Communication between Lex and Yacc

The scanner feeds the parser a stream of tokens. Those tokens are defined by Yacc in "y.tab.h", and are simple #defines -- not really capable of providing a lot of meta-data, such as the actual value of the token.

The scanner has that information. In the lex file, you can use the array "yytext" get at what the actual input was that caused a match. Is that data lost?

No. But the way into the parser is not with the yytext array. It's with the *yylval* variable. If all you want to pass is additional integer values, you don't have to do much beyond using an extern statement in the definition section of the lex file. Frequently you want more than that, and that's where the %union statement comes in.

In the definition section of the yacc file, you can use %union to declare the type of yylval to be a union. For example:

```
$ cat ex5.yacc
%{
/*
 * Example 5 - lex and yacc communication
 */
%}

%union {
   int integer_value;
   char *string_value;
   char op_value;
}

%token OP NAME NUMBER ASSIGN

%%

equation: NAME ASSIGN expression { printf("Got %s.\n",$1.string_value);
                                   return 0;
                                 };

expression: NUMBER
          | NUMBER expression
          ;

%%

int yywrap() { return 1; }

int main( int argc, char **argv ) {
   return yyparse();
}
```

The %union in the above yacc file declares that yylval will be a union of an integer, a string pointer, and a character.

This union is declared in the y.tab.h file:

```
$ yacc -d ex5.yacc
$ cat y.tab.h

typedef union
#ifdef __cplusplus
        YYSTYPE
#endif
 {
   int integer_value;
   char *string_value;
   char op_value;
} YYSTYPE;
extern YYSTYPE yylval;
# define OP 257
# define NAME 258
# define NUMBER 259
# define ASSIGN 260
```

You then import the y.tab.h file into your lex file as usual, and then you can access the yylval union in the actions for a pattern:

```
$ cat ex5.lex
%{
/*
 * Example 5 lexer
 */

#include
#include

#include "y.tab.h"
%}

%%

[\t ]+          /* ignore */ ;

\+ |
\- |
\* |
\/              {
                   yylval.op_value = yytext[0];
                   return OP;
                };

[a-zA-Z]+       {
                   yylval.string_value = strdup( yytext );
                   return NAME;
                };

[0-9]+          {
                   yylval.integer_value = atoi( yytext );
                   return NUMBER;
                };

\=              {return ASSIGN;} ;
```

```
%%
```

Of course, a more interesting grammar would do more, but ours does what it is supposed to do:

```
$ lex ex5.lex
$ gcc -o ex5 y.tab.c lex.yy.c -ly
$ ./ex5
foo = 99 + 1 - 33 * 44 / 2
Got foo.
$ ^D
exit
```

So it runs. But what do we *do* with the yylval variable? It doesn't show up in the yacc file at all.

Well, that where $-directives come in. Recall from the yacc file the equation production?

```
equation: NAME ASSIGN expression { printf("Got %s.\n",$1.string_value);
                                   return 0;
                                 };
```

The `$1.string_value` indicates that we want the yylval value (with the string_value portion of the union) for the first token in the right-hand-side of this production.

The Hard Way

If you really wanted to, you could pass that data around the long way, by writing your own code to manipulate global variables and constructing your productions so that you could readily access the appropriate data. That's more work than you really need to do, considering how easy it is to use yylval for that purpose.

Alternatively, you could avoid the %union and just treat yylval as a char * pointer. All it would take would be a few lines at the top of the lex file:

```
%{
  /*
   * Example lex file definition section.
   */

#include

#define YYSTYPE char *
extern YYSTYPE yylval;

#include "y.tab.h"
%}
%%
```

But again, why bother? This is bad coding style, to force an int to do a char *'s job -- and the way to do it right isn't all that hard.

$Id: YaccAndLex.html,v 1.3 2003/04/29 02:23:35 stremler Exp stremler $