Compilers Coursework 1: WACC Language Specification

COMP50006 - Compilers
Department of Computing
Imperial College London

Summary

You have been provided with a detailed specification for a While-like language, called WACC, as well as a large number of example WACC programs and a reference WACC compiler. Your task is to examine a number of faulty WACC programs and identify the number of errors, and their causes, in each case.

The purpose of this exercise is to prepare you for the WACC compiler project by getting you to read the language specification document in detail, and familiarise yourself with the reference compiler.

You may work in groups of up to 4 to complete this exercise.
We recommend doing so in your WACC groups, but this is not required.

Details

In order to write a compiler for a language it is important to be clear what the definition of that language is. This definition should convey not only the *syntax* of the language (the symbols, keywords and statements that make up a valid program), but also the *semantics* of the language (the meaning/behaviour of a valid program). Consider Java. The Java syntax for declaration of a primitive type is:

```
TYPE IDENTIFIER SEMICOLON
```

Thus '`int x;`' is valid and '`char;`' is invalid. The semantics of Java[1] dictate further rules regarding the scope and meaning of the declaration '`int x;`'. For example, a redeclaration of `x` in the same scope is defined as a compile time error. Taking these two combined factors we can be sure that,

```
int x;
int y;
int z;
```

is a valid Java program and that,

```
int x;
int x;
int;
```

is an invalid Java program, because every variable declaration requires a variable name.

You will be provided with a WACC Language Specification and a number of example WACC programs with their expected output. To begin this exercise, you should look through the example programs and try to work out what each does. In doing this you should also consider what constitutes a valid or invalid program.

[1] Available for light bedtime reading at http://java.sun.com/docs/books/jls/third_edition/html/j3TOC.html

Submit by 19.00 on Monday 20th January 2025

What To Do:

1. **Get the files provided for the exercise:**

   We have set up a public Git repository on the department's `GitLab` server that contains a large number of example WACC programs. You can clone a copy of this repository into your local workspace with the following command:

   ```
   prompt> git clone git@gitlab.doc.ic.ac.uk:lab2425_spring/wacc-examples.git
   ```

   or, if you haven't set up SSH keys on GitLab, then you can use:

   ```
   prompt> git clone https://gitlab.doc.ic.ac.uk/lab2425_spring/wacc-examples.git
   ```

   (which will prompt you to log in with your normal college username and password).

   Note that you can (and should) create your own example programs, but you cannot directly push any changes back to the `wacc-examples` repository. Merge requests for the tests are welcome, but won't necessarily be reviewed or accepted during the project.

   This repository contains numerous example WACC programs, a script that gives you command-line access to the reference compiler and supporting documentation that should help you with this exercise. In particular:

```
wacc_examples
├── invalid ..................................... examples of ill-formed WACC programs
│   ├── semanticErr ............................. example WACC programs with semantic errors
│   ├── syntaxErr ............................... example WACC programs with syntax errors
│   └── whack ................................... example WACC programs with unclassified errors
├── valid ....................................... examples of well-formed WACC programs
├── refCompile .......... Script that provides command-line access to the WACC reference compiler
├── refEmulate ............... Script that provides command-line access to the WACC ARM emulator
├── WACCLangSpec.pdf ....................... Defines the syntax and semantics of the WACC language
└── WACCRefManual.pdf ............ Describes how to access/use the lab's reference WACC compiler
```

2. **Familiarise yourself with the reference compiler:**

   The lab's reference implementation of a WACC compiler can be found on-line at:

   https://wacc-vm.doc.ic.ac.uk.

   The user manual for the reference compiler can be found on-line and also in the `wacc-examples` repository.

3. **Familiarise yourself with the WACC language:**

   You should begin by reading through the provided WACC language specification and example WACC programs. From the specification and expected output of the example programs, you should be able to determine the language constructs of WACC and their behaviours.

   You will then want to test your understanding of these behaviours by writing your own programs and passing them to the reference compiler.

4. **Examine the faulty WACC programs and find why they are invalid:**

   The 10 example faulty WACC programs (`q1.wacc` - `q10.wacc`) in the `invalid/whack` directory are all incorrect, but they have not been classified into the correct sub-directories or commented with any explanation as to why they are incorrect.

   Using your understanding of the WACC language specification, you are to identify **all** of the syntax **and** semantic errors in these example programs, as well as their causes. You will then document all of these errors, their types, and their causes for each example program.

   During this exericse, you should assume that all syntax errors are fixed in the most obvious way for the purpose of identifying semantic errors (*hint: you might want to do this yourself and use the reference compiler to help you*). i.e. you must report the semantic errors in each faulty program, even if the provided program has syntax errors. You may state how you fixed the syntax errors if you believe there is any possible ambiguity in this process.

Documenting the Program Errors

Your final submission for this exercise will be a report on the program errors for the examples in the `invalid/whack` directory. This must be a PDF file.

For each example program in the `invalid/whack` directory, you should provide a list of **all** of the syntax **and** semantic errors. Each error should have a clear **type** (*syntax* or *semantic*), and a **brief explanation** of the cause of the error (*max 2-lines per error*).

You may work with any word processor or typesetting tool that can export or save to PDF format, so long as your submission is machine readable (note that we will *not* accept submission of scanned hand-written documents).

Note: no credit will be awarded for literally copying the output of the reference compiler. You are expected to explain the errors in your own words.

An Example Error Report:

Consider the following faulty example WACC program `q0.wacc`:

```
begin
    int x = ~~5 & 2;
char c = chr 9223372036854775807
 end
```

We could write the following error report for this example:

> The program `q0.wacc` has no semantic errors, but 4 syntax errors:
>
> - Bitwise operators, like ~ and `&` are not allowed in WACC.
> - The number `9223372036854775807` is the largest 64-bit number, but WACC's `int` literals are only 32-bit: this is a syntax error in WACC.

For the first bullet point, you could also say:

- ~~ is not a legal operator, and neither is `&`

- there are two bad ~ operators and a `&` operator

All of these would be considered correct error explanations, and the number of syntax errors would be 3 or 4 (depending on if you say ~ ~ are 2 errors or ~~ is 1).

However, ignoring the oversized literal, `chr 2147483647` would be valid both syntactically and semantically. So, saying "the argument to `chr` is too big" is not correct, as this is a *runtime error* and not a *compile-time* one. For this exercise, we are only interested in syntax and semantic errors. As such, the correct answers are the two bullet points above, and nothing extra (WACC is not indentation-sensitive either).

Submission

You should submit your error report on the example faulty programs, `WhackErrorReport.pdf`, as a group (of up to 4) to Scientia by 19:00 on Monday 20th January 2025.

Assessment

In total there are **25** marks available in this exercise.

There are **2 marks** for each of the example faulty programs `q1.wacc` - `q5.wacc` from the `invalid/whack` directory and there are **3 marks** for each of the example faulty programs `q6.wacc` - `q10.wacc` from the `invalid/whack` directory. The marks for each example will be awarded according to the proportion of correct errors identified in each case.

This exercise will constitute 50% of the marks for the Compilers coursework. We aim to return feedback on your work to you by Monday 3rd February 2025.