some creative /original /inventive element and a potential for making a creative contribution	Working product has several creative /original /inventive /innovative elements and a clear potential for making a creative contribution.
Code Documentation	0.5	Code has not been commented or only small meaning-less comments are present.	The documentatio n is simply comments embedded in the code and does not help the reader understand the code.	The documentati on is simply comments embedded in the code with some simple header comments separating routines. No sample run is shown.	The documentati on consists of embedded comment and some simple header documentati on that is somewhat useful in understandi ng the code. Sampl e run is clearly shown covering some objectives.	The documentation is well written and clearly explains what the code is accomplishing and how. Sample run is clearly shown covering all objectives.