Fundamentals of Programming Languages

Task Two

# Introduction

In this assignment you must implement a compiler and use it to convert programs written using the language that you defined in the first assignment into JavaScript.

This assignment is worth 60% of the module’s marks.

# The task

In assignment one you specified a simple imperative programming language and wrote some sample programs that demonstrated the language. In this assignment you will write a compiler for that language. The compiler will translate those programs into valid JavaScript.

You may use either the visitor or listener implementations. Each will be marked in the same way and to the same standards.

1. Using ANTLR and Java, write an application which builds a parse tree for valid programs. You should write code which demonstrates that the tree can be walked.
2. Extend your application so that:
   * a user can search the parse tree for language elements
   * the parse tree is displayed (you must write your own display code NOT use grun -gui)
   * subtrees can be displayed
3. Transform simple programs into JavaScript. Demonstrate your compiler using the sample programs that you wrote as part of the first assignment.

Two sample grammars are available on the Blackboard site. If you feel that your own grammar lacks sufficient richness to be used in tackling this problem then you may use one of the provided solutions instead. You must, though, write your own sample programs.

# Learning outcomes

This assignment partially covers the following learning outcomes:

* Understand the principles which lie beneath programming language design,
* Design formal representations of language constructs,
* Implement the basic components of a simple compiler.

# Submission

You must submit a single zip file containing all of your code plus instructions for using it and sample programs through the SHU assignment handler on Blackboard. In addition you must prepare a video in which you:

* Show your compiler building each of your sample programs from Semester One.
* Discuss the design and implementation of the compiler.
* Demonstrate each item from the marking scheme.
* Walk through some of the sections of code with which you are most pleased.
* Clearly identify the sample grammar if you are not using your own.

Your video must last no longer than ten minutes. Content after the ten 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 readable.

The video must be in a platform-independent format such as MP4. If I can’t view it, I can’t mark it. Put the video onto Google Drive or as a private video on YouTube or Vimeo, and include a link to it in your instructions document.

The deadline is **3 p.m.** on **Thursday 15th April, 2021**.

# 

# 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 is an incomplete list of the types of thing that you should demonstrate in your video.

|  |  |  |  |
| --- | --- | --- | --- |
| **Aspect** | **Marks available** | **At pass level** | **At distinctive level** |
| **Creation of a parse tree** | 5 | A simple application is provided which may   * make some progress towards parsing the input * work on one type of valid program * accept code from the command line only | * A range of valid programs can be parsed. * Programs are read in from files. * The parse tree can be a (valid) subset of an entire program |
| **Displaying the parse tree** | 25 | * Some attempt is made to print the tree. This may be as a simple string. * Very simple trees can be output | * Trees are displayed successfully for most valid, and parsable, programs. * The output may be a useful intermediate representation such as   + S-expressions   + JSON documents   + XML structures |
| **Searching of the parse tree** | 20 | Some attempt is made which may   * accept a search criterion * build an appropriate data structure to hold the result * start to search the tree * attempt to return a result | * Search criteria can be entered and chained * Most valid programs can be searched * Searches return results where those results are available * Results are presented clearly and usefully * No matching data is still treated as a valid result |
| **Displaying subtrees** | 20 | * A subtree may be displayed – often from the root only * There is code to find the start of a subtree | * Subtrees are found and displayed * Code for searching and displaying trees is re-used sensibly |
| **Transform a program into JavaScript** | 30 | Some attempt is made to produce output. This may include   * + creation of suitable templates   + mapping of templates to rules   + modification of the grammar to support the necessary set of rules | * A range of valid programs can be transformed into valid outputs that can be executed. * Code is clear and appropriate * Correct structures are used * Errors or edge-conditions are handled gracefully |