```
//Jacob Longar
/**
*This part of the compiler parses code from output from the
* languageScanner class. This part also handles error checking
* regarding any grammar or syntax errors produced by the input
* Pascal program.
*
* @author        Jacob Longar
* @version       1.0
*/
import java.io.*;
import java.lang.*;
import java.util.Scanner;
import java.util.StringTokenizer;
import java.util.ArrayList;

/**
* a class created to hold data for our symbols / identifiers
* fed to this program through the languageScanner program.
* @param Identifier holds identifier information from languageScanner
* @param Symbol holds the actual character found in languageScanner
* @param Level  integer value indicating what scope a value is in
* @param ProcedureName this value paired with level let us know
* which scope a symbol exists in.
* @param DataType if the symbol / Identifier is a variable,
* this parameter lets us know which data type that variable is.
* (useful mainly in final part of code for compiler project)
* @param Value if the symbol / Identifier is a variable, this
* parameter tells us what value the variable is associated with.
*/
```

```
class CharacterStruct
{
        public String Identifier;
   public String Symbol;
   public int    Level;
   public String ProcedureName;
   public String DataType;
   public String Value;


   public CharacterStruct (String Identifier, String Symbol, int Level, String ProcedureName, String
DataType, String Value)
   {
        this.Identifier = Identifier;
        this.Symbol = Symbol;
        this.Level = Level;
        this.ProcedureName = ProcedureName;
        this.DataType = DataType;
        this.Value = Value;
   }
};


public class parser
{
        /**
        * This program reads in data from languageScanner.java
        * and also performs error checking and parsing analysis
        * of a pascal program through the use of functions
        * documented below.
        * @param isError if an error occurs while parsing, then this boolean
        * value will become true. If this value is true, no code will finish
```

```
* compiling.
* @throws IOException  If an input or output
*                  exception occurred
*/
public static boolean isError = false;
public static void main(String args[]) throws IOException
{
        String inputVal = "a.out";

        //file IO
        FileReader fileReader = new FileReader(inputVal);
        BufferedReader bufferedReader = new BufferedReader(fileReader);
        StringBuffer lineReader = new StringBuffer();

        //reading lines from the file outputted by the languageScanner code
        String line;
        while ((line = bufferedReader.readLine()) != null)
        {
                lineReader.append(line);
                lineReader.append("\n");
        }

        fileReader.close();

        //creating a tokenizer to read each input from the language Scanner
        StringTokenizer tokenizer = new StringTokenizer(lineReader.toString());

        //initializing our array of structures
        ArrayList<CharacterStruct> cStruct = new ArrayList<CharacterStruct>();
```

```
		//assigning the identifier and symbol for each struct in the ArrayList
		//and assigning default values for all other values for the time being
		int tokenCount = 0;
		while (tokenizer.hasMoreTokens())
		{
			cStruct.add(new CharacterStruct(tokenizer.nextToken(), tokenizer.nextToken(),
0,"default","default","default"));
			tokenCount++;
		}

		//initializing our data structure
		CharacterStruct characterstruct;

		//used for conversion from CharacterStruct to String
		//so we can use string functions on values from data input.
		String temp;

		//creating a large loop to test for any errors while parsing.
		//This section will also act as starter code to the final part of //the project.
		//This will be the main part of project #2 as our output to the
		//final part of the project will simply be an array of structs.
		for (int u = 0; u < tokenCount; u++)
		{
			//setting our structvalue equal to the current data value
			characterstruct = cStruct.get(u);
			temp = characterstruct.Identifier;

			if (u == 0)
			{
				isProgram(cStruct);
```

```
                }


                if (temp.contains("LPAREN"))
                {
                        isParenth(cStruct, u, tokenCount);
                }
                if (temp.contains("RBRACKET"))
                {
                        isSquareBrack(cStruct, u, tokenCount);
                }
                if (temp.contains("MINUS") || temp.contains("PLUS") ||
temp.contains("TIMES") || temp.contains("LESSEQUAL") || temp.contains("GREATEREQUAL") ||
temp.contains("EQUAL") || temp.contains("NOTEQUAL") || temp.contains("LESSTHAN") ||
temp.contains("GREATERTHAN"))
                {
                        isEquation(cStruct, u, tokenCount);
                }
                if (temp.contains("VAR"))
                {
                        initialize(cStruct, u, tokenCount);
                }
                if (temp.contains("PROGRAM") || temp.contains("PROCEDURE") ||
temp.contains("FUNCTION"))
                {
                        isBegin(cStruct, u, tokenCount);
                }
                //for two from the end
                if (u == (tokenCount - 2))
                {
                        isEnd(cStruct,u, tokenCount);
                }
        }
```

```
}

/**
* this program checks to see if the pascal program's first
* non-comment command is the word 'program' followed by an ID
* @param cStruct the ArrayList populated with data from languageScanner
**/
public static void isProgram(ArrayList<CharacterStruct> cStruct)
{
	CharacterStruct characterstruct = cStruct.get(0);
	String temp = characterstruct.Identifier;
	if (temp.contains("PROGRAM"))
	{
		characterstruct = cStruct.get(1);
		// System.out.println("The name of the program is ");
		// System.out.print(characterstruct.Symbol);
	}
	else
	{
		isError = true;
		System.out.println("Please specify the name of the program");
		System.exit(0);
	}
}

/**
* This program checks to see if there are a correct number of
* parantheses on each line of the program.
* @param cStruct the ArrayList populated with data from languageScanner
* @param u counter from the main program so we know where we are in the program
```

```
* @param tokenCount from the main program, used so we know what the max
* limit of the array is (this is primarily done to avoid using an
*  arbitrary number to signify the end of the loop used here).
*/
public static void isParenth(ArrayList<CharacterStruct> cStruct, int u, int tokenCount)
{
	String temp;
	int parencount = 1;
	CharacterStruct characterstruct = cStruct.get(u);
	for (u = u; u < tokenCount; u++)
	{
		characterstruct = cStruct.get(u);
		temp = characterstruct.Identifier;
		if (parencount == 0)
		{
			break;
		}
		if (temp.contains("SEMICOLON"))
		{
			System.out.println("Missing right parenthese on this line ");

			isError = true;
			break;
		}
		if (temp.contains("LPAREN"))
		{
			parencount++;
		}
		if (temp.contains("RPAREN"))
		{
```

```
                                        //not sure why, but this needs to be subtracted twice
                                        parencount--;
                                        parencount--;
                                        if (parencount == 0)
                                        {
                                                //System.out.println("there are a correct number of
parentheses on this line");
                                                break;
                                        }
                                }
                        }
                }


                        /**
                * This program checks to see if there are a correct number of
                * square brackets on each line of the program.
                * @param cStruct the ArrayList populated with data from languageScanner
                * @param u counter from the main program so we know where we are in the program
                * @param tokenCount from the main program, used so we know what the max
                * limit of the array is (this is primarily done to avoid using an
                * arbitrary number to signify the end of the loop used here).
                */
                public static void isSquareBrack(ArrayList<CharacterStruct> cStruct, int u, int tokenCount)
                {
                        String temp;
                        int bracketcount = 1;
                        CharacterStruct characterstruct = cStruct.get(u);
                        for (u = u; u < tokenCount; u++)
                        {
```

```
                    characterstruct = cStruct.get(u);
                    temp = characterstruct.Identifier;
                    if (bracketcount == 0)
                    {
                            break;
                    }
                    if (temp.contains("SEMICOLON"))
                    {
                            System.out.println("Missing right parenthese on this line ");

                            isError = true;
                            break;
                    }
                    if (temp.contains("LBRACKET"))
                    {
                            bracketcount++;
                    }
                    if (temp.contains("RBRACKET"))
                    {
                            //not sure why, but this needs to be subtracted twice
                            bracketcount--;
                            bracketcount--;
                            if (bracketcount == 0)
                            {
                                    //System.out.println("there are a correct number of
parentheses on this line");
                                    break;
                            }
                    }
            }
```

```
}

/**
* This function checks to see if an equation is valid
* based off of the right and left operands of a math function
* @param cStruct the ArrayList populated with data from languageScanner
* @param u counter from the main program so we know where we are in the program
* @param tokenCount from the main program, used so we know what the max
* limit of the array is (this is primarily done to avoid using an
* arbitrary number to signify the end of the loop used here).
*/
public static void isEquation(ArrayList<CharacterStruct> cStruct, int u, int tokenCount)
{
	String temp;
	CharacterStruct characterstruct = cStruct.get(u);
	temp = characterstruct.Identifier;

	//creating an infinite loop until broken
	while (true)
	{
		characterstruct = cStruct.get(u - 1);
		temp = characterstruct.Identifier;
		if (temp.contains("RBRACKET") || temp.contains("RPAREN"))
		{
			//we continue
		}
		if (temp.contains("NUMBER") || temp.contains("ID"))
		{
			//We want to find a number or ID so we break the loop
			break;
```

```
		}

		else
		{
			System.out.println("Could not find a left value to the operand");
			isError = true;
			break;
		}
	}

	while(true)
	{
		characterstruct = cStruct.get(u + 1);
		temp = characterstruct.Identifier;
		if (temp.contains("LBRACKET") || temp.contains("LPAREN"))
		{
			//we continue
		}
		if (temp.contains("NUMBER") || temp.contains("ID"))
		{
			//We want to find a number or ID so we break the loop
			break;
		}
		else
		{
			System.out.println("Could not find a right value to the operand");
			isError = true;
			break;
		}
	}
```

```
}

/**
* This program will ensure that values are initilized
* before they are used in a Pascal program
* @param cStruct the ArrayList populated with data from languageScanner
* @param u counter from the main program so we know where we are in the program
* @param tokenCount from the main program, used so we know what the max
* limit of the array is (this is primarily done to avoid using an
* arbitrary number to signify the end of the loop used here).
*/
public static void initialize(ArrayList<CharacterStruct> cStruct, int u, int tokenCount)
{
	ArrayList initialized = new ArrayList();
	String temp;
	CharacterStruct characterstruct = cStruct.get(u);
	temp = characterstruct.Identifier;

	while(true)
	{
		characterstruct = cStruct.get(u + 1);
		temp = characterstruct.Identifier;
		if (!temp.contains("ID"))
		{
			System.out.println("Please provide an ID");
			isError = true;
			break;
		}
		else
		{
```

```
                                        initialized.add(characterstruct.Symbol);


                                        characterstruct = cStruct.get(u + 1);
                                        temp = characterstruct.Identifier;
                                        if (temp.contains("COLON"))
                                        {
                                                characterstruct = cStruct.get(u + 1);
                                                temp = characterstruct.Identifier;
                                                if (temp.contains("INTEGER") || temp.contains("CHAR") ||
temp.contains("STRING") || temp.contains("CHR"))
                                                {
                                                        characterstruct = cStruct.get(u - 2);
                                                        initialized.add(characterstruct.DataType);
                                                }
                                        }
                                }
                        }
                }

                /**
                * this function checks to see if the program and procedures
                * both have "Begin" statements
                * @param cStruct the ArrayList populated with data from languageScanner
                * @param u counter from the main program so we know where we are in the program
                * @param tokenCount from the main program, used so we know what the max
                * limit of the array is (this is primarily done to avoid using an
                * arbitrary number to signify the end of the loop used here).
                */
                public static void isBegin (ArrayList<CharacterStruct> cStruct, int u, int tokenCount)
                {
```

```
                String temp;
                CharacterStruct characterstruct = cStruct.get(u);
                temp = characterstruct.Identifier;

                //get to the end of the line
                //assuming the user ended the line with a semicolon
                while(true)
                {
                        characterstruct = cStruct.get(u + 1);
                        temp = characterstruct.Identifier;
                        if (temp.contains("SEMICOLON"))
                        {
                                break;
                        }
                }
                characterstruct = cStruct.get(u + 1);
                temp = characterstruct.Identifier;
                if (!temp.contains("BEGIN"))
                {
                        System.out.println("program or procedure does not have a
\"begin\"statement");
                        isError = true;
                }
        }
        /**
        * This function will check to see if the user
        * calls the "End." command at the end of the program
        * @param cStruct the ArrayList populated with data from languageScanner
        * @param tokenCount from the main program, used so we know what the max
        * limit of the array is (this is primarily done to avoid using an
```

```
        * arbitrary number to signify the end of the loop used here).
        */
        public static void isEnd(ArrayList<CharacterStruct> cStruct, int u, int tokenCount)
        {
                String temp;
                CharacterStruct characterstruct = cStruct.get(u);
                temp = characterstruct.Identifier;

                if (!temp.contains("END"))
                {
                        System.out.println("Program does not have an \"End\" statement");
                        isError = true;
                }
                characterstruct = cStruct.get(u + 1);
                temp = characterstruct.Identifier;
                if (!temp.contains("PERIOD"))
                {
                        System.out.println("\"End\" statement is missing a period");
                        isError = true;
                }
        }
}
```