Örebro University

Compilers and Interpreters

Assignment 3

Authors:

Linus Baumgärtner
lbaumgaertner@culba.de

Marko Reichhart
marko.reichhart@web.de

October 13, 2022

Part A

- Study the Makefile, Which commands used to build example are written explicitly in the file?
  - flex example.l (-> example.tab.c, example.tab.h)
  - bison -d example.y (-> lex.yy.c)
  - gcc -Wall -std=c99 -g -o example example.tab.o lex.yy.o (-> example)
  -
- Are there some commands that make will deduce by itself?
  - The object files example.tab.o and lex.yy.o are generated automatically. Commands:
    * gcc -c lex.yy.o
    * gcc -c example.tab.c
- Which types of expressions does the calculator understand?
  - Plus, minus and multiplication
- One common operator is missing. Which one?
  - Division
- Add the missing operator!
  - Add to *expr* in *example.y*:
    `"/"  return SLASH;`
  - Add to *example.l*:
    `| expr SLASH expr  $$ = $1 / $3;`

Part B and Part C

The bison file *parser.y* can be seen in listing 1. The grammar is kept simple by using the precendence functionality of bison by defining left assortativity for the operators in the correct order at the definitions part of the file. Then the non-terminal list consists either of an assignment or an expression. An assignment is a ID followed by an equals sign followed by an expression, an expression can either be a NUM, an ID or a link of two expressions seperated by an operator.

Listing 1: Bison file parser.y

```
%{
    #include <stdio.h>
    #include <math.h>
    #include "global.h"

    extern int yyerror(char const *msg);
    extern int token_value;
    extern int yylex();
%}

%token DONE ID NUM
%left '&' '|' '<' '>' '?' ':'
%left '+' '-'
%left '*'
%left '/' '%'
%left '^'

%%

start: list DONE
        ;
```

```
list: assignment ';' list
        | expr ';' list
        | /* empty */
        ;
assignment : ID '=' expr    { $$ = $3; symtable[$1].value = $3; printf("%s\n=\n%d\n",
                                symtable[$1].lexeme, $3); }
        ;
expr : NUM                  { printf("%d\n", $1); }
    | ID                    { $$ = symtable[$1].value; printf("%d\n", symtable[$1].value); }
    | expr '+' expr         { $$ = $1 + $3; printf("+\n"); }
    | expr '-' expr         { $$ = $1 - $3; printf("-\n"); }
    | expr '*' expr         { $$ = $1 * $3; printf("*\n"); }
    | expr '/' expr         { $$ = $1 / $3; printf("/\n"); }
    | expr '%' expr         { $$ = $1 % $3; printf("%%\n"); }
    | expr '^' expr         { $$ = pow($1, $3); printf("^\n"); }
    | '(' expr ')'    { $$ = $2; }
    | expr '&' expr      { $$ = $1 & $3; printf("&\n"); }
    | expr '|' expr             { $$ = $1 | $3; printf("|\n"); }
    | expr '<' expr             { $$ = $1 < $3; printf("<\n"); }
    | expr '>' expr             { $$ = $1 > $3; printf(">\n"); }
    | expr '?' expr ':' expr    { $$ = $1 ? $3 : $5; printf("?\n"); }
    ;

%%

int yyerror(char const *msg) {
    printf("Error: %s\n", msg);
    return 0;
}

int yylex() {
    int c = lexan();
    yylval = token_value;
    return c;
}

void parse() {
  yyparse();
}
```

Part D

To make the code working in C++ the endings of all the source files where changed to *.ypp*. Then the Makefile was adjusted to those changes and the compilation executed by make.