Fundamentals of Programming Languages

Task One

Introduction

In this module you are learning about the design and implementation of programming languages. Many types of programming language are available, each of which either has a different purpose or lets programmers develop a different style of solution. In this first assignment I am asking you to design a small, limited general purpose language.

This assignment is worth 40% of the module's marks.

The task

1. Create a formal grammar for the simple procedural language that is described at the end of this specification.
2. You may extend the language as you see fit but do not use any of the features that are on the banned list given at the end of this specification. Extensions will not be marked unless an acceptable grammar is provided for the base language.
3. Your grammar must be a valid Antlr 4 style-grammar.
4. Give examples of the usage of your language: try to show all elements of the grammar across your examples. You must prepare at least four examples of your own, explaining each of them in detail. Of course these will not compile - we will create a compiler for the language in the next task.

Learning outcomes

This assignment partially covers the following learning outcomes:

- Understand the principles which lie beneath programming language design,
- Read code written in a variety of languages,
- Design formal representations of language constructs,
- Discuss programming languages knowledgeably.

Submission

You must submit your work in plain text files not in Word or PDF format. Please create three files: one containing the grammar, the second containing your sample programs and the third containing the explanations of those programs. You must upload a zip file that contains all of your documents to the assignment handler on Blackboard.

In addition, you must prepare a video in which you:

- Briefly discuss the design of your language and grammar.
- Demonstrate each item from the marking scheme using IntelliJ IDEA's ANTLR plugin.
- Walk through some examples with which you are most pleased.Your video must last no longer than five minutes.

Content after the five-minute point will not be marked. The video will ideally be a desktop recording or screen capture. If you do not have access to suitable software you may record the video using a camera or phone. In this case it is vital that the screen is easily readable. The video must be in a platform-independent format such as MP4. If I can't view it, I can't mark it. You will receive instructions on how to upload your videos nearer to the submission date

The deadline is **3 p.m. on Thursday 17th December, 2020**.

Marking scheme
Your work will be marked using a grade-based approach that is described in a separate document available on the module's Blackboard site and at http://tinyurl.com/zkej95m.

The following table gives an incomplete list of the things that you should demonstrate in your work.

	Marks available	We are looking for
Structure of the language	20	The language is described fully Rules are sensible Clarity in the rule definitions including sensible naming A third party ought to be able to use your definition to create a reasonable compiler front-end The syntax is clean and sensible The language is not a direct copy of an existing one
Code structures	10	Code has blocks Iteration and selection are presented clearly Arrays are implemented in a conventional way
Variables	10	Local variables work Global variables are usable Data types are specified and can be used sensibly
Functions	10	The language has functions Arguments are passed by value Return values are correctly handled
Lexical rules	10	The rules are appropriate for the language All required characters and set are defined for the lexer
Parser rules	10	The rules describe a complete parser.

		The rules are written in a way which is clear and in which the meaning of each rule is made obvious A hierarchical structure is clear
Extensions	10	Useful extensions are provided and demonstrated.
Use of the grammar	5	The grammar is clear and readable Grammar files are in an Antlr 4 compliant format
Sample programs	15	You have created your own sample program which demonstrates comprehensive coverage of the language

A Description of the Language

This is a small, simple block-structured, imperative language based around function calls. It has three data types and just one collection type. The language is not object-oriented. You may implement any syntax that you wish, be that C-style, Python-ic, Java-ish or based on something more exotic such as COBOL or Perl.

Feature list

- The language uses only characters from the ASCII set. It is not Unicode-compliant.
- The only data types are: floating point numbers, character strings, boolean values.
- The language has a “sensible” set of operations for each data type. You get to decide what those operations are.
- Values can be assigned to named variables.
- Global variables are allowed.
- Constants are permitted.
- Code is structured using blocks.
- You may use grouping to controller operator precedence.
- Use functions. A function receives values as arguments and (optionally) returns a value.
- Use pass-by-value. The arguments cannot have their value changed except through assignment of the function’s return value.
- Functions can return only a single value.
- Variables are scoped to the function that contains them and hoisted to the top of it. This means non-global variables are only available inside a function and that they are available throughout it.
- The only permitted collection type is a simple one-dimensional array.
- The language includes features for repetition and for iteration across the array type.
- Programmers must be able to select alternative paths through the code by using Boolean logic.

Banned list

- Do not implement classes or other abstract data types. This is not an object-oriented language.
- Do not implement access control by making variables public, private etc.
- Do not add further collections such as sets or hash-maps.
- Do not implement low-level memory management. No pointers, no pass-by-reference.
- Do not cast data items from one type to another.
- Do not include functionality for accessing files, databases etc. Input comes from the keyboard. Output goes to the screen.
- Do not have macros.
- Programs must not be formed by bringing together code from a number of files.