[Previous | Next | Contents | Glossary | Home | Search]

AIX Version 4.3 General Programming Concepts: Writing and Debugging Programs

Example Program for the lex and yacc Programs

This section describes example programs for the **lex** and **yacc** commands. Together, these example programs create a simple, desk-calculator program that performs addition, subtraction, multiplication, and division operations. This calculator program also allows you to assign values to variables (each designated by a single, lowercase letter) and then use the variables in calculations. The files that contain the example **lex** and **yacc** programs are:

File	Content
calc.lex	Specifies the **lex** command specification file that defines the lexical analysis rules.
calc.yacc	Specifies the **yacc** command grammar file that defines the parsing rules, and calls the **yylex** subroutine created by the **lex** command to provide input.

The following descriptions assume that the **calc.lex** and **calc.yacc** example programs are found in your current directory.

Compiling the Example Program

Perform the following steps, in order, to create the desk calculator example program:

1. Process the **yacc** grammar file using the **-d** optional flag (which tells the **yacc** command to create a file that defines the tokens used in addition to the C language source code):

   ```
   yacc -d calc.yacc
   ```

2. Use the **li** command to verify that the following files were created:

   y.tab.c The C language source file that the **yacc** command created for the parser.

   y.tab.h A header file containing define statements for the tokens used by the parser.

3. Process the **lex** specification file:

   ```
   lex calc.lex
   ```

4. Use the **li** command to verify that the following file was created:

   lex.yy.c The C language source file that the **lex** command created for the lexical analyzer.

5. Compile and link the two C language source files:

   ```
   cc y.tab.c lex.yy.c
   ```

6. Use the **li** command to verify that the following files were created:

   y.tab.o The object file for the **y.tab.c** source file

   lex.yy.o The object file for the **lex.yy.c** source file

   a.out The executable program file

   To then run the program directly from the **a.out** file, enter:

   ```
   $ a.out
   ```

   Or, to move the program to a file with a more descriptive name, as in the following example, and run it, enter:

```
$ mv a.out calculate
$ calculate
```

In either case, after you start the program, the cursor moves to the line below the `$` (command prompt). Then enter numbers and operators in calculator fashion. When you press the Enter key, the program displays the result of the operation. After you assign a value to a variable:

```
m=4 <enter>
_
```

the cursor moves to the next line. When you use the variable in subsequent calculations, it will have the assigned value:

```
m+5 <enter>
9
_
```

Parser Source Code

The following example shows the contents of the **calc.yacc** file. This file has entries in all three sections of a **yacc** grammar file: declarations, rules, and programs.

```
%{
#include <stdio.h>

int regs[26];
int base;

%}

%start list

%token DIGIT LETTER

%left '|'
%left '&'
%left '+' '-'
%left '*' '/' '%'
%left UMINUS  /*supplies precedence for unary minus */

%%                   /* beginning of rules section */

list:                       /*empty */
         |
        list stat '\n'
         |
        list error '\n'
         {
           yyerrok;
         }
         ;

stat:    expr
         {
           printf("%d\n",$1);
         }
         |
         LETTER '=' expr
         {
           regs[$1] = $3;
         }
         ;

  expr:    '(' expr ')'
           {
             $$ = $2;
           }
           |
           expr '*' expr
           {
             $$ = $1 * $3;
           }
           |
           expr '/' expr
           {
             $$ = $1 / $3;
           }
           |
           expr '%' expr
           {
             $$ = $1 % $3;
           }
           |
           expr '+' expr
           {
             $$ = $1 + $3;
           }
            |

           expr '-' expr
           {
             $$ = $1 - $3;
           }
           |
           expr '&' expr
           {
             $$ = $1 & $3;
           }
           |
           expr '|' expr
           {
             $$ = $1 | $3;
           }
           |

          '-' expr %prec UMINUS
           {
             $$ = -$2;
           }
           |
           LETTER
           {
             $$ = regs[$1];
           }

           |
           number
           ;

  number:  DIGIT
           {
             $$ = $1;
             base = ($1==0) ? 8 : 10;
           }       |
           number DIGIT
           {
```

```
           $$ = base * $1 + $2;
         }
         ;

%%
main()
{
 return(yyparse());
}

yyerror(s)
char *s;
{
  fprintf(stderr, "%s\n",s);
}

yywrap()
{
  return(1);
}
```

Declarations Section

This section contains entries that:

- Include standard I/O header file.
- Define global variables.
- Define the `list` rule as the place to start processing.
- Define the tokens used by the parser.
- Define the operators and their precedence.

Rules Section

The rules section defines the rules that parse the input stream.

Programs Section

The programs section contains the following subroutines. Because these subroutines are included in this file, you do not need to use the **yacc** library when processing this file.

main The required main program that calls the **yyparse** subroutine to start the program.

yyerror(s) This error-handling subroutine only prints a syntax error message.

yywrap The wrap-up subroutine that returns a value of 1 when the end of input occurs.

Lexical Analyzer Source Code

Following are the contents of the **calc.lex** file. This file contains include statements for standard input and output, as well as for the **y.tab.h** file. The **yacc** program generates that file from the **yacc** grammar file information if you use the **-d** flag with the **yacc** command. The **y.tab.h** file contains definitions for the tokens that the parser program uses. In addition, **calc.lex** contains the rules to generate these tokens from the input stream.

```
%{

#include <stdio.h>
#include "y.tab.h"
int c;
extern int yylval;
%}
%%
" "       ;
[a-z]     {
            c = yytext[0];
            yylval = c - 'a';
            return(LETTER);
          }
[0-9]     {
            c = yytext[0];
            yylval = c - '0';
            return(DIGIT);
          }
[^a-z0-9\b]    {
                 c = yytext[0];
                 return(c);
               }
```

Related Information

Tools and Utilities Overview for Programmers

Creating an Input Language with the lex and yacc Commands

Using the lex Program with the yacc Program

ed command, **lex** command, **sed** command, **yacc** command

printf subroutine

[Previous | Next | Contents | Glossary | Home | Search]