CS24210 Syntax Analysis and Topics in Programming Languages

Edel Sherratt

About this Module

Your Aim: to become familiar with the concepts required for specifying and implementing programming lanuguages

Why?

Learning Outcomes – for the whole course

On successful completion of this course, you should be able to

- make effective use of compilers and other language processing software;
- apply the techniques and algorithms used in compilation to other areas of software engineering;
- understand the need for implementation independent semantics of programming languages;

How the course will work

- lectures, will require input from you
- scheduled practicals
- reading – web based materials and library
- own practical work

About the Course

- Syllabus – www.aber.ac.uk/~ dcswww/Dept/Teaching/Syllabus/1999-00/
- The order will change
- The content will vary from year to year
- The examination will be based on the course *as taught*
- For the highest marks, evidence of additional reading, and more specially, *coherent thought* about the taught material will be demanded
- This means that your work should be paced
- Please point out any problems with the course content or presentation quickly – during the lecture, if necessary

Useful sources of information

- www.aber.ac.uk/~ dcswww/Dept/Teaching/Courses/CS24210
- N.B. Bennett is not available - so please ignore the statement in the syllabus that it is essential to purchase that book
- Alternatives should be in the bookshops during the coming week
- Aho, Sethi and Ullman, *Compilers: Principles, Techniques and Tools,* ISBN 0-201-10194-7
- Fischer, C.N. and Leblanc, R.J. *Crafting a compiler with C,* ISBN 0-8053-2166-7 (this will be probably be changed to the Java edition for next year's course – which will affect its resale value)
- Wirth, *Compiler Construction*, ISBN 0-201-40353-6

Language Processing Software

- Editors – e.g. nedit, gvim, vi, sed, ...
- Compilers and Interpreters – e.g. javac, gcc, perl, bash, csh ...
- Text formatters – e.g. LaTeX, nroff ...

...and the Languages they process

- gvim: ascii text, programming languages, markup languages
- javac: java programs
- gcc: ANSI C programs
- LaTeX: text marked up with latex formatting instructions
- html: text marked up with html formatting instructions
- perl: perl scripts
- bash: bash shell commands
- csh: C shell commands

How do we learn to use Language Processing Software Effectively?

- Learn all of these languages and software systems individually?
- In a 10-credit module – 80 hours of your time???!
- How do we cope with new languages and tools?
- How do we make sure our knowledge is good for life?

Lasting, transferable knowledge and skills in Language Processing

- We need some practice
- and also some abstract knowledge – transferable skills –

Recall CS12220/12320 – abstraction as a tool for dealing with complexity. Read this before the next lecture. Think hard about what it means for the study of language.

Abstraction as a tool to support effective use of language processing software

- What kind of information do we need to keep in mind?
  depends on what we're thinking about
  - all languages: the fact that languages have structure;
  - Java: class structure;
  - perl: pattern structure, loop structure, ...
- What kind of information do we omit?
  - all languages: possible structures
  - Java: semicolons etc.
  - perl: * / < > . etc.
- Different abstractions for different purposes?
  - different levels of abstraction level (as above)
  - OR different perspectives (next slide)

Note - there are many other ways to use abstraction when thinking about formal language.

Three aspects of language

- syntax – form, shape, structure
- semantics – meaning
- pragmatics – use

Lexical errors in Java

What's wrong with

```
fruit = 7peach;
```

An identifier may not begin with a digit

Lexical errors in Java

How about

```
fred = _fred_value:
joan = _joan_value;
```

statement delimiter is ; not :

Lexical errors in Java

Or maybe

```
Abstract Class StackD {
  Abstract Object accept(StackVisitorI ask);
}
```

keywords 'abstract', 'class' and 'object' should contain only lowercase letters

Lexical errors in Java

Or even

```
my_counter = 7 * (loop_counter + 5o5);
```

the letter 'o' should not appear in a number

Lexical errors in Java

What elements of the Java programming language were affected by these lexical errors?

- identifier
- delimiter
- keyword
- number (unsigned integer)

What constitutes the lexis of a programming language?

identifiers, delimiters, keywords, numbers (of various kinds), comments, ...

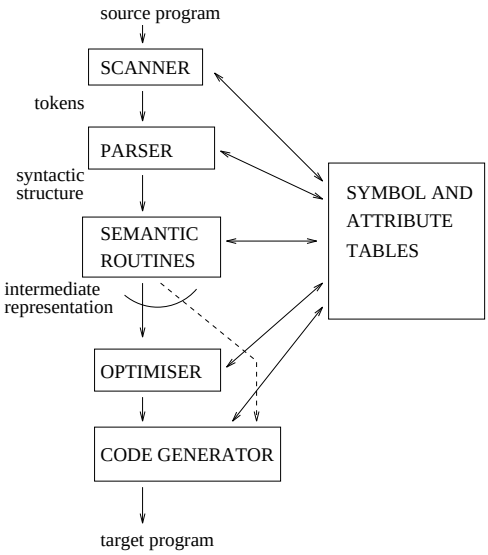
What part of the compiler catches lexical errors?

the scanner

How do we specify the lexis of a programming language?

We use regular expressions – we'll come back to this later.

Structure of a Compiler



Main Jobs of a Compiler

ANALYSIS of the Source Program

- lexical analysis
- syntactic analysis
- context checking

SYNTHESIS of the Target Program

- generation of intermediate representation
- optimisation
- code generation

The Scanner

The first job a compiler must do is to read the input character by character, and group the characters into useful bits called lexemes. This task is called lexical analysis.

Here are some Ada lexemes

␣for␣i␣in␣NumberofItems␣loop

␣ins␣:=␣42␣;␣

There are some nontrivial problems; illustrated by these FORTRAN lexemes:

␣2.E3␣

␣2␣.EQ.␣I␣

The Scanner

The lexemes are represented as tokens – often integers – to facilitate further processing by the compiler.

Tokens indicate the lexical class of the lexeme – identifier, reserved word, delimeter, ... &c. – and usually include a reference to the lexeme itself.

A typical encoding uses one or two digits to represent the lexical class of a token, with other digits acting as indices into tables where the lexemes themselves are stored.

For example,

ins := 42;

might be encoded as

0233 0076 0062 0011

where the lowest order digit indicates the lexical class of the token, and the other digits index an entry for the token itself.

Other jobs done by the scanner

As well as doing its main job – recognising lexemes and encoding them as tokens – the scanner

- eliminates comments and "white space",
- formats and lists the source program, and
- processes compiler directives.

The Scanner in Context