Ankit Agarwal <ankit.agarwal365@gmail.com>

Lab: Learning bison

1 message

Compiler Design <bit.compiler.2014@gmail.com> Fri, Jan 24, 2014 at 10:31 AM
Bcc: ankit.agarwal365@gmail.com

Part-1: Moving from flex-only to bison-flex

- Refer to the lex file emailed earlier. Add the following to it:
  - %option noyywrap
    - place it before the first %%
    - doing this removes the need for yywrap function as explained.
  - "return <token_name>" after printfs
  - "yylval = atoi(yytext)" in the action part of {number} RE.
    - yylval is a flex defined variable, which can be thought of
as the token-attribute.
- Create a <file>.y, with the following:
  - move main function from <file>.l to <file>.y
  - replace yylex by yyparse in main.
  - #include "lex.yy.c" before main.

Part-2: Basic Structure of <file>.y

```
%{
<C-declarations>
%}
<bison declarations>
%%
<bison grammar + actions>
%%
#include "lex.yy.c"
<user-code>
main(..)
```

- C-declarations
  - Global variables
  - externs being used from <file>.l
- Grammar rules
  - Grammar: 5 productions: 1-a, 1-b, 2-a, 2-b, 2-c

```
line: line expr NEW_LINE { ... action ... }
      |
      ;
expr: expr PLUS expr { ... action ... }
     | expr MULT expr { ... action ... }
     | NUMBER { ... action ... }
     ;
```

  - First rule's non-terminal on left always signifies the Start symbol.
  - Note the changes in syntax from the writing syntax:
    - ":" instead of "->"
    - <epsilon> is denoted by having nothing (note 1-b)

- action part is optional
- The actions can make use of
- $i, where "i" is the symbol number on the right side of production.
- if $i is a terminal, then the yylval of flex gets copied here.
- $$ is the non-terminal on the left side of production.
- The default action is $$ = $1

- Bison-declarations:
- %token <...>:
- The terminals (PLUS, MULT, NEW_LINE) of the grammar. Can think of as #defines.
- The ordering of these DO-NOT determine the precedence of tokens (PLUS, MULT)
- %left, %right, %nonassoc
- The terminals of grammar.
- If this is given, the %token is optional for that terminal. This itself acts as an #define.
- Gives the associativity of the token.
- Most importantly, the ordering determines the precedence: those appearing earlier have a lower precedence.

Part-3: Running

- Follow these FOUR steps on the DOS-prompt/terminal:
- flex <file>.l
- Generates lex.yy.c
- bison <file>.y
- Generates <file>.tab.c (or y.tab.c)
- May show some conflicts in case of ambiguous grammar.
- gcc <file>.tab.c -ly
- use -ly only if it gives a linker error without using it
- ./a.exe <input file>

- Let <input_file> have these 2 lines to start with:
10+20*30
10*20+30

Part-4: Sample problems:
---

(1) Do the program with the exact grammar, (without %left ...):
- Note the number of conflicts.
- Also note the output for both lines. We will start next class with this result.

(2) Swap the order of the productions 2-a, 2-b and see the result.

(3), (4): Repeat (1), (2) with following, and see the result.
%left MULT
%left PLUS

(5), (6): Repeat (1), (2) with above reversed.
%left PLUS
%left MULT
(1)-(6): Understand why the result is what it is. You can change the <input_file> how ever you like to experiment.

(7) Introduce some syntax errors in the <input_file> and see what happens. Have more than 2 lines, introduce the error somewhere in between, etc...

(8) Make PLUS non-associative, and see what happens in a line like 2+3+4.
%nonassoc PLUS

(9) Add one more production 2-d to the above grammar:
expr: (expr) {...action...}
Write the corresponding actions and test on various inputs.

Arun