Translator Design - Exercises

1 Exercises

1.1 C++

1. Explain in 2-3 paragraphs the differences between manual memory management, reference counting and garbage collection.
2. What kind of graphs can be represented if the neighbouring vertices are represented using a list of reference-counted pointers? Think about directed, undirected and cyclic graphs. Explain in detail.
3. What happens if we use the `emit` method from Section **??** to write to a file on a little-endian machine which we read on a little-endian machine?
4. What operations are allowed on the STL containers mentioned before and what is their time complexity?

1.2 Language

1. Add another primitive to the interpreter, `print_newline`. Write a test script that reads two integers and prints their sum properly followed by the newline character.
2. Would a `print_char` function be better? Discuss what else should be added to the language for such a function to be useful.
3. Name and discuss in a few sentences each at least 5 different language features which are available in most languages but are absent in Imp.

1.3 Lexer

1. Identify one performance issue in the lexer. Discuss in a few sentences how you would improve it.
2. Extend the lexer to accept operators for multiplication, division and subtraction.
3. Add the commonly used comparison operators to the language.
4. Modify the `Token` class to represent integers and extend the lexer to tokenise them. Argue in a few sentences why there is no need to represent negative numbers at this point. Define an arithmetic operator which can help represent negative numbers. Identify an edge case. How would you handle it?

1.4 Parser

1. Define the AST nodes to represent the operators and constants added to the lexer.
2. What is the priority of the operators defined so far? Are they left or right associative?
3. Extend the parser with syntax to allow users to control the priority of operators. Is there a need for a separate AST node?
4. Extend the parser to accept the newly added arithmetic and comparison operators, as well as the integral constants.

1.5 Interpreter & Code Generator

1. Identify, describe and fix the bugs in the PEEK and ADD opcodes.
2. Define an opcode to push a constant integer to the stack.
3. Define the opcodes for the newly added operators. Do they need any additional arguments?
4. Translate the new AST nodes to bytecode. Do not forget to add tests.