Comp 204: Computer Systems and Their Implementation

Lecture 24: Compiler-Writing Tools

Today

- Compiler-writing tools
  - Regular expressions
  - Lex
  - Yacc
  - Code generator generators

Compiler-Writing Tools

- Various software tools exist which aid in the construction of compilers.
- Parser generators
  - e.g. *yacc*
- Code generator generators
- Scanner generators
  - e.g. *lex*
  - The input to *lex* consists of a definition of each token as a regular expression

Regular Expressions (REs)

- Used in many UNIX tools, e.g. awk, grep, sed, lex, vi
- REs specify patterns to be matched against input text
- An RE may be just a string:
  `cat` matches the string “cat”
- A full stop matches any single char:
  `c.t` matches cat, cut, cot, etc.

REs

- The beginning of a line is specified by ^
- End of line is specified as $
- An asterisk means “zero or more occurrences of the immediately preceding item”
  `xy*z` matches xz, xyz, xyyz, xyyyz, etc.
- A plus sign means “one or more…”
  `xy+z` matches xyz, xyyz, etc.
- A vertical bar means “or”; e.g.
  `x(a|b)y` matches xay or xby

Exercise

- What will be matched by the pattern `a.d` in the following line of characters?

```
add a dog and aardvark
```

- Using the same line of characters what will match `^a.d` ?
- What do we get if we search a file for all occurrences of the following patterns?

```
^hello$
```

```
^$
```

Exercise

- Using the same line of characters as before, what will be matched by the following?

```
an*d
an+d
```

- What will be matched by:

```
10*1
.*
```

Character Classes

- Square brackets denote a character class:
  `[abc]` matches character a, b, or c
- Can also abbreviate:
  `[1-6]` is equivalent to `[123456]`
- Asterisk and plus may be applied to character classes
  e.g. to define hex numbers in a Java or C program:
  `0x[0-9a-fA-F]+`
- Can negate a character class:
  `[^abc]` match any char *except* a,b,c
  - Note different use of ^

Exercise

- Which of the following will match

  `[Kk]a[itl]e`

  Kate kite kale kit ?

- What matches the following? `[ \t\n]+`

Lex

- Input to lex consists of pairs of REs and actions
  - Each RE defines a particular language token
  - Each action is a fragment of C code, to be executed if the token is encountered

- Lex transforms this input to a C function called *yylex ()*, which returns a token each time it is called

- The string that matches an RE is placed in an array called *yytext*

- Extra info about a token can be passed back to calling program via a global variable called *yylval*

Lex Example

```
[ \t\n]+          ;
while             return(WHILE_SYMB);
for               return(FOR_SYMB);
.
.
[0-9]+            { /* convert to int */
                    yylval = atoi(yytext);
                    return(NUMBER);
                  }

[a-zA-Z][a-zA-Z0-9]*
                  { /* find in sym tab */
                    yylval = lookup(yytext);
                    return(IDENT):
                  }
```

Yacc

- Stands for ‘Yet Another Compiler-Compiler’
- It is a parser generator
- Parser generators are programs that take as input the grammar defining a language and produce as output a parser for that language
- A Yacc parser matches sequences of input tokens to the rules of the given grammar

Yacc

- The specification file that Yacc takes as input consists of three sections
- Definitions: contains info about the tokens, data types and grammar rules required to build the parser
- Rules: contains the rules (in a form of BNF) of the grammar, along with actions in C code to be executed whenever a given rule is recognised
- Auxiliary routines: contains any auxiliary procedure and function declarations required to complete the parser

Yacc Example

- Example format of rules:

```
assign : IDENT BECOMES expr SEMI
                { /* action for assignment */
                }
while :   WHILE expr DO statement
                { /* action for while stat */
                }
```

- The parsing procedure produced by Yacc is called *yyparse()*
  - returns an int value : 0 if the parse is successful, 1 otherwise

Error Recovery in Yacc

- Errors need to be recognised and recovered from: Yacc provides error productions as the principal way to achieve this
- Error productions have on their right hand side an **`error`** pseudotoken
- These productions identify a context in which erroneous tokens can be deleted until tokens are encountered that enable the parse to be re-started
- When errors are encountered appropriate syntax error messages will be generated

Code Generator Generators

- CGGs remove the burden of deciding what code to generate for each construct
- Implementer produces a formal description of what each target machine instruction does
- CG automatically searches machine description to find the instructions(s) that produce desired computation
- Code almost as good as conventional compiler, but generation speed much slower

Question

- *Lex* is a software tool that can be used to aid compiler construction. It is an example of which of the following?

- A scanner generator
- A parser generator
- A code generator generator
- A semantic analyser
- A code debugger

Answer: a
Lex is responsible for identifying tokens using regular expressions. It is therefore a scanner generator