Compilers and Interpreters
Tutorial 1

Sorin Manolache, sorma@ida.liu.se

- *flex* and *bison* documentation available at http://www.ida.liu.se/~TDDB44/
- online version of this tutorial: http://www.ida.liu.se/~sorma/teaching/compinterp/tutorial1.pdf

Flex

- is a scanner generator
- the input is a file (specification file) containing mainly a set of rules for the token construction
- the output is a C source file containing the scanner code
- the entry point is the function

```
int yylex()
```

- `yylex` scans tokens from the `FILE *yyin (stdin)` until
  - a rule matches and the associated action executes a `return` statement
  - `EOF` is reached. `yylex` then returns `0`

Flex – Input File Specification

- 3 parts separated by %% on a new line

```
optional definitions
%%
optional rules
%%
optional additional C code
```

- rules section
  - format

    `<unindented pattern><whitespace><action>`
  - everything different from the above pattern is copied verbatim to the output file
  - `action` is C code

Flex – Patterns (1)

`x`	matches the character 'x'
`.`	matches any character except newline
`[xyz]`	
`[C-X5-9]`	matches any (one) of the characters between the brackets
`[^A-Z]`	matches any (one) character different from the characters between the brackets
`r*`	matches zero or more (unbounded) occurrences of r. r is itself a regexp
`r+`	matches one or more (unbounded) occurrences of r. r is itself a regexp
`r?`	matches zero or one r. r is itself a regexp
`r|s`	matches either r or s. r and s are themselves regexps
`rs`	matches r followed by s. r and s are themselves regexps

Flex – Patterns (2)

`(r)`	matches r. Used for priority overriding
`^r`	matches r occurring at the beginning of line
`r$`	matches r at the end of line
`<<EOF>>`	matches the end of file
`<s>r`	matches r in the context s. r is a regexp. s denotes a context (start condition)

- use “\” or \ for avoiding the special meaning of some symbols
- special characters can be used: `\n \r \t \a \b \f \v \0`
- operator priorities:
  - (), [] highest
  - *, +, ?
  - concatenation
  - | lowest

Flex – Matching (1)

- “greedy”, matches as much as possible
- if more than one rule can be applied, then the first appearing in the flex specification file is preferred
- if no rule matches, then the default rule applies: it copies the character to the output
- after matching
  - `yytext` (`extern char yytext[];`) contains the matched text
  - `yyleng` (`extern int yyleng;`) contains the matched text length

Flex – Matching (2)

- Special actions, macros and handy functions:
  - `ECHO`
  - `REJECT`, choose next best rule, either a “later” one matching the same text, or another rule matching a the largest prefix of the matched text
  - `yymore()`, do another match and append its result to the current match
  - `yyless(int n)`, push all but the first n characters back to the input stream (to be matched next time). `yytext` will contain only the first n of the matched characters.
  - `YY_USER_ACTION` denotes an action to be taken before the matched rule action

Flex – Definitions Section

- format:

  ```
  <unindented name><whitespace><definition>
  ```

- similar to macros, referred later in the rules section
- reference to a definition: `{name}`
- example:

  ```
  DIGIT    [0-9]
  %%
  {DIGIT}*  return INT;
  ```

- everything between `%{` and `%}` is copied verbatim to the generated C source file, *both in the definitions and in the rules section*

Flex – Contexts (Start Conditions)

- handy for matching some rules only if the scanner is in some state (context), for instance, when parsing inside comments or inside quotes
- the context (start condition) is defined in the definitions section
- the scanner enters a particular context after executing the action `BEGIN(start_condition_name)`
- if the start condition is *exclusive* then only those rules match that are specific to this context, i.e. those rules prefixed with `<start_condition_name>`
- if the start condition is *inclusive* then non-prefixed rules may match also
- prefixed rules cannot match if the scanner is not in the corresponding context (the start condition is false)

Flex – Contexts (2)

- the scanner exits a particular context either by entering another one or by entering the initial “start condition” (`BEGIN(INITIAL)`)
- syntax of start condition declaration:

```
%x exclusive_start_condition_name
%s inclusive_start_condition_name
```

- example:

```
%x html_tag
%%
[^<]*
“<”                          BEGIN(html_tag);
<html_tag>[^>]*              printf(“%s\n”, yytext);
<html_tag>”>”                BEGIN(INITIAL);
```

Flex – Running and Compiling

- `-i` case insensitive, or `%caseless`
- `%yylineno`
- `%yywrap` or `%noyywrap`
- `%main` or `%nomain`
- `-o` output file, or `%output=”name”`
- if `main` or `yywrap` have to be provided (no `%nomain` or no `%noyywrap`) and are not provided by the user, then one has to link with the `-lfl`