Project #2
CS 3510 – Spring 2014
Seth Yost, Mitch Birti

I. Requirements:
Develop a parser which can parse a program written in the C- language described in Appendix A of the text.

II. Design:
We built a recursive descent parser as described in class and in the book, using the scanner developed in project 1A/1B.

III. Implementation:
We made an object for each language structure (variable declarations, expressions, loops, etc.) and wrote parse methods to parse those structures using the first and follow sets of the grammar.

IV. Testing:
We wrote a Tester class that outputs the tree generated by the parser when it is given a test file. We tested using a program given in the book and with 3 test programs from the course website.

V. Summary/Conclusion:
Our project is working properly. We didn't encounter any major problems with our Parser implementation, which was extremely surprising. We did have a bit of trouble with our grammar not being up to specifications, and we were missing some tokens in our first and follow sets. This was resolved when we tried to parse structures that depended on those first/follow set tokens.

Grammar

```
program → declaration-list
declaration-list → declaration {declaration}
declaration → int ID declaration' | void ID fun-declaration'
declaration' → var-declaration' | fun-declaration'
var-declaration → int ID [ [ NUM ] ] ;
var-declaration' → [ [ NUM ] ] ;
fun-declaration' → ( params ) compound-stmt
params → param-list | void
param-list → param { , param }
param → int ID [ [ ] ]
compound-stmt → { local-declarations statement-list }
local-declarations → {var-declaration}
statement-list → {statement}
statement → expression-stmt | compound-stmt | selection-stmt
            | iteration-stmt | return-stmt
expression-stmt → [expression] ;
selection-stmt → if ( expression ) statement [ else statement ]
iteration-stmt → while ( expression ) statement
return-stmt → return [expression] ;

expression → ( expression ) simple-expression' | NUM simple-expression'
            | ID expression'
expression' → = expression | ( args ) simple-expression'
            | [ expression ] expression'' | simple-expression'
expression'' → = expression | simple-expression'
var → [ [ expression ] ]
simple-expression' → additive-expression' [ relop additive-expression ]
relop → <= | < | > | >= | == | !=
additive-expression → term { addop term }
additive-expression' → term' { addop term }
addop → + | -
term → factor { mulop factor }
term' → { mulop factor }
mulop → * | /
factor → ( expression ) | ID varcall | NUM

varcall → var | call
call → ( args )
args → [ arg-list ]
arg-list → expression { , expression }
```

First and Follow Sets

	First Set	Follow Set
program	INT, VOID	$
declaration-list	INT, VOID	$
declaration	INT, VOID	INT, VOID, $
declaration'	(, [, ;	INT, VOID, $
var-declaration	INT	INT, (, NUM, ID, ;, {, }, IF, WHILE, RETURN, EPSILON
var-declaration'	[, ;	INT, VOID, $
fun-declaration'	(	INT, VOID, $
params	INT, VOID	)
param-list	INT	)
param	INT	COMMA,)
compound-stmt	{	INT, VOID, $, (, NUM, ID, ;, {, IF, WHILE, RETURN, ELSE, }
local-declarations	INT, EPSILON	(, NUM, ID, ;, {, }, IF, WHILE, RETURN, EPSILON
statement-list	(, NUM, ID, ;, {, IF, WHILE, RETURN, EPSILON	}
statement	(, NUM, ID, ;, {, IF, WHILE, RETURN	(, NUM, ID, ;, {, IF, WHILE, RETURN, ELSE, }
expression-stmt	(, NUM, ID, ;	(, NUM, ID, ;, {, IF, WHILE, RETURN, ELSE, }
selection-stmt	IF	(, NUM, ID, ;, {, IF, WHILE, RETURN, ELSE, }
iteration-stmt	WHILE	(, NUM, ID, ;, {, IF, WHILE, RETURN, ELSE, }
return-stmt	RETURN	(, NUM, ID, ;, {, IF, WHILE, RETURN, ELSE, }
expression	(, NUM, ID	;, COMMA,),],)
expression'	=, (, [, *, /, +, /, EPSILON	;, COMMA,),],)
expression''	=, *, /, +, -, EPSILON	;, COMMA,),],)
var	[, EPSILON	+, -, *, /, ;, COMMA,),],)
simple-expression'	*, /, +, -, <=, <, >, >=, ==, !=, EPSILON	;, COMMA,),],)
relop	<=, <, >, >=, ==, !=	(, NUM, ID

additive-expression	(, NUM, ID	;, COMMA,),],)
additive-expression'	*, /, +, -, EPSILON	*, /, ;, COMMA,),],), <=, <, >, >=, ==, !=
addop	+, -	(, NUM, ID
term	(, NUM, ID	+, -, *, /, ;, COMMA,),],), <=, <, >, >=, ==, !=
term'	*, /, EPSILON	+, -, *, /, ;, COMMA,),],)
mulop	*, /	(, NUM, ID
factor	(, NUM, ID	+, -, *, /, ;, COMMA,),],), <=, <, >, >=, ==, !=
varcall	EPSILON, [, (	+, -, *, /, ;, COMMA,),],)
call	(	+, -, *, /, ;, COMMA,),],)
args	(, NUM, ID, EPSILON	)
arg-list	(, NUM, ID	)

```
package parser;

import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;

import com.sun.org.apache.xpath.internal.operations.Variable;

import scanner.Scanner;
import scanner.Token;
import scanner.Token.TokenType;
import parser.expression.*;
import parser.statement.*;

public class CMinusParser implements Parser
{
   private Scanner scanner;
   // populated after a parse
   private Program parsedProgram = null;

   /*
    * First/Follow Sets
    * The first sub-array is the first set, the following one is the follow set.
    * (Makes sense, right?)
    */
   private TokenType[][] PROGRAM = {
         { TokenType.INT, TokenType.VOID },
         { TokenType.EOF }
   };
   private TokenType[][] DECLARATION_LIST = {
         { TokenType.INT, TokenType.VOID },
         { TokenType.EOF }
   };
   private TokenType[][] DECLARATION = {
         { TokenType.INT, TokenType.VOID },
         { TokenType.INT, TokenType.VOID, TokenType.EOF }
   };
   private TokenType[][] DECLARATION_PRIME = {
         { TokenType.OPEN_PAREN, TokenType.OPEN_BRACKET, TokenType.END_STATEMENT
 },
         { TokenType.INT, TokenType.VOID, TokenType.EOF }
   };
   private TokenType[][] VAR_DECLARATION = {
         { TokenType.INT },
         { TokenType.INT, TokenType.OPEN_PAREN, TokenType.NUM, TokenType.ID, Tok
enType.END_STATEMENT, TokenType.OPEN_CBRACE, TokenType.CLOSE_CBRACE, TokenType.I
F, TokenType.WHILE, TokenType.RETURN, TokenType.EPSILON }
   };
   private TokenType[][] VAR_DECLARATION_PRIME = {
         { TokenType.OPEN_BRACKET, TokenType.END_STATEMENT },
         { TokenType.INT, TokenType.VOID, TokenType.EOF }
   };
   private TokenType[][] FUN_DECLARATION_PRIME = {
         { TokenType.OPEN_PAREN },
         { TokenType.INT, TokenType.VOID, TokenType.EOF }
   };
   private TokenType[][] PARAMS = {
```

```
            { TokenType.INT, TokenType.VOID },
            { TokenType.CLOSE_PAREN }
    };
    private TokenType[][] PARAM_LIST = {
            { TokenType.INT },
            { TokenType.CLOSE_PAREN }
    };
    private TokenType[][] PARAM = {
            { TokenType.INT },
            { TokenType.COMMA, TokenType.CLOSE_PAREN }
    };
    private TokenType[][] COMPOUND_STMT = {
            { TokenType.OPEN_CBRACE },
            { TokenType.INT, TokenType.VOID, TokenType.EOF, TokenType.OPEN_PAREN, T
okenType.NUM, TokenType.ID, TokenType.END_STATEMENT, TokenType.OPEN_CBRACE, Toke
nType.IF, TokenType.WHILE, TokenType.RETURN, TokenType.ELSE, TokenType.CLOSE_CBR
ACE }
    };
    private TokenType[][] LOCAL_DECLARATIONS = {
            { TokenType.INT, TokenType.EPSILON },
            { TokenType.OPEN_PAREN, TokenType.NUM, TokenType.ID, TokenType.END_STAT
EMENT, TokenType.OPEN_CBRACE, TokenType.CLOSE_CBRACE, TokenType.IF, TokenType.WH
ILE, TokenType.RETURN, TokenType.EPSILON }
    };
    private TokenType[][] STATEMENT_LIST = {
            { TokenType.OPEN_PAREN, TokenType.NUM, TokenType.ID, TokenType.END_STAT
EMENT, TokenType.OPEN_CBRACE, TokenType.IF, TokenType.WHILE, TokenType.RETURN, T
okenType.EPSILON },
            { TokenType.CLOSE_CBRACE }
    };
    private TokenType[][] STATEMENT = {
            { TokenType.OPEN_PAREN, TokenType.NUM, TokenType.ID, TokenType.END_STAT
EMENT, TokenType.OPEN_CBRACE, TokenType.IF, TokenType.WHILE, TokenType.RETURN },
            { TokenType.OPEN_PAREN, TokenType.NUM, TokenType.ID, TokenType.END_STAT
EMENT, TokenType.OPEN_CBRACE, TokenType.IF, TokenType.WHILE, TokenType.RETURN, T
okenType.ELSE, TokenType.CLOSE_CBRACE }
    };
    private TokenType[][] EXPRESSION_STMT = {
            { TokenType.OPEN_PAREN, TokenType.NUM, TokenType.ID, TokenType.END_STAT
EMENT },
            { TokenType.OPEN_PAREN, TokenType.NUM, TokenType.ID, TokenType.END_STAT
EMENT, TokenType.OPEN_CBRACE, TokenType.IF, TokenType.WHILE, TokenType.RETURN, T
okenType.ELSE, TokenType.CLOSE_CBRACE }
    };
    private TokenType[][] SELECTION_STMT = {
            { TokenType.IF },
            { TokenType.OPEN_PAREN, TokenType.NUM, TokenType.ID, TokenType.END_STAT
EMENT, TokenType.OPEN_CBRACE, TokenType.IF, TokenType.WHILE, TokenType.RETURN, T
okenType.ELSE, TokenType.CLOSE_CBRACE }
    };
    private TokenType[][] ITERATION_STMT = {
            { TokenType.WHILE },
            { TokenType.OPEN_PAREN, TokenType.NUM, TokenType.ID, TokenType.END_STAT
EMENT, TokenType.OPEN_CBRACE, TokenType.IF, TokenType.WHILE, TokenType.RETURN, T
okenType.ELSE, TokenType.CLOSE_CBRACE }
    };
    private TokenType[][] RETURN_STMT = {
            { TokenType.RETURN },
            { TokenType.OPEN_PAREN, TokenType.NUM, TokenType.ID, TokenType.END_STAT
```

```
EMENT, TokenType.OPEN_CBRACE, TokenType.IF, TokenType.WHILE, TokenType.RETURN, T
okenType.ELSE, TokenType.CLOSE_CBRACE }
   };
   private TokenType[][] EXPRESSION = {
         { TokenType.OPEN_PAREN, TokenType.NUM, TokenType.ID },
         { TokenType.END_STATEMENT, TokenType.COMMA, TokenType.CLOSE_PAREN, Toke
nType.CLOSE_BRACKET, TokenType.CLOSE_PAREN }
   };
   private TokenType[][] EXPRESSION_PRIME = {
         { TokenType. ASSIGNMENT, TokenType.OPEN_PAREN, TokenType.OPEN_BRACKET,
TokenType.MULTIPLY, TokenType.DIVIDE, TokenType.PLUS, TokenType.MINUS, TokenType
.EPSILON },
         { TokenType.END_STATEMENT, TokenType.COMMA, TokenType.CLOSE_PAREN, Toke
nType.CLOSE_BRACKET, TokenType.CLOSE_PAREN }
   };
   private TokenType[][] EXPRESSION_PRIME_PRIME = {
         { TokenType. ASSIGNMENT, TokenType.MULTIPLY, TokenType.DIVIDE, TokenTyp
e.PLUS, TokenType.MINUS, TokenType.EPSILON },
         { TokenType.END_STATEMENT, TokenType.COMMA, TokenType.CLOSE_PAREN, Toke
nType.CLOSE_BRACKET, TokenType.CLOSE_PAREN }
   };
   private TokenType[][] VAR = {
         { TokenType.OPEN_BRACKET, TokenType.EPSILON },
         { TokenType.PLUS, TokenType.MINUS, TokenType.MULTIPLY, TokenType.DIVIDE
, TokenType.END_STATEMENT, TokenType.COMMA, TokenType.CLOSE_PAREN, TokenType.CLO
SE_BRACKET, TokenType.CLOSE_PAREN }
   };
   private TokenType[][] SIMPLE_EXPRESSION_PRIME = {
         { TokenType.MULTIPLY, TokenType.DIVIDE, TokenType.PLUS, TokenType.MINUS
, TokenType.LESS_EQUAL_THAN, TokenType.LESS_THAN, TokenType.GREATER_THAN, TokenT
ype.GREATER_EQUAL_THAN, TokenType.EQUALS, TokenType.NOT_EQUALS, TokenType.EPSILO
N },
         { TokenType.END_STATEMENT, TokenType.COMMA, TokenType.CLOSE_PAREN, Toke
nType.CLOSE_BRACKET, TokenType.CLOSE_PAREN }
   };
   private TokenType[][] RELOP = {
         { TokenType.LESS_EQUAL_THAN, TokenType.LESS_THAN, TokenType.GREATER_THA
N, TokenType.GREATER_EQUAL_THAN, TokenType.EQUALS, TokenType.NOT_EQUALS },
         { TokenType.OPEN_PAREN, TokenType.NUM, TokenType.ID }
   };
   private TokenType[][] ADDITIVE_EXPRESSION = {
         { TokenType.OPEN_PAREN, TokenType.NUM, TokenType.ID },
         { TokenType.END_STATEMENT, TokenType.COMMA, TokenType.CLOSE_PAREN, Toke
nType.CLOSE_BRACKET, TokenType.CLOSE_PAREN }
   };
   private TokenType[][] ADDITIVE_EXPRESSION_PRIME = {
         { TokenType.MULTIPLY, TokenType.DIVIDE, TokenType.PLUS, TokenType.MINUS
, TokenType.LESS_EQUAL_THAN, TokenType.LESS_THAN, TokenType.GREATER_THAN, TokenT
ype.GREATER_EQUAL_THAN, TokenType.EQUALS, TokenType.NOT_EQUALS, TokenType.EPSILO
N },
         { TokenType.LESS_EQUAL_THAN, TokenType.LESS_THAN, TokenType.GREATER_THA
N, TokenType.GREATER_EQUAL_THAN, TokenType.EQUALS, TokenType.NOT_EQUALS, TokenTy
pe.MULTIPLY, TokenType.DIVIDE, TokenType.END_STATEMENT, TokenType.COMMA, TokenTy
pe.CLOSE_PAREN, TokenType.CLOSE_BRACKET, TokenType.CLOSE_PAREN }
   };
   private TokenType[][] ADDOP = {
         { TokenType.PLUS, TokenType.MINUS },
         { TokenType.OPEN_PAREN, TokenType.NUM, TokenType.ID }
   };
```

```
   private TokenType[][] TERM = {
         { TokenType.OPEN_PAREN, TokenType.NUM, TokenType.ID },
         { TokenType.LESS_EQUAL_THAN, TokenType.LESS_THAN, TokenType.GREATER_THA
N, TokenType.GREATER_EQUAL_THAN, TokenType.EQUALS, TokenType.NOT_EQUALS, TokenTy
pe.PLUS, TokenType.MINUS, TokenType.MULTIPLY, TokenType.DIVIDE, TokenType.END_ST
ATEMENT, TokenType.COMMA, TokenType.CLOSE_PAREN, TokenType.CLOSE_BRACKET, TokenT
ype.CLOSE_PAREN }
   };
   private TokenType[][] TERM_PRIME = {
         { TokenType.MULTIPLY, TokenType.DIVIDE, TokenType.EPSILON },
         { TokenType.PLUS, TokenType.MINUS, TokenType.MULTIPLY, TokenType.DIVIDE
, TokenType.END_STATEMENT, TokenType.COMMA, TokenType.CLOSE_PAREN, TokenType.CLO
SE_BRACKET, TokenType.CLOSE_PAREN }
   };
   private TokenType[][] MULOP = {
         { TokenType.MULTIPLY, TokenType.DIVIDE },
         { TokenType.OPEN_PAREN, TokenType.NUM, TokenType.ID }
   };
   private TokenType[][] FACTOR = {
         { TokenType.OPEN_PAREN, TokenType.NUM, TokenType.ID },
         { TokenType.LESS_EQUAL_THAN, TokenType.LESS_THAN, TokenType.GREATER_THA
N, TokenType.GREATER_EQUAL_THAN, TokenType.EQUALS, TokenType.NOT_EQUALS, TokenTy
pe.PLUS, TokenType.MINUS, TokenType.MULTIPLY, TokenType.DIVIDE, TokenType.END_ST
ATEMENT, TokenType.COMMA, TokenType.CLOSE_PAREN, TokenType.CLOSE_BRACKET, TokenT
ype.CLOSE_PAREN }
   };
   private TokenType[][] VARCALL = {
         { TokenType.EPSILON, TokenType.OPEN_BRACKET, TokenType.OPEN_PAREN },
         { TokenType.PLUS, TokenType.MINUS, TokenType.MULTIPLY, TokenType.DIVIDE
, TokenType.END_STATEMENT, TokenType.COMMA, TokenType.CLOSE_PAREN, TokenType.CLO
SE_BRACKET, TokenType.CLOSE_PAREN }
   };
   private TokenType[][] CALL = {
         { TokenType.OPEN_PAREN },
         { TokenType.PLUS, TokenType.MINUS, TokenType.MULTIPLY, TokenType.DIVIDE
, TokenType.END_STATEMENT, TokenType.COMMA, TokenType.CLOSE_PAREN, TokenType.CLO
SE_BRACKET, TokenType.CLOSE_PAREN }
   };
   private TokenType[][] ARGS = {
         { TokenType.OPEN_PAREN, TokenType.NUM, TokenType.ID, TokenType.EPSILON
},
         { TokenType.CLOSE_PAREN }
   };
   private TokenType[][] ARG_LIST = {
         { TokenType.OPEN_PAREN, TokenType.NUM, TokenType.ID },
         { TokenType.CLOSE_PAREN }
   };

   public CMinusParser(Scanner s)
   {
      scanner = s;
   }

   /**
    * Satisfies the Parse interface.
    */
   @Override
   public Program parse()
   {
```

```
      if (parsedProgram == null)
      {
         parsedProgram = parseProgram();
      }
      return parsedProgram;
   }

   /**
    * Prints out the parsed program.
    * A call to parse() must precede this call.
    */
   @Override
   public void printTree(String outFile)
   {
      try
      {
         BufferedWriter bw = new BufferedWriter(new FileWriter(outFile));

         if (parsedProgram != null)
         {
            parsedProgram.print(0, bw);
         }

         bw.flush();
         bw.close();
      }
      catch (IOException e)
      {
         e.printStackTrace();
      }
   }

   /**
    * Matches and consumes the given token, if it is present.
    * @returns True if matched, false if the next token isn't the one we are loo
king for.
    */
   private boolean matchToken(TokenType t)
   {
      if (scanner.viewNextToken().getType() == t)
      {
         scanner.getNextToken();
         return true;
      }
      return false;
   }

   /**
    * Searches First/Follow sets for the given token.
    * @param needle
    * @param haystack
    * @return
    */
   private boolean contains(TokenType needle, TokenType[] haystack)
   {
      for (TokenType straw : haystack)
      {
         if (needle == straw)
            return true;
```

```
      }
      return false;
   }

   /**
    * Views the next token type (look ahead).
    * @return
    */
   private TokenType nextTokenType()
   {
      return scanner.viewNextToken().getType();
   }

   /**
    * Matches (and munches) a Token or throws
    * an exception with the given message.
    * @param tt
    * @param msg
    * @return
    */
   private Token matchOrDie(TokenType tt, String msg)
   {
      if (scanner.viewNextToken().getType() != tt)
      {
         throw new RuntimeException(msg + nextTokenType().name());
      }
      return scanner.getNextToken();
   }

   /*
    * Parse methods, BEGIN!
    */

   /**
    * Parses a Program
    * @return
    */
   public Program parseProgram()
   {
      ArrayList<Declaration> declList = new ArrayList<Declaration>();

      while (contains(nextTokenType(), DECLARATION[0]))
      {
         declList.add(parseDeclaration());
      }

      if (!matchToken(TokenType.EOF))
      {
         throw new RuntimeException("parseProgram(): illegal token: " + nextTokenType().na
me());
      }

      return new Program(declList);
   }

   /**
    * Parses a Declaration.
    * @return
    */
```

```
   public Declaration parseDeclaration()
   {
      Declaration toReturn;
      // strip off the first two tokens
      Token typeSpecifier = scanner.getNextToken();
      Token identifier = scanner.getNextToken();

      if (typeSpecifier.getType() == TokenType.VOID || (typeSpecifier.getType()
== TokenType.INT && nextTokenType() == TokenType.OPEN_PAREN))
      {
         // declaration -> void ID fun-declaration'
         // OR  decl    -> int ID fun-declaration'
         // this isn't exactly according to the grammar, but it's better coding
practice

         matchOrDie(TokenType.OPEN_PAREN, "parseDeclaration(): parsing function, open paren expec
ted, got ");
         Params params = parseParams();
         matchOrDie(TokenType.CLOSE_PAREN, "parseDeclaration(): parsing function, close paren exp
ected, got ");

         CompoundStatement body = parseCompoundStatement();

         toReturn = new FunctionDeclaration(typeSpecifier.getType(), (String) id
entifier.getData(), params, body);
      }
      else if (typeSpecifier.getType() == TokenType.INT)
      {
         // declaration -> int ID declaration'
         // AND decl'   -> var-declaration'

         if (matchToken(TokenType.OPEN_BRACKET))
         {
            Token number = matchOrDie(TokenType.NUM, "parseDeclaration(): parsing array decla
ration, expected NUM, got ");

            matchOrDie(TokenType.CLOSE_BRACKET, "parseDeclaration(): parsing array declaration,
expected ']', got " + nextTokenType().name());

            toReturn = new VariableDeclaration((String) identifier.getData(), (I
nteger) number.getData());
         }
         else if (nextTokenType() == TokenType.END_STATEMENT)
         {
            toReturn = new VariableDeclaration((String) identifier.getData());
         }
         else
         {
            throw new RuntimeException("parseDeclaration(): parsing variable, '[' or ';' expected, got
" + nextTokenType().name());
         }

         matchOrDie(TokenType.END_STATEMENT, "parseDeclaration(): parsing variable declaration, ';
' expected, got ");

      }
      else
      {
         throw new RuntimeException("parseDeclaration(): type specifier expected, received " + typ
```

```
eSpecifier.getType().name());
      }

      return toReturn;
   }

   /**
    * Parses function parameters.
    * @return
    */
   private Params parseParams()
   {
      List<VariableDeclaration> params = new ArrayList<VariableDeclaration>();

      if (matchToken(TokenType.VOID))
      {
         //do nothing, since there's no parameters
      }
      else if (nextTokenType() == TokenType.INT)
      {
         // grab the first param
         matchOrDie(TokenType.INT, "parseParams(): INT expected, but got ");
         Token id = matchOrDie(TokenType.ID, "parseParams(): identifier expected, got ");
         if (matchToken(TokenType.OPEN_BRACKET))
         {
            params.add(new VariableDeclaration((String) id.getData(), 0));
            matchOrDie(TokenType.CLOSE_BRACKET, "parseParams(): expected ']', but got ");
         }
         else
         {
            params.add(new VariableDeclaration((String) id.getData()));
         }

         // check for other params
         while (nextTokenType() == TokenType.COMMA)
         {
            scanner.getNextToken();

            matchOrDie(TokenType.INT, "parseParams(): INT expected, but got ");
            id = matchOrDie(TokenType.ID, "parseParams(): identifier expected, got ");
            if (matchToken(TokenType.OPEN_BRACKET))
            {
               params.add(new VariableDeclaration((String) id.getData(), 0));
               matchOrDie(TokenType.CLOSE_BRACKET, "parseParams(): expected ']', but got ");
            }
            else
            {
               params.add(new VariableDeclaration((String) id.getData()));
            }
         }
      }
      else
      {
         throw new RuntimeException("parseParams(): expected 'void' or 'int', got " + nextToken
Type().name());
      }

      return new Params(params);
   }
```

```
   /**
    * Parses a CompoundStatement.
    * @return
    */
   private CompoundStatement parseCompoundStatement()
   {
      matchOrDie(TokenType.OPEN_CBRACE, "parseCompoundStatement(): expected '{', got ");
      List<Declaration> decls = new ArrayList<Declaration>();
      List<Statement> stmts = new ArrayList<Statement>();

      while (nextTokenType() == TokenType.INT)
      {
         scanner.getNextToken();
         Token id = matchOrDie(TokenType.ID, "parseCompoundStatement(): parsing variable decl
aration, expected identifier and got ");

         if (matchToken(TokenType.OPEN_BRACKET))
         {
            Token num = matchOrDie(TokenType.NUM, "parseCompoundStatement(): expected a nu
mber, but got ");
            matchOrDie(TokenType.CLOSE_BRACKET, "parseCompoundStatement(): expected ']', but
got ");
            matchOrDie(TokenType.END_STATEMENT, "parseCompoundStatement(): expected ';', but
got ");
            decls.add(new VariableDeclaration((String) id.getData(), (Integer) n
um.getData()));
         }
         else if (matchToken(TokenType.END_STATEMENT))
         {
            decls.add(new VariableDeclaration((String) id.getData()));
         }
         else
         {
            throw new RuntimeException("parseCompoundStatement(): expected '[' or ';', got ");
         }
      }

      while (contains(nextTokenType(), STATEMENT_LIST[0]) && nextTokenType() !=
TokenType.CLOSE_CBRACE)
      {
         stmts.add(parseStatement());
      }

      matchOrDie(TokenType.CLOSE_CBRACE, "parseCompoundStatement(): expected '}', got ");
      return new CompoundStatement(decls, stmts);
   }

   /**
    * Parses Statement
    * @return
    */
   private Statement parseStatement()
   {
      Statement toReturn = null;
      if(nextTokenType() == TokenType.IF)
      {
         //if ( expression ) statement [ else statement ]
         toReturn = parseSelectionStatement();
```

```
        }
        else if(nextTokenType() == TokenType.WHILE)
        {
           //while ( expression ) statement
           toReturn = parseIterationStatement();
        }
        else if(nextTokenType() == TokenType.RETURN)
        {
           //return [expression] ;
           toReturn = parseReturnStatement();
        }
        else if(nextTokenType() == TokenType.OPEN_CBRACE)
        {
           //{ local-declarations statement-list }
           toReturn = parseCompoundStatement();
        }
        else if(contains(nextTokenType(), EXPRESSION[0]))
        {
           //[expression] ;
           toReturn = parseExpressionStatement();
        }
        else
        {
           throw new RuntimeException("parseStatement(): Invalid token for statement, got ");
        }
        return toReturn;
    }

    /**
     * Parses an Expression Statement
     * @return
     */
    private Statement parseExpressionStatement()
    {
       //[expression] ;
       Expression body = null;
       if(contains(nextTokenType(), EXPRESSION[0]))
       {
          body = parseExpression();
       }
       matchOrDie(TokenType.END_STATEMENT, "parseReturnStatement(): Did not recieve ';', got");
       Statement toReturn = new ExpressionStatement(body);
       return toReturn;
    }

    /**
     * Parses a SelectionStatement
     * @return
     */
    private Statement parseSelectionStatement()
    {
       //if ( expression ) statement [ else statement ]
       Statement else_part = null;
       matchOrDie(TokenType.IF,  "parseSelectionStatement(): Did not recieve 'IF', got");
       matchOrDie(TokenType.OPEN_PAREN, "parseSelectionStatement(): Did not recieve '(', got");
       Expression compare = parseExpression();
       matchOrDie(TokenType.CLOSE_PAREN, "parseSelectionStatement(): Did not recieve ')' after '(', go
t");
       Statement body = parseStatement();
```

```
      if(nextTokenType() == TokenType.ELSE)
      {
         matchOrDie(TokenType.ELSE, "parseSelectionStatement(): Did not recieve 'else' after 'if', got"
);
         else_part = parseStatement();
      }
      return new SelectionStatement(compare, body, else_part);
   }

   /**
    * Parses Iteration Statement
    * @return
    */
   private Statement parseIterationStatement()
   {
      //while ( expression ) statement
      matchOrDie(TokenType.WHILE, "parseIterationStatement(): Did not receive WHILE token, got ");
      matchOrDie(TokenType.OPEN_PAREN, "parseIterationStatement(): Did not recieve '(', got ");
      Expression compare = parseExpression();
      matchOrDie(TokenType.CLOSE_PAREN, "parseIterationStatement(): Did not recieve ')' after '(', got
");
      Statement body = parseStatement();

      return new IterationStatement(compare, body);
   }

   /**
    * Parses Return Statement
    * @return
    */
   private Statement parseReturnStatement()
   {
      //return [expression] ;
      Expression body = null;
      matchOrDie(TokenType.RETURN, "parseReturnStatement(): Did not recieve 'RETURN', got");
      if(contains(nextTokenType(), EXPRESSION[0]))
      {
         body = parseExpression();
      }
      matchOrDie(TokenType.END_STATEMENT, "parseReturnStatement(): Did not recieve ';', got");
      return new ReturnStatement(body);
   }

   /**
    * Parses Additive-Expression and Additive-Expression'
    * @param lhs
    * @return
    */
   private Expression parseAdditiveExpression(Expression lhs)
   {
      Expression term = null;
      if(lhs == null)
      {
         term = parseTerm(null);
      }
      else
      {
         term = parseTerm(lhs);
      }
```

```
      while (contains(nextTokenType(), ADDOP[0]))
      {
         if (nextTokenType() == TokenType.PLUS||nextTokenType() == TokenType.MIN
US)
         {
            term = new BinaryExpression(term, scanner.getNextToken().getType(),
parseTerm(null));
         }
         else
         {
            throw new RuntimeException("parseTerm(): '*' or '/' expected, but got ");
         }
      }

      return term;
   }

   /**
    * Parses term and  term'.
    * @param term
    * @return
    */
   private Expression parseTerm(Expression term)
   {
      if(term == null)
      {
         term = parseFactor();
      }

      while (contains(nextTokenType(), MULOP[0]))
      {
         if (nextTokenType() == TokenType.MULTIPLY||nextTokenType() == TokenType
.DIVIDE)
         {
            term = new BinaryExpression(term, scanner.getNextToken().getType(),
parseFactor());
         }
         else
         {
            throw new RuntimeException("parseTerm(): '*' or '/' expected, but got ");
         }
      }

      return term;
   }

   /**
    * Parses a Factor.
    * @return
    */
   private Expression parseFactor()
   {
      Expression toReturn;

      if (matchToken(TokenType.OPEN_PAREN))
      {
         // factor -> ( expression )
```

```
            toReturn = parseExpression();

            matchOrDie(TokenType.CLOSE_PAREN, "parseFactor(): No ')' found, got ");
         }
         else if (nextTokenType() == TokenType.NUM)
         {
            // factor -> NUM
            toReturn = new NumberExpression((Integer)(scanner.getNextToken().getDat
a()));
         }
         else if (nextTokenType() == TokenType.ID)
         {
            // factor -> ID varcall
            Token id = scanner.getNextToken();

            if (matchToken(TokenType.OPEN_PAREN))
            {
               // varcall -> call -> ( args )
               List<Expression> args = parseArgs();

               toReturn = new CallExpression((String) id.getData(), args);

               matchOrDie(TokenType.CLOSE_PAREN, "parseFactor(): No ')' found after args in function
call, got ");
            }
            else if (matchToken(TokenType.OPEN_BRACKET))
            {
               // varcall -> var -> [ expression ]
               Expression xpr = parseExpression();
               toReturn = new VariableExpression((String) id.getData(), xpr);

               matchOrDie(TokenType.CLOSE_BRACKET, "parseFactor(): No ']' found after '[', got");
            }
            else if (contains(nextTokenType(), VARCALL[1]))
            {
               // next token is in $varcall
               // varcall -> var -> EPSILON
               toReturn = new VariableExpression((String) id.getData());
            }
            else
            {
               throw new RuntimeException("parseFactor(): Illegal token after ID!");
            }
         }
         else
         {
            throw new RuntimeException("parseFactor(): Illegal token for factor, got " + nextTokenT
ype().name());
         }

         return toReturn;
      }

      /**
       * Parses Expression, Expression', and Expression''
       * @return
       */
      private Expression parseExpression()
      {
```

```
      Expression toReturn = null;

      if(matchToken(TokenType.OPEN_PAREN))
      {
         //Expression -> ( expression ) simple-expression'
         Expression compare = parseExpression();
         matchOrDie(TokenType.CLOSE_PAREN, "parseExpression(): No ')' found after '(', got ");
         toReturn = parseSimpleExpression(compare);
      }
      else if(nextTokenType() == TokenType.ID)
      {
         Token ID = scanner.getNextToken();

         //Expression -> ID expression'
         if(matchToken(TokenType.ASSIGNMENT))
         {
            //expression' -> = expression
            VariableExpression var = new VariableExpression((String)ID.getData()
);
            toReturn = new AssignExpression(var, parseExpression());
         }
         else if(matchToken(TokenType.OPEN_PAREN))
         {
            //expression' -> ( args ) simple-expression'
            List<Expression> args = parseArgs();
            Expression func = new CallExpression((String)ID.getData(), args);
            matchOrDie(TokenType.CLOSE_PAREN, "parseExpression(): No ')' found after '(', got ")
;
            toReturn = parseSimpleExpression(func);
         }
         else if(matchToken(TokenType.OPEN_BRACKET))
         {
            //expression' -> [ expression ] expression''

               Expression internalExpr = parseExpression();

               matchOrDie(TokenType.CLOSE_BRACKET, "parseExpression(): No ']' found after '[',
got ");
               Expression varExpression = new VariableExpression((String)ID.getD
ata(), internalExpr);

               if(matchToken(TokenType.ASSIGNMENT))
               {
                  //expression'' -> = expression
                  toReturn = new AssignExpression((VariableExpression) varExpres
sion, parseExpression());
               }
               else if(contains(nextTokenType(), SIMPLE_EXPRESSION_PRIME[0]))
               {
                  //expression'' -> simple-expression'
                  toReturn = parseSimpleExpression(varExpression);
               }
               else if(contains(nextTokenType(), SIMPLE_EXPRESSION_PRIME[1]))
               {
                  // expression'' -> simple-expression' -> epsilon
                  toReturn = varExpression;
               }
               else
               {
```

```
                    throw new RuntimeException("parseExpression(): Illegal token following ]!");
                }
            }
            else if(contains(nextTokenType(), SIMPLE_EXPRESSION_PRIME[0]) || contai
ns(nextTokenType(), SIMPLE_EXPRESSION_PRIME[1]))
            {
               //expression' -> simple-expression'
               Expression temp = new VariableExpression((String) ID.getData());
               toReturn = parseSimpleExpression(temp);
            }
            else
            {
               throw new RuntimeException("parseExpression(): Illegal token following " + ID.getT
ype().name() + ":" + ID.getData() + ", got " + nextTokenType());
            }
         }
         else if(nextTokenType() == TokenType.NUM)
         {
            //Expression -> NUM simple-expression'
            Token num = scanner.getNextToken();
            Expression Num = new NumberExpression((Integer)num.getData());
            toReturn = parseSimpleExpression(Num);
         }
         else
         {
            throw new RuntimeException("parseExpression(): Illegal token for Expression, got " + next
TokenType());
         }
         return toReturn;
      }

      /**
       * Parses Simple-Expression'
       * @param lhs
       * @return
       */
      private Expression parseSimpleExpression(Expression lhs) {
         Expression toReturn = null;
         Expression left = parseAdditiveExpression(lhs);

         if(contains(nextTokenType(), RELOP[0]))
         {
            //match Relop
            TokenType opp = scanner.getNextToken().getType();

            Expression right = parseAdditiveExpression(null);
            toReturn = new BinaryExpression(left, opp, right);
         }
         else
         {
            toReturn = left;
         }

         return toReturn;
      }

      /**
       * Parses Args and Arg-list
       * @return
```

```
     */
    private List<Expression> parseArgs()
    {
       List<Expression> args = new ArrayList<Expression>();

       while (nextTokenType() != TokenType.CLOSE_PAREN)
       {
          args.add(parseExpression());

          if (nextTokenType() != TokenType.CLOSE_PAREN)
          {
             matchOrDie(TokenType.COMMA, "parseArgs(): We didn't match the comma?? Instead we fo
und a ");
          }
       }

       return args;
    }
}
```

```
package parser;

import java.io.BufferedWriter;
import java.io.IOException;

/**
 * Represents a declaration (variable or function).
 */
public abstract class Declaration
{
   public abstract void print(int indent, BufferedWriter out) throws IOException
;
}
```

```
package parser;

import java.io.BufferedWriter;
import java.io.IOException;

import parser.statement.CompoundStatement;
import scanner.Token.TokenType;

/**
 * Represents a function declaration.
 */
public class FunctionDeclaration extends Declaration
{
   // the function's return type (INT or VOID)
   private TokenType returnType;
   // expected parameters
   private Params parameters;
   // function name
   private String name;
   // function body
   private CompoundStatement body;

   public FunctionDeclaration(TokenType returnType, String functionName, Params
params, CompoundStatement body)
   {
      this.returnType = returnType;
      parameters = params;
      name = functionName;
      this.body = body;
   }

   public String getName()
   {
      return name;
   }

   public Params getParams()
   {
      return parameters;
   }

   public CompoundStatement getBody()
   {
      return body;
   }

   @Override
   public void print(int indent, BufferedWriter out) throws IOException
   {
      String prefix = "";
      for (int i = 0; i < indent; i++)
         prefix += "\t";

      out.write(prefix + "<FunctionDeclaration>\n");

      out.write(prefix + "\t<Name>" + name + "</Name>\n");
      out.write(prefix + "\t<ReturnType>" + returnType.name() + "</ReturnType>\n");

      parameters.print(indent + 1, out);
```

```
        body.print(indent + 1, out);
        out.write(prefix + "</FunctionDeclaration>\n");
    }
}
```

```
package parser;

import java.io.BufferedWriter;
import java.io.IOException;
import java.util.List;

/**
 * Represents parameters for a function definition.
 */
public class Params
{
   // the parameters
   List<VariableDeclaration> paramList;

   public Params(List<VariableDeclaration> list)
   {
      paramList = list;
   }

   public List<VariableDeclaration> getParameters()
   {
      return paramList;
   }

   public void print(int indent, BufferedWriter out) throws IOException
   {
      String prefix = "";
      for (int i = 0; i < indent; i++)
         prefix += "\t";

      out.write(prefix + "<Params>\n");

      for (VariableDeclaration varDec : paramList)
      {
         varDec.print(indent + 1, out);
      }

      out.write(prefix + "</Params>\n");
   }
}
```

```
package parser;

/**
 * Parser interface.
 * Meets the specifications for the project.
 */
public interface Parser
{
   public void printTree(String outFile);
   public Program parse();
}
```

```
package parser;

import java.io.BufferedWriter;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;

/**
 * Represents a program in a single file.
 */
public class Program
{
   // the top-level declarations in this program (var and func)
   private List<Declaration> declarations;

   public Program(List<Declaration> decls)
   {
      declarations = decls;
   }

   public List<Declaration> getDeclarations()
   {
      return declarations;
   }

   public void print(int indent, BufferedWriter out) throws IOException
   {
      String prefix = "";
      for (int i = 0; i < indent; i++)
         prefix += "\t";

      out.write(prefix + "<Program>\n");

      for (Declaration decl : declarations)
         decl.print(indent + 1, out);

      out.write(prefix + "</Program>\n");
   }
}
```

```
package parser;

import java.io.BufferedWriter;
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;

import scanner.CMinusScanner;
import scanner.Scanner;
import scanner.Token;
import scanner.Token.TokenType;

/**
* @author Mitch Birti
* @author Seth Yost
* @version 1.0
* File: Tester.java
* Created: Feb 2014
* Â©Copyright the authors. All rights reserved.
*
* Description: Tests CMinusParser.java
*/
public class Tester
{

   public static void main(String[] args)
   {
      try
      {
         // set up the scanner and the input/output file
         String baseName = "TestFile";
         Scanner s = new CMinusScanner(new BufferedReader(new FileReader("tests/"
+ baseName + ".cm")));

         // make the parser
         Parser parser = new CMinusParser(s);

         // parse the program
         parser.parse();
         // write the tree to file
         parser.printTree("tests/" + baseName + ".xml");
      }
      catch (Exception e)
      {
         e.printStackTrace();
      }
   }


}
```

```
package parser;

import java.io.BufferedWriter;
import java.io.IOException;

/**
 * Represents a variable declaration.
 */
public class VariableDeclaration extends Declaration
{
   // the name of the variable
   private String id;

   // -1 indicates this isn't an array
   // 0 indicates unknown size (for function params)
   private int arraySize;

   /**
    * Makes a variable of type int.
    * @param id
    */
   public VariableDeclaration(String id)
   {
      this.id = id;
      arraySize = -1;
   }

   /**
    * Makes an array of type int.
    * @param id
    * @param size
    */
   public VariableDeclaration(String id, int size)
   {
      this.id = id;
      arraySize = size;
   }

   public String getId()
   {
      return id;
   }

   /**
    * Gets the size of this variable array.
    * -1 indicates it is not an array.
    * @return
    */
   public int arraySize()
   {
      return arraySize;
   }

   @Override
   public void print(int indent, BufferedWriter out) throws IOException
   {
      String prefix = "";
      for (int i = 0; i < indent; i++)
         prefix += "\t";
```

```
      out.write(prefix + "<VariableDeclaration>\n");

      out.write(prefix + "\t<Name>" + id + "</Name>\n");
      out.write(prefix + "\t<Type>" + "INT" + "</Type>\n");

      if (arraySize != -1)
         out.write(prefix + "\t<Size>" + arraySize + "</Size>\n");

      out.write(prefix + "</VariableDeclaration>\n");
   }
}
```

```
package parser.expression;

import java.io.BufferedWriter;
import java.io.IOException;

/**
 * Represents assigning a value to a variable.
 */
public class AssignExpression extends Expression
{
   // the variable being written to
   private VariableExpression variable;
   // the value to assign to the variable
   private Expression rightSide;

   public AssignExpression(VariableExpression var, Expression rhs)
   {
      variable = var;
      rightSide = rhs;
   }

   public VariableExpression getVariableExpr()
   {
      return variable;
   }

   public Expression getRightSide()
   {
      return rightSide;
   }

   public void print(int indent, BufferedWriter out) throws IOException
   {
      String prefix = "";
      for (int i = 0; i < indent; i++)
         prefix += "\t";

      out.write(prefix + "<AssignExpression>\n");

      out.write(prefix + "\t<Variable>\n");
      variable.print(indent + 2, out);
      out.write(prefix + "\t</Variable>\n");

      out.write(prefix + "\t<Value>\n");
      rightSide.print(indent + 2, out);
      out.write(prefix + "\t</Value>\n");

      out.write(prefix + "</AssignExpression>\n");
   }
}
```

```
package parser.expression;

import java.io.BufferedWriter;
import java.io.IOException;

import scanner.Token.TokenType;

/**
 * Represents a binary expression.
 */
public class BinaryExpression extends Expression
{
   // the left side of the expression
   private Expression leftSide;
   // the right side of the expression
   private Expression rightSide;
   // the operator: anything in relop, addop, mulop
   private TokenType operand;

   public BinaryExpression(Expression lhs, TokenType op, Expression rhs)
   {
      leftSide = lhs;
      operand = op;
      rightSide = rhs;
   }

   public Expression getLeftSide()
   {
      return leftSide;
   }

   public Expression getRightSide()
   {
      return rightSide;
   }

   public TokenType getOperand()
   {
      return operand;
   }

   @Override
   public void print(int indent, BufferedWriter out) throws IOException
   {
      String prefix = "";
      for (int i = 0; i < indent; i++)
         prefix += "\t";

      out.write(prefix + "<BinaryExpression>\n");

      out.write(prefix + "\t<Operand>" + operand.name() + "</Operand>\n");

      out.write(prefix + "\t<LeftSide>\n");
      leftSide.print(indent + 2, out);
      out.write(prefix + "\t</LeftSide>\n");

      out.write(prefix + "\t<RightSide>\n");
      rightSide.print(indent + 2, out);
      out.write(prefix + "\t</RightSide>\n");
```

```
        out.write(prefix + "</BinaryExpression>\n");
    }
}
```

```
package parser.expression;

import java.io.BufferedWriter;
import java.io.IOException;
import java.util.List;

/**
 * Represents a function call.
 */
public class CallExpression extends Expression
{
   // the name of the function
   String functionName;
   // the arguments being passed to the function
   List<Expression> arguments;

   public CallExpression(String funcName, List<Expression> args)
   {
      functionName = funcName;
      arguments = args;
   }

   public String getFunctionName()
   {
      return functionName;
   }

   public List<Expression> getArgs()
   {
      return arguments;
   }

   @Override
   public void print(int indent, BufferedWriter out) throws IOException
   {
      String prefix = "";
      for (int i = 0; i < indent; i++)
         prefix += "\t";

      out.write(prefix + "<CallExpression>\n");

      out.write(prefix + "\t<FunctionName>" + functionName + "</FunctionName>\n");

      out.write(prefix + "\t<Arguments>\n");

      for (Expression arg : arguments)
         arg.print(indent + 2, out);

      out.write(prefix + "\t</Arguments>\n");
      out.write(prefix + "</CallExpression>\n");
   }
}
```

```
package parser.expression;

import java.io.BufferedWriter;
import java.io.IOException;

/**
 * Represents any kind of expression.
 */
public abstract class Expression
{
   public abstract void print(int indent, BufferedWriter out) throws IOException
;
}
```

```
package parser.expression;

import java.io.BufferedWriter;
import java.io.IOException;

/**
 * Represents a numerical value.
 * e.g. '5'
 */
public class NumberExpression extends Expression
{
   // the numerical value
   private int value;

   public NumberExpression(int val)
   {
      value = val;
   }

   public int getValue()
   {
      return value;
   }

   @Override
   public void print(int index, BufferedWriter out) throws IOException
   {
      String prefix = "";
      for (int i = 0; i < index; i++)
         prefix += "\t";

      out.write(prefix + "<NumberExpression>" + value + "</NumberExpression>\n");
   }
}
```

```
package parser.expression;

import java.io.BufferedWriter;
import java.io.IOException;

/**
 * Represents an expression consisting of a single variable.
 */
public class VariableExpression extends Expression
{
   private String identifier;
   private Expression index;

   public VariableExpression(String id)
   {
      identifier = id;
      index = null;
   }

   public VariableExpression(String id, Expression index)
   {
      identifier = id;
      this.index = index;
   }

   public String getIdentifier()
   {
      return identifier;
   }

   /**
    * Gets the index expression for this array.
    * If null, this isn't an array variable.
    * @return
    */
   public Expression getIndex()
   {
      return index;
   }

   @Override
   public void print(int indent, BufferedWriter out) throws IOException
   {
      String prefix = "";
      for (int i = 0; i < indent; i++)
         prefix += "\t";

      out.write(prefix + "<VariableExpression>\n");

      out.write(prefix + "\t<Identifier>" + identifier + "</Identifier>\n");
      if(index != null)
      {
         out.write(prefix + "\t<Index>\n");
         index.print(indent + 2, out);
         out.write(prefix + "\t</Index>\n");
      }

      out.write(prefix + "</VariableExpression>\n");
   }
```

```
}
```

```
package parser.statement;

import parser.*;

import java.io.BufferedWriter;
import java.io.IOException;
import java.util.List;

/**
 * Represents a group of declarations and statements
 * that are enclosed in curly braces.
 */
public class CompoundStatement extends Statement
{
   // the local declarations
   private List<Declaration> locals;
   // the statements to be executed in this block
   private List<Statement> body;

   public CompoundStatement(List<Declaration> locals, List<Statement> body)
   {
      this.locals = locals;
      this.body = body;
   }

   public List<Declaration> getLocals()
   {
      return locals;
   }

   public List<Statement> getBody()
   {
      return body;
   }

   // to keep Statement happy in the meantime
   @Override
   public void print(int indent, BufferedWriter out) throws IOException
   {
      String prefix = "";
      for (int i = 0; i < indent; i++)
         prefix += "\t";

      out.write(prefix + "<CompoundStatement>\n");
      out.write(prefix + "\t<Declarations>\n");
      for (Declaration decl : locals)
      {
         decl.print(indent+2, out);
      }
      out.write(prefix + "\t</Declarations>\n");
      out.write(prefix + "\t<Statments>\n");
      for (Statement temp : body)
      {
         temp.print(indent+2, out);
      }
      out.write(prefix + "\t</Statments>\n");
      out.write(prefix + "</CompoundStatement>\n");
   }
}
```

```
package parser.statement;

import java.io.BufferedWriter;
import java.io.IOException;

import parser.expression.*;

/**
 * Represents an expression statement.
 * e.g. 'func();'
 * or 'A + B;' (though I don't know why you'd want to do that)
 */
public class ExpressionStatement extends Statement
{
   // the expression
   private Expression data;

   public ExpressionStatement(Expression data)
   {
      this.data = data;
   }

   public Expression getData()
   {
      return data;
   }

   public void print(int indent, BufferedWriter out) throws IOException
   {
      String prefix = "";
      for(int i = 0; i < indent; i++)
         prefix += "\t";

      out.write(prefix + "<ExpressionStatement>\n");
      data.print(indent+1, out);
      out.write(prefix + "</ExpressionStatement>\n");
   }

}
```

```
package parser.statement;

import java.io.BufferedWriter;
import java.io.IOException;

import parser.expression.*;

/**
 * Represents a while loop.
 */
public class IterationStatement extends Statement
{
   // the deciding expression
   private Expression compare;
   // the code to execute in the loop
   private Statement body;

   public IterationStatement(Expression compare, Statement body)
   {
      this.compare = compare;
      this.body = body;
   }

   public Expression getCompare()
   {
      return compare;
   }

   public Statement getBody()
   {
      return body;
   }

   public void print(int indent, BufferedWriter out) throws IOException
   {
      String prefix = "";
      for(int i = 0; i < indent; i++)
         prefix += "\t";

      out.write(prefix + "<IterationStatement>\n");
      out.write(prefix + "\t<Expression>\n");
      compare.print(indent+2, out);
      out.write(prefix + "\t</Expression>\n");
      out.write(prefix + "\t<Do>\n");
      body.print(indent+2, out);
      out.write(prefix + "\t</Do>\n");
      out.write(prefix + "</IterationStatement>\n");
   }
}
```

```
package parser.statement;

import java.io.BufferedWriter;
import java.io.IOException;

import parser.expression.*;

/**
 * Represents a return from a function.
 */
public class ReturnStatement extends Statement
{
   // optional expression to evaluate and return
   private Expression body;

   public ReturnStatement(Expression body)
   {
      this.body = body;
   }

   public Expression getBody()
   {
      return body;
   }

   public void print(int indent, BufferedWriter out) throws IOException
   {
      String prefix = "";
      for(int i = 0; i < indent; i++)
         prefix += "\t";

      out.write(prefix + "<ReturnStatement>");
      if (body != null)
      {
         out.write("\n" + prefix + "\t<Expression>\n");
         body.print(indent+2, out);
         out.write(prefix + "\t</Expression>\n" + prefix);
      }
      out.write("</ReturnStatement>\n");
   }
}
```

```
package parser.statement;

import java.io.BufferedWriter;
import java.io.IOException;

import parser.statement.*;
import parser.expression.*;

/**
 * Represents an if statement.
 */
public class SelectionStatement extends Statement
{
   // the decision expession
   private Expression compare;
   // the body to be executed if the expression is 1
   private Statement body;
   // the optional else statement
   private Statement else_part;

   public SelectionStatement(Expression compare, Statement body, Statement else_
part)
   {
      this.compare = compare;
      this.body = body;
      this.else_part = else_part;
   }

   public Expression getCompare() {
      return compare;
   }

   public Statement getBody() {
      return body;
   }

   public Statement getElse_part() {
      return else_part;
   }

   public void print(int indent, BufferedWriter out) throws IOException
   {
      String prefix = "";
      for (int i = 0; i < indent; i++)
         prefix += "\t";

      out.write(prefix + "<SelectionStatement>\n");

      out.write(prefix + "\t<Expression>\n");
      compare.print(indent+2, out);
      out.write(prefix + "\t</Expression>\n");

      out.write(prefix + "\t<Then>\n");
      body.print(indent+2, out);
      out.write(prefix + "\t</Then>\n");

      if(else_part != null)
      {
         out.write(prefix + "\t<Else>\n");
```

```
            else_part.print(indent+2, out);
            out.write(prefix + "\t</Else>\n");
        }

        out.write(prefix + "</SelectionStatement>\n");
    }

}
```

```
package parser.statement;

import java.io.BufferedWriter;
import java.io.IOException;

/**
 * Abstract class for statements (epxressions, if, while, return, etc.).
 */
public abstract class Statement
{
   public abstract void print(int indent, BufferedWriter out) throws IOException
;
}
```

```
void test(void){
  int a;
  int b;
  a = 0;
  b = 1;
  a = b = 0;
  if(a > b){
    b = b - 1;
  }
  else{
    b = 2;
    while( b == 2){
      b = 2;
      if(b == 2){
        b = b + 1;
      }
      else{
        b = 1;
        while(b == 1){
          b = b + 2;
        }
      }
    }
  }
  if(a == b){
    while (a == b){
      while(a == b){
        a = b  - 1;
      }
    }
  }
  b = 3;
  return;
}
```

```
<Program>
   <FunctionDeclaration>
      <Name>test</Name>
      <ReturnType>VOID</ReturnType>
      <Params>
      </Params>
      <CompoundStatement>
         <Declarations>
            <VariableDeclaration>
               <Name>a</Name>
               <Type>INT</Type>
            </VariableDeclaration>
            <VariableDeclaration>
               <Name>b</Name>
               <Type>INT</Type>
            </VariableDeclaration>
         </Declarations>
         <Statments>
            <ExpressionStatement>
               <AssignExpression>
                  <Variable>
                     <VariableExpression>
                        <Identifier>a</Identifier>
                     </VariableExpression>
                  </Variable>
                  <Value>
                     <NumberExpression>0</NumberExpression>
                  </Value>
               </AssignExpression>
            </ExpressionStatement>
            <ExpressionStatement>
               <AssignExpression>
                  <Variable>
                     <VariableExpression>
                        <Identifier>b</Identifier>
                     </VariableExpression>
                  </Variable>
                  <Value>
                     <NumberExpression>1</NumberExpression>
                  </Value>
               </AssignExpression>
            </ExpressionStatement>
            <ExpressionStatement>
               <AssignExpression>
                  <Variable>
                     <VariableExpression>
                        <Identifier>a</Identifier>
                     </VariableExpression>
                  </Variable>
                  <Value>
                     <AssignExpression>
                        <Variable>
                           <VariableExpression>
                              <Identifier>b</Identifier>
                           </VariableExpression>
                        </Variable>
                        <Value>
                           <NumberExpression>0</NumberExpression>
                        </Value>
```

```
                  </AssignExpression>
               </Value>
            </AssignExpression>
         </ExpressionStatement>
         <SelectionStatement>
            <Expression>
               <BinaryExpression>
                  <Operand>GREATER_THAN</Operand>
                  <LeftSide>
                     <VariableExpression>
                        <Identifier>a</Identifier>
                     </VariableExpression>
                  </LeftSide>
                  <RightSide>
                     <VariableExpression>
                        <Identifier>b</Identifier>
                     </VariableExpression>
                  </RightSide>
               </BinaryExpression>
            </Expression>
            <Then>
               <CompoundStatement>
                  <Declarations>
                  </Declarations>
                  <Statments>
                     <ExpressionStatement>
                        <AssignExpression>
                           <Variable>
                              <VariableExpression>
                                 <Identifier>b</Identifier>
                              </VariableExpression>
                           </Variable>
                           <Value>
                              <BinaryExpression>
                                 <Operand>MINUS</Operand>
                                 <LeftSide>
                                    <VariableExpression>
                                       <Identifier>b</Identifier>
                                    </VariableExpression>
                                 </LeftSide>
                                 <RightSide>
                                    <NumberExpression>1</NumberExpression>
                                 </RightSide>
                              </BinaryExpression>
                           </Value>
                        </AssignExpression>
                     </ExpressionStatement>
                  </Statments>
               </CompoundStatement>
            </Then>
            <Else>
               <CompoundStatement>
                  <Declarations>
                  </Declarations>
                  <Statments>
                     <ExpressionStatement>
                        <AssignExpression>
                           <Variable>
                              <VariableExpression>
```

```
                                    <Identifier>b</Identifier>
                                 </VariableExpression>
                              </Variable>
                              <Value>
                                 <NumberExpression>2</NumberExpression>
                              </Value>
                           </AssignExpression>
                        </ExpressionStatement>
                        <IterationStatement>
                           <Expression>
                              <BinaryExpression>
                                 <Operand>EQUALS</Operand>
                                 <LeftSide>
                                    <VariableExpression>
                                       <Identifier>b</Identifier>
                                    </VariableExpression>
                                 </LeftSide>
                                 <RightSide>
                                    <NumberExpression>2</NumberExpression>
                                 </RightSide>
                              </BinaryExpression>
                           </Expression>
                           <Do>
                              <CompoundStatement>
                                 <Declarations>
                                 </Declarations>
                                 <Statments>
                                    <ExpressionStatement>
                                       <AssignExpression>
                                          <Variable>
                                             <VariableExpression>
                                                <Identifier>b</Identifier>
                                             </VariableExpression>
                                          </Variable>
                                          <Value>
                                             <NumberExpression>2</NumberExpressi
on>

                                          </Value>
                                       </AssignExpression>
                                    </ExpressionStatement>
                                    <SelectionStatement>
                                       <Expression>
                                          <BinaryExpression>
                                             <Operand>EQUALS</Operand>
                                             <LeftSide>
                                                <VariableExpression>
                                                   <Identifier>b</Identifier>
                                                </VariableExpression>
                                             </LeftSide>
                                             <RightSide>
                                                <NumberExpression>2</NumberExpre
ssion>

                                             </RightSide>
                                          </BinaryExpression>
                                       </Expression>
                                       <Then>
                                          <CompoundStatement>
                                             <Declarations>
                                             </Declarations>
```

```
                                                <Statments>
                                                   <ExpressionStatement>
                                                      <AssignExpression>
                                                         <Variable>
                                                            <VariableExpression>
                                                               <Identifier>b</Ident
ifier>
                                                            </VariableExpression>
                                                         </Variable>
                                                         <Value>
                                                            <BinaryExpression>
                                                               <Operand>PLUS</Opera
nd>
                                                               <LeftSide>
                                                                  <VariableExpressi
on>
                                                                     <Identifier>b<
/Identifier>
                                                                  </VariableExpress
ion>
                                                               </LeftSide>
                                                               <RightSide>
                                                                  <NumberExpression
>1</NumberExpression>
                                                               </RightSide>
                                                            </BinaryExpression>
                                                         </Value>
                                                      </AssignExpression>
                                                   </ExpressionStatement>
                                                </Statments>
                                             </CompoundStatement>
                                          </Then>
                                          <Else>
                                             <CompoundStatement>
                                                <Declarations>
                                                </Declarations>
                                                <Statments>
                                                   <ExpressionStatement>
                                                      <AssignExpression>
                                                         <Variable>
                                                            <VariableExpression>
                                                               <Identifier>b</Ident
ifier>
                                                            </VariableExpression>
                                                         </Variable>
                                                         <Value>
                                                            <NumberExpression>1</Nu
mberExpression>
                                                         </Value>
                                                      </AssignExpression>
                                                   </ExpressionStatement>
                                                   <IterationStatement>
                                                      <Expression>
                                                         <BinaryExpression>
                                                            <Operand>EQUALS</Operan
d>
                                                            <LeftSide>
                                                               <VariableExpression>
                                                                  <Identifier>b</Id
```

```
entifier>
                                                                  </VariableExpression
>
                                                               </LeftSide>
                                                               <RightSide>
                                                                  <NumberExpression>1<
/NumberExpression>
                                                               </RightSide>
                                                            </BinaryExpression>
                                                         </Expression>
                                                         <Do>
                                                            <CompoundStatement>
                                                               <Declarations>
                                                               </Declarations>
                                                               <Statments>
                                                                  <ExpressionStatement
>
                                                                     <AssignExpression
>
                                                                        <Variable>
                                                                           <VariableEx
pression>
                                                                              <Identif
ier>b</Identifier>
                                                                           </VariableE
xpression>
                                                                        </Variable>
                                                                        <Value>
                                                                           <BinaryExpr
ession>
                                                                              <Operand
>PLUS</Operand>
                                                                              <LeftSid
e>
                                                                                 <Vari
ableExpression>
                                                                                    <I
dentifier>b</Identifier>
                                                                                 </Var
iableExpression>
                                                                              </LeftSi
de>
                                                                              <RightSi
de>
                                                                                 <Numb
erExpression>2</NumberExpression>
                                                                              </RightS
ide>
                                                                           </BinaryExp
ression>
                                                                        </Value>
                                                                     </AssignExpressio
n>
                                                                  </ExpressionStatemen
t>
                                                               </Statments>
                                                            </CompoundStatement>
                                                         </Do>
                                                      </IterationStatement>
```

```
                                    </Statments>
                                 </CompoundStatement>
                              </Else>
                           </SelectionStatement>
                        </Statments>
                     </CompoundStatement>
                  </Do>
               </IterationStatement>
            </Statments>
         </CompoundStatement>
      </Else>
   </SelectionStatement>
   <SelectionStatement>
      <Expression>
         <BinaryExpression>
            <Operand>EQUALS</Operand>
            <LeftSide>
               <VariableExpression>
                  <Identifier>a</Identifier>
               </VariableExpression>
            </LeftSide>
            <RightSide>
               <VariableExpression>
                  <Identifier>b</Identifier>
               </VariableExpression>
            </RightSide>
         </BinaryExpression>
      </Expression>
      <Then>
         <CompoundStatement>
            <Declarations>
            </Declarations>
            <Statments>
               <IterationStatement>
                  <Expression>
                     <BinaryExpression>
                        <Operand>EQUALS</Operand>
                        <LeftSide>
                           <VariableExpression>
                              <Identifier>a</Identifier>
                           </VariableExpression>
                        </LeftSide>
                        <RightSide>
                           <VariableExpression>
                              <Identifier>b</Identifier>
                           </VariableExpression>
                        </RightSide>
                     </BinaryExpression>
                  </Expression>
                  <Do>
                     <CompoundStatement>
                        <Declarations>
                        </Declarations>
                        <Statments>
                           <IterationStatement>
                              <Expression>
                                 <BinaryExpression>
                                    <Operand>EQUALS</Operand>
                                    <LeftSide>
```

```
                                                <VariableExpression>
                                                   <Identifier>a</Identifier>
                                                </VariableExpression>
                                             </LeftSide>
                                             <RightSide>
                                                <VariableExpression>
                                                   <Identifier>b</Identifier>
                                                </VariableExpression>
                                             </RightSide>
                                          </BinaryExpression>
                                       </Expression>
                                       <Do>
                                          <CompoundStatement>
                                             <Declarations>
                                             </Declarations>
                                             <Statments>
                                                <ExpressionStatement>
                                                   <AssignExpression>
                                                      <Variable>
                                                         <VariableExpression>
                                                            <Identifier>a</Ident
ifier>
                                                         </VariableExpression>
                                                      </Variable>
                                                      <Value>
                                                         <BinaryExpression>
                                                            <Operand>MINUS</Oper
and>
                                                            <LeftSide>
                                                               <VariableExpressi
on>
                                                                  <Identifier>b<
/Identifier>
                                                               </VariableExpress
ion>
                                                            </LeftSide>
                                                            <RightSide>
                                                               <NumberExpression
>1</NumberExpression>
                                                            </RightSide>
                                                         </BinaryExpression>
                                                      </Value>
                                                   </AssignExpression>
                                                </ExpressionStatement>
                                             </Statments>
                                          </CompoundStatement>
                                       </Do>
                                    </IterationStatement>
                                 </Statments>
                              </CompoundStatement>
                           </Do>
                        </IterationStatement>
                     </Statments>
                  </CompoundStatement>
               </Then>
            </SelectionStatement>
            <ExpressionStatement>
               <AssignExpression>
                  <Variable>
```

```
                     <VariableExpression>
                        <Identifier>b</Identifier>
                     </VariableExpression>
                  </Variable>
                  <Value>
                     <NumberExpression>3</NumberExpression>
                  </Value>
               </AssignExpression>
            </ExpressionStatement>
            <ReturnStatement></ReturnStatement>
         </Statments>
      </CompoundStatement>
   </FunctionDeclaration>
</Program>
```

```
int a;

int addThem(int d, int e) {
  int f;
  f = d + e;

  return f;
}

int main (void) {

  int b;
  int c;
  int g;
  int h;
  int i;

  b = 5;

  if (b == 5) {
    a = 3;
  }
  else {
    a = 4;
  }

  g = 0;
  i = 1;
  while (i <= 8) {
    g = g + i;
    i = i+1;
  }
  h = g / 3;
  g = h * 4;

  c = addThem(a, b);
  putchar (c+g);
  putchar (10);

  return 0;
}
```

```
<Program>
   <VariableDeclaration>
      <Name>a</Name>
      <Type>INT</Type>
   </VariableDeclaration>
   <FunctionDeclaration>
      <Name>addThem</Name>
      <ReturnType>INT</ReturnType>
      <Params>
         <VariableDeclaration>
            <Name>d</Name>
            <Type>INT</Type>
         </VariableDeclaration>
         <VariableDeclaration>
            <Name>e</Name>
            <Type>INT</Type>
         </VariableDeclaration>
      </Params>
      <CompoundStatement>
         <Declarations>
            <VariableDeclaration>
               <Name>f</Name>
               <Type>INT</Type>
            </VariableDeclaration>
         </Declarations>
         <Statments>
            <ExpressionStatement>
               <AssignExpression>
                  <Variable>
                     <VariableExpression>
                        <Identifier>f</Identifier>
                     </VariableExpression>
                  </Variable>
                  <Value>
                     <BinaryExpression>
                        <Operand>PLUS</Operand>
                        <LeftSide>
                           <VariableExpression>
                              <Identifier>d</Identifier>
                           </VariableExpression>
                        </LeftSide>
                        <RightSide>
                           <VariableExpression>
                              <Identifier>e</Identifier>
                           </VariableExpression>
                        </RightSide>
                     </BinaryExpression>
                  </Value>
               </AssignExpression>
            </ExpressionStatement>
            <ReturnStatement>
               <Expression>
                  <VariableExpression>
                     <Identifier>f</Identifier>
                  </VariableExpression>
               </Expression>
            </ReturnStatement>
         </Statments>
      </CompoundStatement>
```

```
   </FunctionDeclaration>
   <FunctionDeclaration>
      <Name>main</Name>
      <ReturnType>INT</ReturnType>
      <Params>
      </Params>
      <CompoundStatement>
         <Declarations>
            <VariableDeclaration>
               <Name>b</Name>
               <Type>INT</Type>
            </VariableDeclaration>
            <VariableDeclaration>
               <Name>c</Name>
               <Type>INT</Type>
            </VariableDeclaration>
            <VariableDeclaration>
               <Name>g</Name>
               <Type>INT</Type>
            </VariableDeclaration>
            <VariableDeclaration>
               <Name>h</Name>
               <Type>INT</Type>
            </VariableDeclaration>
            <VariableDeclaration>
               <Name>i</Name>
               <Type>INT</Type>
            </VariableDeclaration>
         </Declarations>
         <Statments>
            <ExpressionStatement>
               <AssignExpression>
                  <Variable>
                     <VariableExpression>
                        <Identifier>b</Identifier>
                     </VariableExpression>
                  </Variable>
                  <Value>
                     <NumberExpression>5</NumberExpression>
                  </Value>
               </AssignExpression>
            </ExpressionStatement>
            <SelectionStatement>
               <Expression>
                  <BinaryExpression>
                     <Operand>EQUALS</Operand>
                     <LeftSide>
                        <VariableExpression>
                           <Identifier>b</Identifier>
                        </VariableExpression>
                     </LeftSide>
                     <RightSide>
                        <NumberExpression>5</NumberExpression>
                     </RightSide>
                  </BinaryExpression>
               </Expression>
               <Then>
                  <CompoundStatement>
                     <Declarations>
```

```
                  </Declarations>
                  <Statments>
                     <ExpressionStatement>
                        <AssignExpression>
                           <Variable>
                              <VariableExpression>
                                 <Identifier>a</Identifier>
                              </VariableExpression>
                           </Variable>
                           <Value>
                              <NumberExpression>3</NumberExpression>
                           </Value>
                        </AssignExpression>
                     </ExpressionStatement>
                  </Statments>
               </CompoundStatement>
            </Then>
            <Else>
               <CompoundStatement>
                  <Declarations>
                  </Declarations>
                  <Statments>
                     <ExpressionStatement>
                        <AssignExpression>
                           <Variable>
                              <VariableExpression>
                                 <Identifier>a</Identifier>
                              </VariableExpression>
                           </Variable>
                           <Value>
                              <NumberExpression>4</NumberExpression>
                           </Value>
                        </AssignExpression>
                     </ExpressionStatement>
                  </Statments>
               </CompoundStatement>
            </Else>
         </SelectionStatement>
         <ExpressionStatement>
            <AssignExpression>
               <Variable>
                  <VariableExpression>
                     <Identifier>g</Identifier>
                  </VariableExpression>
               </Variable>
               <Value>
                  <NumberExpression>0</NumberExpression>
               </Value>
            </AssignExpression>
         </ExpressionStatement>
         <ExpressionStatement>
            <AssignExpression>
               <Variable>
                  <VariableExpression>
                     <Identifier>i</Identifier>
                  </VariableExpression>
               </Variable>
               <Value>
                  <NumberExpression>1</NumberExpression>
```

```
            </Value>
         </AssignExpression>
      </ExpressionStatement>
      <IterationStatement>
         <Expression>
            <BinaryExpression>
               <Operand>LESS_EQUAL_THAN</Operand>
               <LeftSide>
                  <VariableExpression>
                     <Identifier>i</Identifier>
                  </VariableExpression>
               </LeftSide>
               <RightSide>
                  <NumberExpression>8</NumberExpression>
               </RightSide>
            </BinaryExpression>
         </Expression>
         <Do>
            <CompoundStatement>
               <Declarations>
               </Declarations>
               <Statments>
                  <ExpressionStatement>
                     <AssignExpression>
                        <Variable>
                           <VariableExpression>
                              <Identifier>g</Identifier>
                           </VariableExpression>
                        </Variable>
                        <Value>
                           <BinaryExpression>
                              <Operand>PLUS</Operand>
                              <LeftSide>
                                 <VariableExpression>
                                    <Identifier>g</Identifier>
                                 </VariableExpression>
                              </LeftSide>
                              <RightSide>
                                 <VariableExpression>
                                    <Identifier>i</Identifier>
                                 </VariableExpression>
                              </RightSide>
                           </BinaryExpression>
                        </Value>
                     </AssignExpression>
                  </ExpressionStatement>
                  <ExpressionStatement>
                     <AssignExpression>
                        <Variable>
                           <VariableExpression>
                              <Identifier>i</Identifier>
                           </VariableExpression>
                        </Variable>
                        <Value>
                           <BinaryExpression>
                              <Operand>PLUS</Operand>
                              <LeftSide>
                                 <VariableExpression>
                                    <Identifier>i</Identifier>
```

```
                                    </VariableExpression>
                                 </LeftSide>
                                 <RightSide>
                                    <NumberExpression>1</NumberExpression>
                                 </RightSide>
                              </BinaryExpression>
                           </Value>
                        </AssignExpression>
                     </ExpressionStatement>
                  </Statments>
               </CompoundStatement>
            </Do>
         </IterationStatement>
         <ExpressionStatement>
            <AssignExpression>
               <Variable>
                  <VariableExpression>
                     <Identifier>h</Identifier>
                  </VariableExpression>
               </Variable>
               <Value>
                  <BinaryExpression>
                     <Operand>DIVIDE</Operand>
                     <LeftSide>
                        <VariableExpression>
                           <Identifier>g</Identifier>
                        </VariableExpression>
                     </LeftSide>
                     <RightSide>
                        <NumberExpression>3</NumberExpression>
                     </RightSide>
                  </BinaryExpression>
               </Value>
            </AssignExpression>
         </ExpressionStatement>
         <ExpressionStatement>
            <AssignExpression>
               <Variable>
                  <VariableExpression>
                     <Identifier>g</Identifier>
                  </VariableExpression>
               </Variable>
               <Value>
                  <BinaryExpression>
                     <Operand>MULTIPLY</Operand>
                     <LeftSide>
                        <VariableExpression>
                           <Identifier>h</Identifier>
                        </VariableExpression>
                     </LeftSide>
                     <RightSide>
                        <NumberExpression>4</NumberExpression>
                     </RightSide>
                  </BinaryExpression>
               </Value>
            </AssignExpression>
         </ExpressionStatement>
         <ExpressionStatement>
            <AssignExpression>
```

```
                  <Variable>
                     <VariableExpression>
                        <Identifier>c</Identifier>
                     </VariableExpression>
                  </Variable>
                  <Value>
                     <CallExpression>
                        <FunctionName>addThem</FunctionName>
                        <Arguments>
                           <VariableExpression>
                              <Identifier>a</Identifier>
                           </VariableExpression>
                           <VariableExpression>
                              <Identifier>b</Identifier>
                           </VariableExpression>
                        </Arguments>
                     </CallExpression>
                  </Value>
               </AssignExpression>
            </ExpressionStatement>
            <ExpressionStatement>
               <CallExpression>
                  <FunctionName>putchar</FunctionName>
                  <Arguments>
                     <BinaryExpression>
                        <Operand>PLUS</Operand>
                        <LeftSide>
                           <VariableExpression>
                              <Identifier>c</Identifier>
                           </VariableExpression>
                        </LeftSide>
                        <RightSide>
                           <VariableExpression>
                              <Identifier>g</Identifier>
                           </VariableExpression>
                        </RightSide>
                     </BinaryExpression>
                  </Arguments>
               </CallExpression>
            </ExpressionStatement>
            <ExpressionStatement>
               <CallExpression>
                  <FunctionName>putchar</FunctionName>
                  <Arguments>
                     <NumberExpression>10</NumberExpression>
                  </Arguments>
               </CallExpression>
            </ExpressionStatement>
            <ReturnStatement>
               <Expression>
                  <NumberExpression>0</NumberExpression>
               </Expression>
            </ReturnStatement>
         </Statments>
      </CompoundStatement>
   </FunctionDeclaration>
</Program>
```

```
int a;

int addThem(int d, int e) {
  int f;
  f = d + e;

  return f;
}

void putDigit(int s) {
   putchar(48+s);
}

void printInt(int r) {
  int t;
  int found;

  found = 0;

  if (r >= 10000) {
      /* print -1) */
    putchar(45);
    putDigit(1);
    return;
  }
  else {
    if (r >= 1000) {
       t = r / 1000;
       putDigit(t);
       r = r - t * 1000;
       found=1;
    }

    if (r >= 100) {
       t = r / 100;
       putDigit(t);
       r = r - t * 100;
       found=1;
    }
    else if (found == 1) {
       putDigit(0);
    }

    if (r >= 10) {
       t = r / 10;
       putDigit(t);
       r = r - t * 10;
    }
    else if (found == 1) {
       putDigit(0);
    }

    putDigit(r);
  }

}

int main (void) {
```

```
  int b;
  int c;
  int g;
  int h;
  int i;

  b = c = 5;

  if (b == 5) {
    a = 3;
  }
  else {
    a = 4;
  }

  g = 0;
  i = 1;
  while (i <= 8) {
    g = g + i;
    i = i+1;
  }
  h = g / 3;
  g = h * 4;

  c = addThem(a, b);
  putchar (56);
  putchar (61);
  putchar (c+g);
  putchar (10);

  i = 0;
  while (i < 10) {
    putchar(48+i);
    i = i+1;
  }
  putchar(10);
  putchar(67);
  putchar(83);
  printInt(3510);
  putchar(10);

  b = 0;
  c = 1;
  g = 1;
  h = 0;
  i = 0;

  if (b == 0) {
    if (c==0) {
      i = 1;
    }
    else if (g == 0) {
      i = 2;
    }
    else if (h == 0) {
      i = 10;
    }
    else {
      i = 3;
```

```
    }
  }
  else {
    i = 0;
  }


  if (i == 10) {
    putchar(99);
    putDigit(0);
    putDigit(0);
    putchar(108);
  }
  else {
    putchar(98);
    putchar(97);
    putchar(100);
    putchar(61);
    printInt(i);
  }
  putchar(10);

  return 0;
}
```

```
<Program>
   <VariableDeclaration>
      <Name>a</Name>
      <Type>INT</Type>
   </VariableDeclaration>
   <FunctionDeclaration>
      <Name>addThem</Name>
      <ReturnType>INT</ReturnType>
      <Params>
         <VariableDeclaration>
            <Name>d</Name>
            <Type>INT</Type>
         </VariableDeclaration>
         <VariableDeclaration>
            <Name>e</Name>
            <Type>INT</Type>
         </VariableDeclaration>
      </Params>
      <CompoundStatement>
         <Declarations>
            <VariableDeclaration>
               <Name>f</Name>
               <Type>INT</Type>
            </VariableDeclaration>
         </Declarations>
         <Statments>
            <ExpressionStatement>
               <AssignExpression>
                  <Variable>
                     <VariableExpression>
                        <Identifier>f</Identifier>
                     </VariableExpression>
                  </Variable>
                  <Value>
                     <BinaryExpression>
                        <Operand>PLUS</Operand>
                        <LeftSide>
                           <VariableExpression>
                              <Identifier>d</Identifier>
                           </VariableExpression>
                        </LeftSide>
                        <RightSide>
                           <VariableExpression>
                              <Identifier>e</Identifier>
                           </VariableExpression>
                        </RightSide>
                     </BinaryExpression>
                  </Value>
               </AssignExpression>
            </ExpressionStatement>
            <ReturnStatement>
               <Expression>
                  <VariableExpression>
                     <Identifier>f</Identifier>
                  </VariableExpression>
               </Expression>
            </ReturnStatement>
         </Statments>
      </CompoundStatement>
```

```
   </FunctionDeclaration>
   <FunctionDeclaration>
      <Name>putDigit</Name>
      <ReturnType>VOID</ReturnType>
      <Params>
         <VariableDeclaration>
            <Name>s</Name>
            <Type>INT</Type>
         </VariableDeclaration>
      </Params>
      <CompoundStatement>
         <Declarations>
         </Declarations>
         <Statments>
            <ExpressionStatement>
               <CallExpression>
                  <FunctionName>putchar</FunctionName>
                  <Arguments>
                     <BinaryExpression>
                        <Operand>PLUS</Operand>
                        <LeftSide>
                           <NumberExpression>48</NumberExpression>
                        </LeftSide>
                        <RightSide>
                           <VariableExpression>
                              <Identifier>s</Identifier>
                           </VariableExpression>
                        </RightSide>
                     </BinaryExpression>
                  </Arguments>
               </CallExpression>
            </ExpressionStatement>
         </Statments>
      </CompoundStatement>
   </FunctionDeclaration>
   <FunctionDeclaration>
      <Name>printInt</Name>
      <ReturnType>VOID</ReturnType>
      <Params>
         <VariableDeclaration>
            <Name>r</Name>
            <Type>INT</Type>
         </VariableDeclaration>
      </Params>
      <CompoundStatement>
         <Declarations>
            <VariableDeclaration>
               <Name>t</Name>
               <Type>INT</Type>
            </VariableDeclaration>
            <VariableDeclaration>
               <Name>found</Name>
               <Type>INT</Type>
            </VariableDeclaration>
         </Declarations>
         <Statments>
            <ExpressionStatement>
               <AssignExpression>
                  <Variable>
```

```
                  <VariableExpression>
                     <Identifier>found</Identifier>
                  </VariableExpression>
               </Variable>
               <Value>
                  <NumberExpression>0</NumberExpression>
               </Value>
            </AssignExpression>
         </ExpressionStatement>
         <SelectionStatement>
            <Expression>
               <BinaryExpression>
                  <Operand>GREATER_EQUAL_THAN</Operand>
                  <LeftSide>
                     <VariableExpression>
                        <Identifier>r</Identifier>
                     </VariableExpression>
                  </LeftSide>
                  <RightSide>
                     <NumberExpression>10000</NumberExpression>
                  </RightSide>
               </BinaryExpression>
            </Expression>
            <Then>
               <CompoundStatement>
                  <Declarations>
                  </Declarations>
                  <Statments>
                     <ExpressionStatement>
                        <CallExpression>
                           <FunctionName>putchar</FunctionName>
                           <Arguments>
                              <NumberExpression>45</NumberExpression>
                           </Arguments>
                        </CallExpression>
                     </ExpressionStatement>
                     <ExpressionStatement>
                        <CallExpression>
                           <FunctionName>putDigit</FunctionName>
                           <Arguments>
                              <NumberExpression>1</NumberExpression>
                           </Arguments>
                        </CallExpression>
                     </ExpressionStatement>
                     <ReturnStatement></ReturnStatement>
                  </Statments>
               </CompoundStatement>
            </Then>
            <Else>
               <CompoundStatement>
                  <Declarations>
                  </Declarations>
                  <Statments>
                     <SelectionStatement>
                        <Expression>
                           <BinaryExpression>
                              <Operand>GREATER_EQUAL_THAN</Operand>
                              <LeftSide>
                                 <VariableExpression>
```

```
                                    <Identifier>r</Identifier>
                                 </VariableExpression>
                              </LeftSide>
                              <RightSide>
                                 <NumberExpression>1000</NumberExpression>
                              </RightSide>
                           </BinaryExpression>
                        </Expression>
                        <Then>
                           <CompoundStatement>
                              <Declarations>
                              </Declarations>
                              <Statments>
                                 <ExpressionStatement>
                                    <AssignExpression>
                                       <Variable>
                                          <VariableExpression>
                                             <Identifier>t</Identifier>
                                          </VariableExpression>
                                       </Variable>
                                       <Value>
                                          <BinaryExpression>
                                             <Operand>DIVIDE</Operand>
                                             <LeftSide>
                                                <VariableExpression>
                                                   <Identifier>r</Identifier>
                                                </VariableExpression>
                                             </LeftSide>
                                             <RightSide>
                                                <NumberExpression>1000</Numbe
rExpression>
                                             </RightSide>
                                          </BinaryExpression>
                                       </Value>
                                    </AssignExpression>
                                 </ExpressionStatement>
                                 <ExpressionStatement>
                                    <CallExpression>
                                       <FunctionName>putDigit</FunctionName>
                                       <Arguments>
                                          <VariableExpression>
                                             <Identifier>t</Identifier>
                                          </VariableExpression>
                                       </Arguments>
                                    </CallExpression>
                                 </ExpressionStatement>
                                 <ExpressionStatement>
                                    <AssignExpression>
                                       <Variable>
                                          <VariableExpression>
                                             <Identifier>r</Identifier>
                                          </VariableExpression>
                                       </Variable>
                                       <Value>
                                          <BinaryExpression>
                                             <Operand>MINUS</Operand>
                                             <LeftSide>
                                                <VariableExpression>
                                                   <Identifier>r</Identifier>
```

```
                                                </VariableExpression>
                                             </LeftSide>
                                             <RightSide>
                                                <BinaryExpression>
                                                   <Operand>MULTIPLY</Operand
>
                                                   <LeftSide>
                                                      <VariableExpression>
                                                         <Identifier>t</Ident
ifier>
                                                      </VariableExpression>
                                                   </LeftSide>
                                                   <RightSide>
                                                      <NumberExpression>1000<
/NumberExpression>
                                                   </RightSide>
                                                </BinaryExpression>
                                             </RightSide>
                                          </BinaryExpression>
                                       </Value>
                                    </AssignExpression>
                                 </ExpressionStatement>
                                 <ExpressionStatement>
                                    <AssignExpression>
                                       <Variable>
                                          <VariableExpression>
                                             <Identifier>found</Identifier>
                                          </VariableExpression>
                                       </Variable>
                                       <Value>
                                          <NumberExpression>1</NumberExpressi
on>
                                       </Value>
                                    </AssignExpression>
                                 </ExpressionStatement>
                              </Statments>
                           </CompoundStatement>
                        </Then>
                     </SelectionStatement>
                     <SelectionStatement>
                        <Expression>
                           <BinaryExpression>
                              <Operand>GREATER_EQUAL_THAN</Operand>
                              <LeftSide>
                                 <VariableExpression>
                                    <Identifier>r</Identifier>
                                 </VariableExpression>
                              </LeftSide>
                              <RightSide>
                                 <NumberExpression>100</NumberExpression>
                              </RightSide>
                           </BinaryExpression>
                        </Expression>
                        <Then>
                           <CompoundStatement>
                              <Declarations>
                              </Declarations>
                              <Statments>
                                 <ExpressionStatement>
```

```
                                    <AssignExpression>
                                       <Variable>
                                          <VariableExpression>
                                             <Identifier>t</Identifier>
                                          </VariableExpression>
                                       </Variable>
                                       <Value>
                                          <BinaryExpression>
                                             <Operand>DIVIDE</Operand>
                                             <LeftSide>
                                                <VariableExpression>
                                                   <Identifier>r</Identifier>
                                                </VariableExpression>
                                             </LeftSide>
                                             <RightSide>
                                                <NumberExpression>100</Number
Expression>
                                             </RightSide>
                                          </BinaryExpression>
                                       </Value>
                                    </AssignExpression>
                                 </ExpressionStatement>
                                 <ExpressionStatement>
                                    <CallExpression>
                                       <FunctionName>putDigit</FunctionName>
                                       <Arguments>
                                          <VariableExpression>
                                             <Identifier>t</Identifier>
                                          </VariableExpression>
                                       </Arguments>
                                    </CallExpression>
                                 </ExpressionStatement>
                                 <ExpressionStatement>
                                    <AssignExpression>
                                       <Variable>
                                          <VariableExpression>
                                             <Identifier>r</Identifier>
                                          </VariableExpression>
                                       </Variable>
                                       <Value>
                                          <BinaryExpression>
                                             <Operand>MINUS</Operand>
                                             <LeftSide>
                                                <VariableExpression>
                                                   <Identifier>r</Identifier>
                                                </VariableExpression>
                                             </LeftSide>
                                             <RightSide>
                                                <BinaryExpression>
                                                   <Operand>MULTIPLY</Operand
>
                                                   <LeftSide>
                                                      <VariableExpression>
                                                         <Identifier>t</Ident
ifier>
                                                      </VariableExpression>
                                                   </LeftSide>
                                                   <RightSide>
                                                      <NumberExpression>100</
```

```
NumberExpression>
                                                    </RightSide>
                                                 </BinaryExpression>
                                              </RightSide>
                                           </BinaryExpression>
                                        </Value>
                                     </AssignExpression>
                                  </ExpressionStatement>
                                  <ExpressionStatement>
                                     <AssignExpression>
                                        <Variable>
                                           <VariableExpression>
                                              <Identifier>found</Identifier>
                                           </VariableExpression>
                                        </Variable>
                                        <Value>
                                           <NumberExpression>1</NumberExpressi
on>
                                        </Value>
                                     </AssignExpression>
                                  </ExpressionStatement>
                               </Statments>
                            </CompoundStatement>
                         </Then>
                         <Else>
                            <SelectionStatement>
                               <Expression>
                                  <BinaryExpression>
                                     <Operand>EQUALS</Operand>
                                     <LeftSide>
                                        <VariableExpression>
                                           <Identifier>found</Identifier>
                                        </VariableExpression>
                                     </LeftSide>
                                     <RightSide>
                                        <NumberExpression>1</NumberExpression>
                                     </RightSide>
                                  </BinaryExpression>
                               </Expression>
                               <Then>
                                  <CompoundStatement>
                                     <Declarations>
                                     </Declarations>
                                     <Statments>
                                        <ExpressionStatement>
                                           <CallExpression>
                                              <FunctionName>putDigit</Function
Name>
                                              <Arguments>
                                                 <NumberExpression>0</NumberEx
pression>
                                              </Arguments>
                                           </CallExpression>
                                        </ExpressionStatement>
                                     </Statments>
                                  </CompoundStatement>
                               </Then>
                            </SelectionStatement>
                         </Else>
```

```
                    </SelectionStatement>
                    <SelectionStatement>
                       <Expression>
                          <BinaryExpression>
                             <Operand>GREATER_EQUAL_THAN</Operand>
                             <LeftSide>
                                <VariableExpression>
                                   <Identifier>r</Identifier>
                                </VariableExpression>
                             </LeftSide>
                             <RightSide>
                                <NumberExpression>10</NumberExpression>
                             </RightSide>
                          </BinaryExpression>
                       </Expression>
                       <Then>
                          <CompoundStatement>
                             <Declarations>
                             </Declarations>
                             <Statments>
                                <ExpressionStatement>
                                   <AssignExpression>
                                      <Variable>
                                         <VariableExpression>
                                            <Identifier>t</Identifier>
                                         </VariableExpression>
                                      </Variable>
                                      <Value>
                                         <BinaryExpression>
                                            <Operand>DIVIDE</Operand>
                                            <LeftSide>
                                               <VariableExpression>
                                                  <Identifier>r</Identifier>
                                               </VariableExpression>
                                            </LeftSide>
                                            <RightSide>
                                               <NumberExpression>10</NumberE
xpression>
                                            </RightSide>
                                         </BinaryExpression>
                                      </Value>
                                   </AssignExpression>
                                </ExpressionStatement>
                                <ExpressionStatement>
                                   <CallExpression>
                                      <FunctionName>putDigit</FunctionName>
                                      <Arguments>
                                         <VariableExpression>
                                            <Identifier>t</Identifier>
                                         </VariableExpression>
                                      </Arguments>
                                   </CallExpression>
                                </ExpressionStatement>
                                <ExpressionStatement>
                                   <AssignExpression>
                                      <Variable>
                                         <VariableExpression>
                                            <Identifier>r</Identifier>
                                         </VariableExpression>
```

```
                                    </Variable>
                                    <Value>
                                       <BinaryExpression>
                                          <Operand>MINUS</Operand>
                                          <LeftSide>
                                             <VariableExpression>
                                                <Identifier>r</Identifier>
                                             </VariableExpression>
                                          </LeftSide>
                                          <RightSide>
                                             <BinaryExpression>
                                                <Operand>MULTIPLY</Operand
>
                                                <LeftSide>
                                                   <VariableExpression>
                                                      <Identifier>t</Ident
ifier>
                                                   </VariableExpression>
                                                </LeftSide>
                                                <RightSide>
                                                   <NumberExpression>10</N
umberExpression>
                                                </RightSide>
                                             </BinaryExpression>
                                          </RightSide>
                                       </BinaryExpression>
                                    </Value>
                                 </AssignExpression>
                              </ExpressionStatement>
                           </Statments>
                        </CompoundStatement>
                     </Then>
                     <Else>
                        <SelectionStatement>
                           <Expression>
                              <BinaryExpression>
                                 <Operand>EQUALS</Operand>
                                 <LeftSide>
                                    <VariableExpression>
                                       <Identifier>found</Identifier>
                                    </VariableExpression>
                                 </LeftSide>
                                 <RightSide>
                                    <NumberExpression>1</NumberExpression>
                                 </RightSide>
                              </BinaryExpression>
                           </Expression>
                           <Then>
                              <CompoundStatement>
                                 <Declarations>
                                 </Declarations>
                                 <Statments>
                                    <ExpressionStatement>
                                       <CallExpression>
                                          <FunctionName>putDigit</Function
Name>
                                          <Arguments>
                                             <NumberExpression>0</NumberEx
pression>
```

```
                                                </Arguments>
                                             </CallExpression>
                                          </ExpressionStatement>
                                       </Statments>
                                    </CompoundStatement>
                                 </Then>
                              </SelectionStatement>
                           </Else>
                        </SelectionStatement>
                        <ExpressionStatement>
                           <CallExpression>
                              <FunctionName>putDigit</FunctionName>
                              <Arguments>
                                 <VariableExpression>
                                    <Identifier>r</Identifier>
                                 </VariableExpression>
                              </Arguments>
                           </CallExpression>
                        </ExpressionStatement>
                     </Statments>
                  </CompoundStatement>
               </Else>
            </SelectionStatement>
         </Statments>
      </CompoundStatement>
   </FunctionDeclaration>
   <FunctionDeclaration>
      <Name>main</Name>
      <ReturnType>INT</ReturnType>
      <Params>
      </Params>
      <CompoundStatement>
         <Declarations>
            <VariableDeclaration>
               <Name>b</Name>
               <Type>INT</Type>
            </VariableDeclaration>
            <VariableDeclaration>
               <Name>c</Name>
               <Type>INT</Type>
            </VariableDeclaration>
            <VariableDeclaration>
               <Name>g</Name>
               <Type>INT</Type>
            </VariableDeclaration>
            <VariableDeclaration>
               <Name>h</Name>
               <Type>INT</Type>
            </VariableDeclaration>
            <VariableDeclaration>
               <Name>i</Name>
               <Type>INT</Type>
            </VariableDeclaration>
         </Declarations>
         <Statments>
            <ExpressionStatement>
               <AssignExpression>
                  <Variable>
                     <VariableExpression>
```

```
                  <Identifier>b</Identifier>
               </VariableExpression>
            </Variable>
            <Value>
               <AssignExpression>
                  <Variable>
                     <VariableExpression>
                        <Identifier>c</Identifier>
                     </VariableExpression>
                  </Variable>
                  <Value>
                     <NumberExpression>5</NumberExpression>
                  </Value>
               </AssignExpression>
            </Value>
         </AssignExpression>
      </ExpressionStatement>
      <SelectionStatement>
         <Expression>
            <BinaryExpression>
               <Operand>EQUALS</Operand>
               <LeftSide>
                  <VariableExpression>
                     <Identifier>b</Identifier>
                  </VariableExpression>
               </LeftSide>
               <RightSide>
                  <NumberExpression>5</NumberExpression>
               </RightSide>
            </BinaryExpression>
         </Expression>
         <Then>
            <CompoundStatement>
               <Declarations>
               </Declarations>
               <Statments>
                  <ExpressionStatement>
                     <AssignExpression>
                        <Variable>
                           <VariableExpression>
                              <Identifier>a</Identifier>
                           </VariableExpression>
                        </Variable>
                        <Value>
                           <NumberExpression>3</NumberExpression>
                        </Value>
                     </AssignExpression>
                  </ExpressionStatement>
               </Statments>
            </CompoundStatement>
         </Then>
         <Else>
            <CompoundStatement>
               <Declarations>
               </Declarations>
               <Statments>
                  <ExpressionStatement>
                     <AssignExpression>
                        <Variable>
```

```
                        <VariableExpression>
                           <Identifier>a</Identifier>
                        </VariableExpression>
                     </Variable>
                     <Value>
                        <NumberExpression>4</NumberExpression>
                     </Value>
                  </AssignExpression>
               </ExpressionStatement>
            </Statments>
         </CompoundStatement>
      </Else>
   </SelectionStatement>
   <ExpressionStatement>
      <AssignExpression>
         <Variable>
            <VariableExpression>
               <Identifier>g</Identifier>
            </VariableExpression>
         </Variable>
         <Value>
            <NumberExpression>0</NumberExpression>
         </Value>
      </AssignExpression>
   </ExpressionStatement>
   <ExpressionStatement>
      <AssignExpression>
         <Variable>
            <VariableExpression>
               <Identifier>i</Identifier>
            </VariableExpression>
         </Variable>
         <Value>
            <NumberExpression>1</NumberExpression>
         </Value>
      </AssignExpression>
   </ExpressionStatement>
   <IterationStatement>
      <Expression>
         <BinaryExpression>
            <Operand>LESS_EQUAL_THAN</Operand>
            <LeftSide>
               <VariableExpression>
                  <Identifier>i</Identifier>
               </VariableExpression>
            </LeftSide>
            <RightSide>
               <NumberExpression>8</NumberExpression>
            </RightSide>
         </BinaryExpression>
      </Expression>
      <Do>
         <CompoundStatement>
            <Declarations>
            </Declarations>
            <Statments>
               <ExpressionStatement>
                  <AssignExpression>
                     <Variable>
```

```
                              <VariableExpression>
                                 <Identifier>g</Identifier>
                              </VariableExpression>
                           </Variable>
                           <Value>
                              <BinaryExpression>
                                 <Operand>PLUS</Operand>
                                 <LeftSide>
                                    <VariableExpression>
                                       <Identifier>g</Identifier>
                                    </VariableExpression>
                                 </LeftSide>
                                 <RightSide>
                                    <VariableExpression>
                                       <Identifier>i</Identifier>
                                    </VariableExpression>
                                 </RightSide>
                              </BinaryExpression>
                           </Value>
                        </AssignExpression>
                     </ExpressionStatement>
                     <ExpressionStatement>
                        <AssignExpression>
                           <Variable>
                              <VariableExpression>
                                 <Identifier>i</Identifier>
                              </VariableExpression>
                           </Variable>
                           <Value>
                              <BinaryExpression>
                                 <Operand>PLUS</Operand>
                                 <LeftSide>
                                    <VariableExpression>
                                       <Identifier>i</Identifier>
                                    </VariableExpression>
                                 </LeftSide>
                                 <RightSide>
                                    <NumberExpression>1</NumberExpression>
                                 </RightSide>
                              </BinaryExpression>
                           </Value>
                        </AssignExpression>
                     </ExpressionStatement>
                  </Statments>
               </CompoundStatement>
            </Do>
         </IterationStatement>
         <ExpressionStatement>
            <AssignExpression>
               <Variable>
                  <VariableExpression>
                     <Identifier>h</Identifier>
                  </VariableExpression>
               </Variable>
               <Value>
                  <BinaryExpression>
                     <Operand>DIVIDE</Operand>
                     <LeftSide>
                        <VariableExpression>
```

```
                        <Identifier>g</Identifier>
                     </VariableExpression>
                  </LeftSide>
                  <RightSide>
                     <NumberExpression>3</NumberExpression>
                  </RightSide>
               </BinaryExpression>
            </Value>
         </AssignExpression>
      </ExpressionStatement>
      <ExpressionStatement>
         <AssignExpression>
            <Variable>
               <VariableExpression>
                  <Identifier>g</Identifier>
               </VariableExpression>
            </Variable>
            <Value>
               <BinaryExpression>
                  <Operand>MULTIPLY</Operand>
                  <LeftSide>
                     <VariableExpression>
                        <Identifier>h</Identifier>
                     </VariableExpression>
                  </LeftSide>
                  <RightSide>
                     <NumberExpression>4</NumberExpression>
                  </RightSide>
               </BinaryExpression>
            </Value>
         </AssignExpression>
      </ExpressionStatement>
      <ExpressionStatement>
         <AssignExpression>
            <Variable>
               <VariableExpression>
                  <Identifier>c</Identifier>
               </VariableExpression>
            </Variable>
            <Value>
               <CallExpression>
                  <FunctionName>addThem</FunctionName>
                  <Arguments>
                     <VariableExpression>
                        <Identifier>a</Identifier>
                     </VariableExpression>
                     <VariableExpression>
                        <Identifier>b</Identifier>
                     </VariableExpression>
                  </Arguments>
               </CallExpression>
            </Value>
         </AssignExpression>
      </ExpressionStatement>
      <ExpressionStatement>
         <CallExpression>
            <FunctionName>putchar</FunctionName>
            <Arguments>
               <NumberExpression>56</NumberExpression>
```

```
            </Arguments>
         </CallExpression>
      </ExpressionStatement>
      <ExpressionStatement>
         <CallExpression>
            <FunctionName>putchar</FunctionName>
            <Arguments>
               <NumberExpression>61</NumberExpression>
            </Arguments>
         </CallExpression>
      </ExpressionStatement>
      <ExpressionStatement>
         <CallExpression>
            <FunctionName>putchar</FunctionName>
            <Arguments>
               <BinaryExpression>
                  <Operand>PLUS</Operand>
                  <LeftSide>
                     <VariableExpression>
                        <Identifier>c</Identifier>
                     </VariableExpression>
                  </LeftSide>
                  <RightSide>
                     <VariableExpression>
                        <Identifier>g</Identifier>
                     </VariableExpression>
                  </RightSide>
               </BinaryExpression>
            </Arguments>
         </CallExpression>
      </ExpressionStatement>
      <ExpressionStatement>
         <CallExpression>
            <FunctionName>putchar</FunctionName>
            <Arguments>
               <NumberExpression>10</NumberExpression>
            </Arguments>
         </CallExpression>
      </ExpressionStatement>
      <ExpressionStatement>
         <AssignExpression>
            <Variable>
               <VariableExpression>
                  <Identifier>i</Identifier>
               </VariableExpression>
            </Variable>
            <Value>
               <NumberExpression>0</NumberExpression>
            </Value>
         </AssignExpression>
      </ExpressionStatement>
      <IterationStatement>
         <Expression>
            <BinaryExpression>
               <Operand>LESS_THAN</Operand>
               <LeftSide>
                  <VariableExpression>
                     <Identifier>i</Identifier>
                  </VariableExpression>
```

```
                  </LeftSide>
                  <RightSide>
                     <NumberExpression>10</NumberExpression>
                  </RightSide>
               </BinaryExpression>
            </Expression>
            <Do>
               <CompoundStatement>
                  <Declarations>
                  </Declarations>
                  <Statments>
                     <ExpressionStatement>
                        <CallExpression>
                           <FunctionName>putchar</FunctionName>
                           <Arguments>
                              <BinaryExpression>
                                 <Operand>PLUS</Operand>
                                 <LeftSide>
                                    <NumberExpression>48</NumberExpression>
                                 </LeftSide>
                                 <RightSide>
                                    <VariableExpression>
                                       <Identifier>i</Identifier>
                                    </VariableExpression>
                                 </RightSide>
                              </BinaryExpression>
                           </Arguments>
                        </CallExpression>
                     </ExpressionStatement>
                     <ExpressionStatement>
                        <AssignExpression>
                           <Variable>
                              <VariableExpression>
                                 <Identifier>i</Identifier>
                              </VariableExpression>
                           </Variable>
                           <Value>
                              <BinaryExpression>
                                 <Operand>PLUS</Operand>
                                 <LeftSide>
                                    <VariableExpression>
                                       <Identifier>i</Identifier>
                                    </VariableExpression>
                                 </LeftSide>
                                 <RightSide>
                                    <NumberExpression>1</NumberExpression>
                                 </RightSide>
                              </BinaryExpression>
                           </Value>
                        </AssignExpression>
                     </ExpressionStatement>
                  </Statments>
               </CompoundStatement>
            </Do>
         </IterationStatement>
         <ExpressionStatement>
            <CallExpression>
               <FunctionName>putchar</FunctionName>
               <Arguments>
```

```
            <NumberExpression>10</NumberExpression>
         </Arguments>
      </CallExpression>
   </ExpressionStatement>
   <ExpressionStatement>
      <CallExpression>
         <FunctionName>putchar</FunctionName>
         <Arguments>
            <NumberExpression>67</NumberExpression>
         </Arguments>
      </CallExpression>
   </ExpressionStatement>
   <ExpressionStatement>
      <CallExpression>
         <FunctionName>putchar</FunctionName>
         <Arguments>
            <NumberExpression>83</NumberExpression>
         </Arguments>
      </CallExpression>
   </ExpressionStatement>
   <ExpressionStatement>
      <CallExpression>
         <FunctionName>printInt</FunctionName>
         <Arguments>
            <NumberExpression>3510</NumberExpression>
         </Arguments>
      </CallExpression>
   </ExpressionStatement>
   <ExpressionStatement>
      <CallExpression>
         <FunctionName>putchar</FunctionName>
         <Arguments>
            <NumberExpression>10</NumberExpression>
         </Arguments>
      </CallExpression>
   </ExpressionStatement>
   <ExpressionStatement>
      <AssignExpression>
         <Variable>
            <VariableExpression>
               <Identifier>b</Identifier>
            </VariableExpression>
         </Variable>
         <Value>
            <NumberExpression>0</NumberExpression>
         </Value>
      </AssignExpression>
   </ExpressionStatement>
   <ExpressionStatement>
      <AssignExpression>
         <Variable>
            <VariableExpression>
               <Identifier>c</Identifier>
            </VariableExpression>
         </Variable>
         <Value>
            <NumberExpression>1</NumberExpression>
         </Value>
      </AssignExpression>
```

```
            </ExpressionStatement>
            <ExpressionStatement>
               <AssignExpression>
                  <Variable>
                     <VariableExpression>
                        <Identifier>g</Identifier>
                     </VariableExpression>
                  </Variable>
                  <Value>
                     <NumberExpression>1</NumberExpression>
                  </Value>
               </AssignExpression>
            </ExpressionStatement>
            <ExpressionStatement>
               <AssignExpression>
                  <Variable>
                     <VariableExpression>
                        <Identifier>h</Identifier>
                     </VariableExpression>
                  </Variable>
                  <Value>
                     <NumberExpression>0</NumberExpression>
                  </Value>
               </AssignExpression>
            </ExpressionStatement>
            <ExpressionStatement>
               <AssignExpression>
                  <Variable>
                     <VariableExpression>
                        <Identifier>i</Identifier>
                     </VariableExpression>
                  </Variable>
                  <Value>
                     <NumberExpression>0</NumberExpression>
                  </Value>
               </AssignExpression>
            </ExpressionStatement>
            <SelectionStatement>
               <Expression>
                  <BinaryExpression>
                     <Operand>EQUALS</Operand>
                     <LeftSide>
                        <VariableExpression>
                           <Identifier>b</Identifier>
                        </VariableExpression>
                     </LeftSide>
                     <RightSide>
                        <NumberExpression>0</NumberExpression>
                     </RightSide>
                  </BinaryExpression>
               </Expression>
               <Then>
                  <CompoundStatement>
                     <Declarations>
                     </Declarations>
                     <Statments>
                        <SelectionStatement>
                           <Expression>
                              <BinaryExpression>
```

```
                        <Operand>EQUALS</Operand>
                        <LeftSide>
                           <VariableExpression>
                              <Identifier>c</Identifier>
                           </VariableExpression>
                        </LeftSide>
                        <RightSide>
                           <NumberExpression>0</NumberExpression>
                        </RightSide>
                     </BinaryExpression>
                  </Expression>
                  <Then>
                     <CompoundStatement>
                        <Declarations>
                        </Declarations>
                        <Statments>
                           <ExpressionStatement>
                              <AssignExpression>
                                 <Variable>
                                    <VariableExpression>
                                       <Identifier>i</Identifier>
                                    </VariableExpression>
                                 </Variable>
                                 <Value>
                                    <NumberExpression>1</NumberExpressi
on>

                                 </Value>
                              </AssignExpression>
                           </ExpressionStatement>
                        </Statments>
                     </CompoundStatement>
                  </Then>
                  <Else>
                     <SelectionStatement>
                        <Expression>
                           <BinaryExpression>
                              <Operand>EQUALS</Operand>
                              <LeftSide>
                                 <VariableExpression>
                                    <Identifier>g</Identifier>
                                 </VariableExpression>
                              </LeftSide>
                              <RightSide>
                                 <NumberExpression>0</NumberExpression>
                              </RightSide>
                           </BinaryExpression>
                        </Expression>
                        <Then>
                           <CompoundStatement>
                              <Declarations>
                              </Declarations>
                              <Statments>
                                 <ExpressionStatement>
                                    <AssignExpression>
                                       <Variable>
                                          <VariableExpression>
                                             <Identifier>i</Identifier>
                                          </VariableExpression>
                                       </Variable>
```

```
                                                <Value>
                                                   <NumberExpression>2</NumberEx
pression>
                                                </Value>
                                             </AssignExpression>
                                          </ExpressionStatement>
                                       </Statments>
                                    </CompoundStatement>
                                 </Then>
                                 <Else>
                                    <SelectionStatement>
                                       <Expression>
                                          <BinaryExpression>
                                             <Operand>EQUALS</Operand>
                                             <LeftSide>
                                                <VariableExpression>
                                                   <Identifier>h</Identifier>
                                                </VariableExpression>
                                             </LeftSide>
                                             <RightSide>
                                                <NumberExpression>0</NumberExpre
ssion>
                                             </RightSide>
                                          </BinaryExpression>
                                       </Expression>
                                       <Then>
                                          <CompoundStatement>
                                             <Declarations>
                                             </Declarations>
                                             <Statments>
                                                <ExpressionStatement>
                                                   <AssignExpression>
                                                      <Variable>
                                                         <VariableExpression>
                                                            <Identifier>i</Ident
ifier>
                                                         </VariableExpression>
                                                      </Variable>
                                                      <Value>
                                                         <NumberExpression>10</N
umberExpression>
                                                      </Value>
                                                   </AssignExpression>
                                                </ExpressionStatement>
                                             </Statments>
                                          </CompoundStatement>
                                       </Then>
                                       <Else>
                                          <CompoundStatement>
                                             <Declarations>
                                             </Declarations>
                                             <Statments>
                                                <ExpressionStatement>
                                                   <AssignExpression>
                                                      <Variable>
                                                         <VariableExpression>
                                                            <Identifier>i</Ident
ifier>
                                                         </VariableExpression>
```

```
                                                                </Variable>
                                                                <Value>
                                                                   <NumberExpression>3</Nu
mberExpression>
                                                                </Value>
                                                             </AssignExpression>
                                                          </ExpressionStatement>
                                                       </Statments>
                                                    </CompoundStatement>
                                                 </Else>
                                              </SelectionStatement>
                                           </Else>
                                        </SelectionStatement>
                                     </Else>
                                  </SelectionStatement>
                               </Statments>
                            </CompoundStatement>
                         </Then>
                         <Else>
                            <CompoundStatement>
                               <Declarations>
                               </Declarations>
                               <Statments>
                                  <ExpressionStatement>
                                     <AssignExpression>
                                        <Variable>
                                           <VariableExpression>
                                              <Identifier>i</Identifier>
                                           </VariableExpression>
                                        </Variable>
                                        <Value>
                                           <NumberExpression>0</NumberExpression>
                                        </Value>
                                     </AssignExpression>
                                  </ExpressionStatement>
                               </Statments>
                            </CompoundStatement>
                         </Else>
                      </SelectionStatement>
                      <SelectionStatement>
                         <Expression>
                            <BinaryExpression>
                               <Operand>EQUALS</Operand>
                               <LeftSide>
                                  <VariableExpression>
                                     <Identifier>i</Identifier>
                                  </VariableExpression>
                               </LeftSide>
                               <RightSide>
                                  <NumberExpression>10</NumberExpression>
                               </RightSide>
                            </BinaryExpression>
                         </Expression>
                         <Then>
                            <CompoundStatement>
                               <Declarations>
                               </Declarations>
                               <Statments>
                                  <ExpressionStatement>
```

```
                        <CallExpression>
                           <FunctionName>putchar</FunctionName>
                           <Arguments>
                              <NumberExpression>99</NumberExpression>
                           </Arguments>
                        </CallExpression>
                     </ExpressionStatement>
                     <ExpressionStatement>
                        <CallExpression>
                           <FunctionName>putDigit</FunctionName>
                           <Arguments>
                              <NumberExpression>0</NumberExpression>
                           </Arguments>
                        </CallExpression>
                     </ExpressionStatement>
                     <ExpressionStatement>
                        <CallExpression>
                           <FunctionName>putDigit</FunctionName>
                           <Arguments>
                              <NumberExpression>0</NumberExpression>
                           </Arguments>
                        </CallExpression>
                     </ExpressionStatement>
                     <ExpressionStatement>
                        <CallExpression>
                           <FunctionName>putchar</FunctionName>
                           <Arguments>
                              <NumberExpression>108</NumberExpression>
                           </Arguments>
                        </CallExpression>
                     </ExpressionStatement>
                  </Statments>
               </CompoundStatement>
            </Then>
            <Else>
               <CompoundStatement>
                  <Declarations>
                  </Declarations>
                  <Statments>
                     <ExpressionStatement>
                        <CallExpression>
                           <FunctionName>putchar</FunctionName>
                           <Arguments>
                              <NumberExpression>98</NumberExpression>
                           </Arguments>
                        </CallExpression>
                     </ExpressionStatement>
                     <ExpressionStatement>
                        <CallExpression>
                           <FunctionName>putchar</FunctionName>
                           <Arguments>
                              <NumberExpression>97</NumberExpression>
                           </Arguments>
                        </CallExpression>
                     </ExpressionStatement>
                     <ExpressionStatement>
                        <CallExpression>
                           <FunctionName>putchar</FunctionName>
                           <Arguments>
```

```
                              <NumberExpression>100</NumberExpression>
                           </Arguments>
                        </CallExpression>
                     </ExpressionStatement>
                     <ExpressionStatement>
                        <CallExpression>
                           <FunctionName>putchar</FunctionName>
                           <Arguments>
                              <NumberExpression>61</NumberExpression>
                           </Arguments>
                        </CallExpression>
                     </ExpressionStatement>
                     <ExpressionStatement>
                        <CallExpression>
                           <FunctionName>printInt</FunctionName>
                           <Arguments>
                              <VariableExpression>
                                 <Identifier>i</Identifier>
                              </VariableExpression>
                           </Arguments>
                        </CallExpression>
                     </ExpressionStatement>
                  </Statments>
               </CompoundStatement>
            </Else>
         </SelectionStatement>
         <ExpressionStatement>
            <CallExpression>
               <FunctionName>putchar</FunctionName>
               <Arguments>
                  <NumberExpression>10</NumberExpression>
               </Arguments>
            </CallExpression>
         </ExpressionStatement>
         <ReturnStatement>
            <Expression>
               <NumberExpression>0</NumberExpression>
            </Expression>
         </ReturnStatement>
      </Statments>
   </CompoundStatement>
</FunctionDeclaration>
</Program>
```

```
int fact( int x )
/* recursive factorial function */
{ if (x > 1)
   return x * fact(x−1);
else
    return 1;
}

void main (void)
{
   int x;
    x = read();
   if(x > 0) write (fact(x));
}
```

```
<Program>
   <FunctionDeclaration>
      <Name>fact</Name>
      <ReturnType>INT</ReturnType>
      <Params>
         <VariableDeclaration>
            <Name>x</Name>
            <Type>INT</Type>
         </VariableDeclaration>
      </Params>
      <CompoundStatement>
         <Declarations>
         </Declarations>
         <Statments>
            <SelectionStatement>
               <Expression>
                  <BinaryExpression>
                     <Operand>GREATER_THAN</Operand>
                     <LeftSide>
                        <VariableExpression>
                           <Identifier>x</Identifier>
                        </VariableExpression>
                     </LeftSide>
                     <RightSide>
                        <NumberExpression>1</NumberExpression>
                     </RightSide>
                  </BinaryExpression>
               </Expression>
               <Then>
                  <ReturnStatement>
                     <Expression>
                        <BinaryExpression>
                           <Operand>MULTIPLY</Operand>
                           <LeftSide>
                              <VariableExpression>
                                 <Identifier>x</Identifier>
                              </VariableExpression>
                           </LeftSide>
                           <RightSide>
                              <CallExpression>
                                 <FunctionName>fact</FunctionName>
                                 <Arguments>
                                    <BinaryExpression>
                                       <Operand>MINUS</Operand>
                                       <LeftSide>
                                          <VariableExpression>
                                             <Identifier>x</Identifier>
                                          </VariableExpression>
                                       </LeftSide>
                                       <RightSide>
                                          <NumberExpression>1</NumberExpression>
                                       </RightSide>
                                    </BinaryExpression>
                                 </Arguments>
                              </CallExpression>
                           </RightSide>
                        </BinaryExpression>
                     </Expression>
                  </ReturnStatement>
```

```
            </Then>
            <Else>
               <ReturnStatement>
                  <Expression>
                     <NumberExpression>1</NumberExpression>
                  </Expression>
               </ReturnStatement>
            </Else>
         </SelectionStatement>
      </Statments>
   </CompoundStatement>
</FunctionDeclaration>
<FunctionDeclaration>
   <Name>main</Name>
   <ReturnType>VOID</ReturnType>
   <Params>
   </Params>
   <CompoundStatement>
      <Declarations>
         <VariableDeclaration>
            <Name>x</Name>
            <Type>INT</Type>
         </VariableDeclaration>
      </Declarations>
      <Statments>
         <ExpressionStatement>
            <AssignExpression>
               <Variable>
                  <VariableExpression>
                     <Identifier>x</Identifier>
                  </VariableExpression>
               </Variable>
               <Value>
                  <CallExpression>
                     <FunctionName>read</FunctionName>
                     <Arguments>
                     </Arguments>
                  </CallExpression>
               </Value>
            </AssignExpression>
         </ExpressionStatement>
         <SelectionStatement>
            <Expression>
               <BinaryExpression>
                  <Operand>GREATER_THAN</Operand>
                  <LeftSide>
                     <VariableExpression>
                        <Identifier>x</Identifier>
                     </VariableExpression>
                  </LeftSide>
                  <RightSide>
                     <NumberExpression>0</NumberExpression>
                  </RightSide>
               </BinaryExpression>
            </Expression>
            <Then>
               <ExpressionStatement>
                  <CallExpression>
                     <FunctionName>write</FunctionName>
```

```
                        <Arguments>
                           <CallExpression>
                              <FunctionName>fact</FunctionName>
                              <Arguments>
                                 <VariableExpression>
                                    <Identifier>x</Identifier>
                                 </VariableExpression>
                              </Arguments>
                           </CallExpression>
                        </Arguments>
                     </CallExpression>
                  </ExpressionStatement>
               </Then>
            </SelectionStatement>
         </Statments>
      </CompoundStatement>
   </FunctionDeclaration>
</Program>
```

```
#!/bin/bash
a2ps -R -s1 --media=letter --columns=1 --rows=1 --chars-per-line=80 --major=rows
 --border=no --tabsize=3 src/parser/* src/parser/expression/* src/parser/stateme
nt/* tests/* print.sh -o - | ps2pdf - Program.pdf
pdfunite write-ups/Project2WriteUp.pdf Grammar-MkII.pdf First-Follow-MkII.pdf Pr
ogram.pdf Printout.pdf
rm Program.pdf
```