Languages and Compilers *CM4106*

Coursework Overview (Semester 1)

Date of Assessment Release	Coursework will be released in Week 2 of Semester 1
Submission Overview	Element 1: Scanner, parser and implementation report Element 2: Semantic analyser, code generator and implementation report
Submission Date	Element 1: Friday 26 October before 23:59 Element 2: Friday 30 November before 23:59
Submission Method	All coursework submissions are to be submitted via CampusMoodle
Number of Copies to be Submitted	Only 1 digital copy of each submission should be made. No physical copies are required
Date for Release of Marks	Marks will be released 2 weeks after the final submission
Date for Release of Feedback	Feedback will be given within 2 weeks of each submission
How Feedback will be Returned	Feedback will be returned via email and CampusMoodle
Learning Outcomes	This coursework covers learning outcomes 4 & 5 for this module.
How Grades are Calculated	Please see the **Module Grading Grid**

The main purpose of the coursework is to allow you to demonstrate that you have developed the skills needed to develop a code generating compiler for a given language.

The compiler you develop will be tested for completeness against a set of unseen scripts, with a known output, to check its completeness. This will mean that you have to develop your compiler to work completely for the specified language.

The language is described fully in a separate document available on the module Moodle page.

The coursework is to be undertaken individually and will include a report asking you to describe the reasoning for the implementation you have chosen. Coursework will be checked for plagiarism and collusion as per the University's standard procedure.

1 Coursework Sections Outline

General Requirements

The two submissions for the coursework are detailed below. It is the aim of the coursework to create a complete working compiler for the given language and platform. You should adhere to standard software engineering best practices at all times. The overall design of the compiler is up to you as the developer but you should ensure that the system is developed in a modular fashion as this will let you test each component in isolation. I would also advise that you maintain your code using a recognised Version Control System (e.g. GitHub, Bit Bucket). **Use a private repository to ensure the privacy of your code.**

As above your compiler will be tested with a set of scripts of known output, not all these scripts will be designed to compile and as such your compiler system should be able to report meaningful errors to the user for both syntax and semantic errors.

You are free to develop your system using C# for .NET or .NET Core. But you should make sure you make it clear which framework you have used on your submission.

Submission

Both submissions will be online only via Moodle, you should provide a zip archive containing your code and your implementation report. Please name your archive using your name and student number. You should ensure that your application runs before submission!

1.1 Element 1 - Scanner & Parser (40% of Coursework Grade)

The first submission will constitute the two primary components of the front end of your compiler.

Scanner implementation – 15%

Your Scanner (Lexical Analyser) should be developed to read in a source file written in the given source language. The characters in the source file should be compiled into a sequence of recognised language **tokens**.

During this process your scanner should fulfil the following basic compiler requirements

- Identify and remove whitespace.
- Identify and remove language comments
- Identify and produce errors for unknown characters in the language
- Identify and produce errors for unterminated character literals

Parser implementation – 20%

Your Parser (Syntax Analyser) should work on the sequence of tokens created by the scanner. These tokens should be grouped into sentences, creating a set of instructions. The instruction set will represent the purpose of the program defined in the source code.

Your Parser should be able to identify Syntax Errors in the source program and produce errors for instructions that do not conform to the language definition.

On completion of the Parser stage your compiler should maintain an Internal Representation of the instruction set and write out an instruction set, in the correct order, to the console.

Implementation Report – 5%

In addition to the two components above you should also submit an implementation report describing how you have implemented the compiler. Your report should a description of how you have covered the requirements of the Scanner and Parser detailed above, including how you have transferred data between the components. Your implementation report for this stage should be no more than **2000 words** plus appendices.

As an appendix to your report you should include any source programs written in the given language that you used to test your Scanner and Parser.

1.2 Element 2 - Full Compiler (60% of Coursework Grade)

The second submission will perform semantic analysis and generate target code.

After the first submission, you will be provided with the code for a fully correct scanner and parser. It is recommended that you use these in the second submission.

Semantic Analysis – 25%

You should implement a Semantic Analyser which work with the instruction set provided by the Parser. The main purpose of the semantic analyser is to identify semantic errors in the source program. Your parser should fulfil the following requirements;

- Create and maintain the symbol table
- Check declarations and variable use
- Perform type checking on the source program
- Enforce program scope

The Semantic Analysis component should at this stage produce a semantically correct instruction set.

Code Generation – 25%

Finally, you should implement your Code Generator component. This component should take the instruction set from the semantic analyser and generate the code required to perform these actions on the TAM platform used in lab practical sessions. For this coursework you will **not** be required to develop Optimisation steps.

Implementation Report – 10%

In addition to the two components above you should also submit an implementation report describing how you have implemented the compiler. Your report should provide a description of how you have covered the requirements of the Semantic Analyser and Code Generation steps detailed above.

You should also include in this document an explanation of any aspect of your compiler that you have identified as non-functional or not-implemented. You should include any reasons or suggestions to why this feature of your compiler does not function. Your implementation report for this stage should be no more than **3000 words** plus appendices.

As an appendix to your report you should include any source programs written in the given language that you used to test your entire compiler.

2 Marking Grids

The following grids outline the requirements for each grade for the two submissions

Element Value	Element Description	A	B	C	D	E	F	NS
40%	Scanner & Parser Development	A fully functioning scanner has been developed which functions flawlessly with all test scripts and reports errors in incorrect scripts. A fully functioning parser has been developed which functions flawlessly with all test scripts and reports syntax errors in incorrect scripts. An excellent standard of code is maintained throughout the submission. Code is extremely well structured and commented throughout. A complete and full implementation report has been provided that explains and justifies the approach taken to the development. Full evidence of test scripts and testing has been provided.	A fully functional scanner has been implemented and complete test scripts are scanned successfully. Not all scripts with lexical errors are reported correctly and errors may not be helpful to a developer. A fully functional parser has been implemented and complete test scripts are parsed successfully from the scanner. Not all scripts with syntax errors are reported correctly and errors may not be helpful to a developer. A very good code standard is maintained throughout the submission. The code is well structured and commented throughout. A good implementation report has been included which does justify the approach taken. Limitations are mentioned and generally explained. Appropriate test scripts are included and the components have been mostly well tested.	A mostly functional scanner has been implemented but one or more of the required properties for successful lexical analysis have been not been included and the scanner fails to successfully scan all test scripts. A mostly functional parser has been implemented but one or more of the required properties for successful syntax analysis have not been included and the parser fails to successfully parse all test scripts. A good code standard is mostly maintained throughout the submission. A mostly component based structure is used and the code is generally well commented. A good implementation report has been included which does justify the approach taken. Limitations are mentioned but not fully explained. Appropriate test scripts are included and the components have been mostly well tested.	A partly functional scanner has been implemented but not all of the required properties for successful lexical analysis have been included and the scanner fails in a major regard. A partly functional parser has been implemented but not all of the required properties for successful Syntax analysis have been included and the parser fails in a major regard. Overall code is below standard with some structure but is limited by the work completed. Commenting is brief and unhelpful. Brief implementation report that makes some attempt to justify work completed. Some evidence of testing has been provided with mostly appropriate test scripts.	Non-functional scanner with some attempt at lexical analysis implementation. Non-functional parser with some obvious attempt at syntax analysis implementation. Overall code is of a poor standard with little obvious structure or commenting. Brief implementation report that makes some attempt to justify work completed. No test scripts o evidence of testing has been supplied.	Non-functional scanner with very little obvious attempt at lexical analysis implementation. Non-functional parser with very little obvious attempt at syntax analysis implementation. Non-substantial implementation report that makes little attempt to justify work completed.	No Submission

60%	Final Compiler	A fully functioning semantic analyser has been developed which functions flawlessly with all test scripts. A fully functioning code generation component has been developed which functions flawlessly with all test scripts and generates working code on the target system. An excellent standard of code is maintained throughout the submission. Code is extremely well structured and commented throughout. A complete and full implementation report has been provided that explains and justifies the approach taken to the development. Full evidence of test scripts and testing has been provided.	A fully functional semantic analyser has been implemented but one of the required properties for AST annotation have been not been included and the semantic analyser fails to identify all semantic errors. The symbol table has been created and is correct for type checking. A mostly functional code generator has been implemented but one or more of the required properties for successful target code generation have been not been included and complete code is not generated for all example scripts.. A good code standard is mostly maintained throughout the submission. A mostly component based structure is used and the code is generally well commented. A good implementation report has been included which does justify the approach taken. Limitations are mentioned but not fully explained. Appropriate test scripts are included and the components have been mostly well tested.	A mostly functional semantic analyser has been implemented but one or more of the required properties for AST annotation have been not been included The symbol table has been created and is mostly correct for type checking. A mostly functional code generator has been implemented but one or more of the required properties for successful target code generation have been not been included. A good code standard is mostly maintained throughout the submission. A mostly component based structure is used and the code is generally well commented. A good implementation report has been included which does justify the approach taken. Limitations are mentioned but not fully explained. Appropriate test scripts are included and the components have been mostly well tested.	A partly functional semantic analyser has been implemented but not all of the required properties for successful annotation of the AST have been included and the analysis fails in a major regard. An attempt at symbol table implementation has been made. A partly functional code generator has been implemented but not all of the required properties for successful code generation have been included and the compiler fails in a major regard. Overall code is below standard with some structure but is limited by the work completed. Commenting is brief and unhelpful. Brief implementation report that makes some attempt to justify work completed. Some evidence of testing has been provided with mostly appropriate test scripts.	Non-functional semantic analyser with some attempt at implementation. No attempt at symbol table. Non-functional code generator with some obvious attempt at implementation. Overall code is of a poor standard with little obvious structure or commenting. Brief implementation report that makes some attempt to justify work completed. No test scripts to evidence of testing has been supplied.	Non-functional Semantic Analyser with very little obvious attempt at implementation. Non-functional code generation with very little obvious attempt at implementation. Non-substantial implementation report that makes little attempt to justify work completed.	No submission