Second Year Compilers Exercise: The MAlice Compiler

Milestone 1

Summary

You are to provide a language specification detailing your understanding of the MAlice programs you have been provided with. This specification will provide the basis for the front-end of your compiler. A formal specification is not required. You should, however, aim to be accurate enough that were your specification to be given to a competent coder unfamiliar with the MAlice language they would be able to write a valid program.

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.

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

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

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

is an invalid Java program.

While you are not expected to provide a full formal language specification you may find it helpful for future milestones if you provide a BNF (Backus-Naur Form) grammar to describe the syntax of MAlice.

You are provided with a number of sample MAlice programs and their output. To begin this exercise you should look through these and try to work out what each does. In doing this you should also consider what constitutes invalid programs.

Some languages also include "Implementation Defined" behaviour in their specification. This can cause portability issues but allows a language designer to not have to design around the limits of any particular system. Consider, as an example, the number of bits in a byte. While you are no doubt used to a byte being precisely 8 bits, a byte historically merely described a collection of enough bits to contain the full character set of the system. The ANSI C standard defines a byte as:

> Byte — the unit of data storage in the execution environment large enough to hold any member of the basic character set of the execution environment. It shall be possible to express the address of each individual byte of an object uniquely. A byte is composed of a contiguous sequence of bits, the number of which is implementation-defined. The least significant bit is called the low-order bit; the most significant bit is called the high-order bit.

With the size of a byte providing the abstraction from bit level specifics, a char can be defined as a unit of storage one byte wide on all systems. This gives programmers a consistent expectation across systems, while leaving the actual details of implementation to the compiler writer.

You should make decisions regarding what features of the language will be "Implementation Defined" and include these in your specification. Consider carefully the trade offs you are making between code portability and system abstraction when you make these decisions.

You should also describe what has not been defined by the example programs. Details of the language which cannot be discerned should be noted in your specification along with a description of the assumption you would make as to their semantic meaning.

Submit by Thursday 28th October

What To Do

1. Form a group of up to three people. You will work on all phases of the exercise in this group. Since it requires action from all group members, also set up your group on CATE as early as possible.
2. Download the file `provided.zip` from CATE. In the `archaeology` folder of this archive there are:
   - Twenty example MAlice programs. Some are correct, others contain syntactic and semantic errors
   - x86 assembly code corresponding to the source program in `ex09.alice`
   - A file displaying the output of the compilation and execution of the examples
3. Read through the provided programs and work out the language constructs of MAlice from their expected output.
4. Create a Language Specification Document. This should be a maximum of four pages. It should specify the BNF for the language, and should explain clearly the semantics of the language as far as you understand it from the examples.

The purpose of this exercise is to prepare for the second milestone, which involves building a compiler for this language. You may, of course, start as soon as you like.

Additional Help Getting Started

Try defining the rules for arithmetic expressions first. Try asking yourself some of the following questions.

What are the operators that MAlice allows?

What meaning do these operators have?

What is the correct result for (5 + 5)?

What about (3 + 5 * 6)?

Should (6 / 0) be a compile time error, a run time error, or should it be allowed to cause a processor exception?

What data types will the operations be legal for?

What should the result of ("hello" + 5) be?

Is there a limit to the magnitude an arithmetic expression can evaluate to? What happens if this limit is exceeded?

Can you justify this limit on all systems, or is the limit a consequence of the execution environment?

What implications do your decisions have for your implementation?

Creating the Document

Produce your document as a text file or a PDF. For PDF you can use LaTeX, or a word processor that can export or save to PDF format. Do not print out your document and scan it as this produces very large files. Your submitted document is limited to 1MB. This file (produced in LaTeX) is 45kB!

Using LaTeX

- The provided file `malice_spec.tex` can be edited to produce your document if you wish. The file contains some examples of how to produce different types of output using LaTeX. To generate `malice_spec.pdf` from the source file, use the command

  ```
  pdflatex  malice_spec
  ```

- For a good introduction to using LaTeX see `http://en.wikibooks.org/wiki/LaTeX`, or there are many books and other websites available.

Submission

- First, you will need to set up your group on CATE. Nominate one person as group 'leader'. The leader should set up the group and add the other group members. Once this has been done the other members will need to go into CATE and confirm their participation.
- The group leader should then submit your specification as either `malice_spec.pdf` or `malice_spec.txt`.

Feedback

You will receive the mark and feedback on your work by **11th November**.

Assessment

This milestone will constitute 15% of the marks for the exercise.