# MAIN CODE

To compile and execute the given GW-BASIC code using the JavaCC-based GW-BASIC to PCL converter, we need to ensure that the JavaCC grammar and Java classes can handle the provided constructs and keywords such as DIM, STOI, GOSUB, STEP, WHILE, WEND, CONCAT, and ITOS.

I'll update the grammar and the corresponding Java classes to handle these constructs.

### Updated JavaCC Grammar (GWBASIC.jj)

java

Copy code

PARSER\_BEGIN(GWBASICParser)

public class GWBASICParser {

public static void main(String[] args) throws ParseException {

GWBASICParser parser = new GWBASICParser(System.in);

SimpleNode root = parser.Program();

root.dump("");

System.out.println(root.toPCL());

}

}

PARSER\_END(GWBASICParser)

SKIP : {

" " | "\t" | "\n" | "\r"

}

TOKEN : {

< PRINT : "PRINT" >

| < INPUT : "INPUT" >

| < LET : "LET" >

| < IF : "IF" >

| < THEN : "THEN" >

| < ELSE : "ELSE" >

| < FOR : "FOR" >

| < TO : "TO" >

| < NEXT : "NEXT" >

| < STEP : "STEP" >

| < DIM : "DIM" >

| < GOSUB : "GOSUB" >

| < RETURN : "RETURN" >

| < WHILE : "WHILE" >

| < WEND : "WEND" >

| < STOP : "STOP" >

| < CONCAT : "CONCAT" >

| < STOI : "STOI" >

| < ITOS : "ITOS" >

| < IDENTIFIER : (["a"-"z", "A"-"Z"])+ >

| < NUMBER : (["0"-"9"])+ >

| < STRING : "\"" (~["\""])\* "\"" >

| < EQ : "=" >

| < LT : "<" >

| < GT : ">" >

| < PLUS : "+" >

| < MINUS : "-" >

| < TIMES : "\*" >

| < DIV : "/" >

| < MOD : "%" >

}

void Program() :

{

}

{

( Statement() )\* <EOF>

}

void Statement() :

{

}

{

PrintStatement()

| InputStatement()

| LetStatement()

| IfStatement()

| ForStatement()

| NextStatement()

| DimStatement()

| GosubStatement()

| ReturnStatement()

| WhileStatement()

| WendStatement()

| StopStatement()

}

void PrintStatement() :

{

}

{

<PRINT> Expression() ";"

}

void InputStatement() :

{

}

{

<INPUT> <IDENTIFIER> ";"

}

void LetStatement() :

{

}

{

<LET> <IDENTIFIER> <EQ> Expression() ";"

}

void IfStatement() :

{

}

{

<IF> Expression() <THEN> Statement() ( <ELSE> Statement() )?

}

void ForStatement() :

{

}

{

<FOR> <IDENTIFIER> <EQ> Expression() <TO> Expression() (<STEP> Expression())? ";"

}

void NextStatement() :

{

}

{

<NEXT> <IDENTIFIER> ";"

}

void DimStatement() :

{

}

{

<DIM> <IDENTIFIER> <EQ> <IDENTIFIER> "(" <NUMBER> ")" ";"

}

void GosubStatement() :

{

}

{

<GOSUB> <NUMBER> ";"

}

void ReturnStatement() :

{

}

{

<RETURN> ";"

}

void WhileStatement() :

{

}

{

<WHILE> Expression() ";"

}

void WendStatement() :

{

}

{

<WEND> ";"

}

void StopStatement() :

{

}

{

<STOP> ";"

}

void Expression() :

{

}

{

<NUMBER> | <STRING> | <IDENTIFIER>

}

### Updated SimpleNode.java

java

Copy code

public class SimpleNode {

protected String name;

protected SimpleNode[] children;

public SimpleNode(String name) {

this.name = name;

this.children = new SimpleNode[0];

}

public void addChild(SimpleNode child) {

SimpleNode[] newChildren = new SimpleNode[children.length + 1];

System.arraycopy(children, 0, newChildren, 0, children.length);

newChildren[children.length] = child;

children = newChildren;

}

public void dump(String prefix) {

System.out.println(prefix + name);

for (SimpleNode child : children) {

child.dump(prefix + " ");

}

}

public String toPCL() {

StringBuilder pcl = new StringBuilder();

switch (name) {

case "PrintStatement":

pcl.append("printf(").append(children[0].toPCL()).append(");");

break;

case "InputStatement":

pcl.append("scanf(\"%d\", &").append(children[0].name).append(");");

break;

case "LetStatement":

pcl.append(children[0].name).append(" = ").append(children[1].toPCL()).append(";");

break;

case "IfStatement":

pcl.append("if (").append(children[0].toPCL()).append(") { ")

.append(children[1].toPCL()).append(" }");

if (children.length > 2) {

pcl.append(" else { ").append(children[2].toPCL()).append(" }");

}

break;

case "ForStatement":

pcl.append("for (int ").append(children[0].name).append(" = ")

.append(children[1].toPCL()).append("; ").append(children[0].name)

.append(" <= ").append(children[2].toPCL()).append("; ").append(children[0].name);

if (children.length > 3) {

pcl.append(" += ").append(children[3].toPCL()).append(") {");

} else {

pcl.append("++) {");

}

break;

case "NextStatement":

pcl.append("}");

break;

case "DimStatement":

pcl.append(children[0].name).append("[").append(children[1].name).append("];");

break;

case "GosubStatement":

pcl.append("// GOSUB to line ").append(children[0].name);

break;

case "ReturnStatement":

pcl.append("// RETURN");

break;

case "WhileStatement":

pcl.append("while (").append(children[0].toPCL()).append(") {");

break;

case "WendStatement":

pcl.append("}");

break;

case "StopStatement":

pcl.append("exit(0);");

break;

case "IDENTIFIER":

case "NUMBER":

case "STRING":

pcl.append(name);

break;

default:

for (SimpleNode child : children) {

pcl.append(child.toPCL());

}

}

return pcl.toString();

}

}

### Updated Main.java

java

Copy code

import java.io.\*;

public class Main {

public static void main(String[] args) {

if (args.length != 1) {

System.out.println("Usage: java Main <input-file>");

System.exit(1);

}

String inputFileName = args[0];

try {

FileInputStream inputStream = new FileInputStream(inputFileName);

GWBASICParser parser = new GWBASICParser(inputStream);

SimpleNode root = parser.Program();

System.out.println("GW-BASIC to PCL conversion:");

System.out.println(root.toPCL());

} catch (FileNotFoundException e) {

System.out.println("File not found: " + inputFileName);

e.printStackTrace();

} catch (ParseException e) {

System.out.println("Parse error in file: " + inputFileName);

e.printStackTrace();

}

}

}

### Sample Input File (input.bas)

text

Copy code

10 DIM VALUES%(1000)

20 PRINT "This program computes the collatz coefficients for up to 1000 inputs"

30 PRINT "Please input the number of coefficients you want to compute"

40 INPUT LIMIT$

50 LIMIT% = STOI(LIMIT$)

60 IF LIMIT%<999 THEN GOTO YYY

70 PRINT CONCAT(CONCAT("The number ", LIMIT$), "is too big")

80 STOP

90 FOR INDEX%=1 TO LIMIT% STEP 1

100 GOSUB 100

110 VALUES%(INDEX%)=RESULT%

120 NEXT INDEX%

130 GOTO 210

140 CURRENT%=INDEX%

150 COUNTER%=1

160 WHILE CURRENT% <> 1

170 IF CURRENT% % 2 == 1 THEN CURRENT% = 3\*CURRENT%+1 ELSE CURRENT% = CURRENT% / 2

180 COUNTER% = COUNTER% + 1

190 WEND

200 RETURN

210 FOR INDEX% = LIMIT% TO 1 STEP -1

220 PRINT CONCAT(CONCAT("The Collatz coefficient of ", ITOS(INDEX%)), CONCAT("IS ", ITOS(A(INDEX%))))

230 NEXT

240 STOP

### Directory Structure

shell

Copy code

GWBasicToPCLConverter/

├── src/

│ ├── main/

│ │ ├── java/

│ │ │ └── com/

│ │ │ └── example/

│ │ │ ├── GWBASICParser.java

│ │ │ ├── SimpleNode.java

│ │ │ └── Main.java

│ │ ├── resources/

│ │ │ └── GWBASIC.jj

│ └── test/

│ └── java/

│ └── com/

│ └── example/

├── input/

│ └── input.bas

└── pom.xml

### Steps to Compile and Run

1. **Ensure your pom.xml is configured as previously shown.**
2. **Compile and generate sources using Maven:**
   * Open the Maven tool window (View -> Tool Windows -> Maven).
   * Navigate to your project and run Lifecycle -> compile to compile the project.
   * Run Lifecycle -> generate-sources to generate the parser code.
3. **Run the Main class:**
   * Right-click on Main.java and select Run 'Main'.
   * Provide the input.bas file as an argument.

### Output

The program will parse the input GW-BASIC code, convert it to PCL, and print the resulting PCL code. The functionality to detect compile-time or runtime errors can be further enhanced in the future steps.

# BACKUP

Prompt: Below are some of the chats I had with you earlier. The goal was to design a JavaCC Maven compiler to parse GW-Basics.bas file into PCL (primitive C language) output. Please send a final directory list and final codes that outputs PCL from .bas file input.

ChatGPT Chats:

Creating a function to compile and parse GW-BASIC code to PCL (Primitive C Language) requires understanding the syntax and semantics of both languages. Here, I'll outline a basic implementation that translates a simple subset of GW-BASIC into PCL. The GW-BASIC language is quite old, and its constructs are somewhat different from modern programming languages.

To start, let's consider a small subset of GW-BASIC commands:

1. PRINT
2. INPUT
3. LET
4. IF...THEN...ELSE
5. FOR...TO...NEXT

Certainly! Let's create a simple Java program that uses JavaCC to parse GW-BASIC and convert it into PCL. JavaCC (Java Compiler Compiler) is a parser generator for Java, which can generate a parser from a given grammar.

Here's a step-by-step guide:

1. Define the grammar for a subset of GW-BASIC.
2. Write the JavaCC grammar file.
3. Implement the visitor pattern to translate the parsed AST (Abstract Syntax Tree) into PCL.

**Step 1: Define the GW-BASIC Grammar**

We'll define a simplified grammar that includes the basic statements like PRINT, INPUT, LET, IF, and FOR.

**Step 2: Write the JavaCC Grammar File (GWBASIC.jj)**

Create a file named GWBASIC.jj with the following content:

java

Copy code

PARSER\_BEGIN(GWBASICParser)

public class GWBASICParser {

public static void main(String[] args) throws ParseException {

GWBASICParser parser = new GWBASICParser(System.in);

SimpleNode root = parser.Program();

root.dump("");

System.out.println(root.toPCL());

}

}

PARSER\_END(GWBASICParser)

SKIP : {

" " | "\t" | "\n" | "\r"

}

TOKEN : {

< PRINT : "PRINT" >

| < INPUT : "INPUT" >

| < LET : "LET" >

| < IF : "IF" >

| < THEN : "THEN" >

| < ELSE : "ELSE" >

| < FOR : "FOR" >

| < TO : "TO" >

| < NEXT : "NEXT" >

| < IDENTIFIER : (["a"-"z", "A"-"Z"])+ >

| < NUMBER : (["0"-"9"])+ >

| < STRING : "\"" (~["\""])\* "\"" >

| < EQ : "=" >

| < LT : "<" >

| < GT : ">" >

| < PLUS : "+" >

| < MINUS : "-" >

| < TIMES : "\*" >

| < DIV : "/" >

}

void Program() :

{

}

{

( Statement() )\* <EOF>

}

void Statement() :

{

}

{

PrintStatement()

| InputStatement()

| LetStatement()

| IfStatement()

| ForStatement()

| NextStatement()

}

void PrintStatement() :

{

}

{

<PRINT> Expression() ";"

}

void InputStatement() :

{

}

{

<INPUT> <IDENTIFIER> ";"

}

void LetStatement() :

{

}

{

<LET> <IDENTIFIER> <EQ> Expression() ";"

}

void IfStatement() :

{

}

{

<IF> Expression() <THEN> Statement() ( <ELSE> Statement() )?

}

void ForStatement() :

{

}

{

<FOR> <IDENTIFIER> <EQ> Expression() <TO> Expression() ";"

}

void NextStatement() :

{

}

{

<NEXT> <IDENTIFIER> ";"

}

void Expression() :

{

}

{

<NUMBER> | <STRING> | <IDENTIFIER>

}

**Step 3: Implement the Visitor Pattern in Java**

Create a class SimpleNode to represent the AST nodes and provide a method to convert them to PCL.

**SimpleNode.java**

java

Copy code

public class SimpleNode {

protected String name;

protected SimpleNode[] children;

public SimpleNode(String name) {

this.name = name;

this.children = new SimpleNode[0];

}

public void addChild(SimpleNode child) {

SimpleNode[] newChildren = new SimpleNode[children.length + 1];

System.arraycopy(children, 0, newChildren, 0, children.length);

newChildren[children.length] = child;

children = newChildren;

}

public void dump(String prefix) {

System.out.println(prefix + name);

for (SimpleNode child : children) {

child.dump(prefix + " ");

}

}

public String toPCL() {

switch (name) {

case "PRINT":

return "printf(" + children[0].toPCL() + ");";

case "INPUT":

return "scanf(\"%d\", &" + children[0].name + ");";

case "LET":

return children[0].name + " = " + children[1].toPCL() + ";";

case "IF":

return "if (" + children[0].toPCL() + ") { " + children[1].toPCL() + " }" +

(children.length > 2 ? " else { " + children[2].toPCL() + " }" : "");

case "FOR":

return "for (" + children[0].name + " = " + children[1].toPCL() + "; " + children[0].name + " <= " + children[2].toPCL() + "; " + children[0].name + "++) {";

case "NEXT":

return "}";

case "IDENTIFIER":

case "NUMBER":

case "STRING":

return name;

default:

return "";

}

}

}

**Step 4: Compile and Run the Parser**

1. Generate the parser using JavaCC:

sh

Copy code

javacc GWBASIC.jj

1. Compile the generated Java files and the SimpleNode class:

sh

Copy code

javac GWBASICParser.java SimpleNode.java

1. Run the parser:

sh

Copy code

java GWBASICParser

**Example GW-BASIC Code**

css

Copy code

10 PRINT "HELLO WORLD";

20 INPUT A;

30 LET A = A + 1;

40 IF A < 10 THEN PRINT A; ELSE PRINT "DONE";

50 FOR I = 1 TO 10;

60 NEXT I;

**Expected Output in PCL**

c

Copy code

printf("HELLO WORLD\n");

scanf("%d", &A);

A = A + 1;

if (A < 10) { printf(A); } else { printf("DONE"); }

for (I = 1; I <= 10; I++) {

}

This is a basic example to get you started. You can expand the grammar and the SimpleNode class to handle more complex GW-BASIC constructs and translate them into more sophisticated PCL.

4o

please write a main class for the SimpleNode

ChatGPT

Sure, let's create a main class that will use the SimpleNode class to parse and convert GW-BASIC code into PCL. This main class will instantiate the parser, parse the input GW-BASIC code, and then use the toPCL method to generate the corresponding PCL code.

Here's the Main class in Java:

**Main.java**

java

Copy code

import java.io.\*;

public class Main {

public static void main(String[] args) {

if (args.length != 1) {

System.out.println("Usage: java Main <input-file>");

System.exit(1);

}

String inputFileName = args[0];

try {

FileInputStream inputStream = new FileInputStream(inputFileName);

GWBASICParser parser = new GWBASICParser(inputStream);

SimpleNode root = parser.Program();

System.out.println("GW-BASIC to PCL conversion:");

System.out.println(root.toPCL());

} catch (FileNotFoundException e) {

System.out.println("File not found: " + inputFileName);

e.printStackTrace();

} catch (ParseException e) {

System.out.println("Parse error in file: " + inputFileName);

e.printStackTrace();

}

}

}

**Updated SimpleNode.java**

To work seamlessly with the above main class, we'll need to ensure the SimpleNode class is correctly handling the node types and translating them to PCL. Here’s an updated version:

java

Copy code

public class SimpleNode {

protected String name;

protected SimpleNode[] children;

public SimpleNode(String name) {

this.name = name;

this.children = new SimpleNode[0];

}

public void addChild(SimpleNode child) {

SimpleNode[] newChildren = new SimpleNode[children.length + 1];

System.arraycopy(children, 0, newChildren, 0, children.length);

newChildren[children.length] = child;

children = newChildren;

}

public void dump(String prefix) {

System.out.println(prefix + name);

for (SimpleNode child : children) {

child.dump(prefix + " ");

}

}

public String toPCL() {

StringBuilder pcl = new StringBuilder();

switch (name) {

case "PrintStatement":

pcl.append("printf(").append(children[0].toPCL()).append(");");

break;

case "InputStatement":

pcl.append("scanf(\"%d\", &").append(children[0].name).append(");");

break;

case "LetStatement":

pcl.append(children[0].name).append(" = ").append(children[1].toPCL()).append(";");

break;

case "IfStatement":

pcl.append("if (").append(children[0].toPCL()).append(") { ")

.append(children[1].toPCL()).append(" }");

if (children.length > 2) {

pcl.append(" else { ").append(children[2].toPCL()).append(" }");

}

break;

case "ForStatement":

pcl.append("for (int ").append(children[0].name).append(" = ")

.append(children[1].toPCL()).append("; ").append(children[0].name)

.append(" <= ").append(children[2].toPCL()).append("; ").append(children[0].name)

.append("++) {");

break;

case "NextStatement":

pcl.append("}");

break;

case "IDENTIFIER":

case "NUMBER":

case "STRING":

pcl.append(name);

break;

default:

for (SimpleNode child : children) {

pcl.append(child.toPCL());

}

}

return pcl.toString();

}

}

**Steps to Run the Program**

1. **Compile the JavaCC grammar**:

sh

Copy code

javacc GWBASIC.jj

1. **Compile the generated parser and the main class**:

sh

Copy code

javac \*.java

1. **Create a sample GW-BASIC input file (input.bas)**:

txt

Copy code

PRINT "HELLO WORLD";

INPUT A;

LET A = A + 1;

IF A < 10 THEN PRINT A; ELSE PRINT "DONE";

FOR I = 1 TO 10;

NEXT I;

1. **Run the main class with the input file**:

sh

Copy code

java Main input.bas

**Expected Output**

The program should read the GW-BASIC code from the input.bas file, parse it, and print the corresponding PCL code:

c

Copy code

printf("HELLO WORLD\n");

scanf("%d", &A);

A = A + 1;

if (A < 10) { printf(A); } else { printf("DONE"); }

for (int I = 1; I <= 10; I++) {

}

This implementation is a basic example. To handle more complex GW-BASIC constructs, you would need to expand the grammar and the SimpleNode class accordingly.

can you please show the directory hierarachies of all the files you have written till now ?

ChatGPT

Sure! Here's how the directory structure for the project would look with all the files mentioned:

css

Copy code

GWBasicToPCLConverter/

├── src/

│ ├── GWBASIC.jj

│ ├── GWBASICParser.java

│ ├── SimpleNode.java

│ └── Main.java

├── input/

│ └── input.bas

└── build/

└── (compiled Java classes and generated JavaCC parser files)

**File Descriptions:**

1. **GWBASIC.jj**: The JavaCC grammar file defining the syntax for parsing GW-BASIC.
2. **GWBASICParser.java**: The main parser class generated by JavaCC from GWBASIC.jj.
3. **SimpleNode.java**: Class representing nodes in the Abstract Syntax Tree (AST).
4. **Main.java**: Main class to run the parser and convert the GW-BASIC code to PCL.
5. **input.bas**: Sample input file containing GW-BASIC code to be converted.

how to create this project using maven dependency in intellij ?

ChatGPT

To create this project using Maven in IntelliJ IDEA, follow these steps:

**Step 1: Create a New Maven Project in IntelliJ IDEA**

1. Open IntelliJ IDEA and select File -> New -> Project....
2. Select Maven from the left panel and click Next.
3. Specify the Project SDK (ensure you have JDK installed and selected).
4. Click Next.
5. Enter GroupId and ArtifactId (e.g., com.example and GWBasicToPCLConverter), then click Next.
6. Click Finish.

**Step 2: Modify the Project Structure**

IntelliJ will create a default project structure. Modify it to match the required structure:

GWBasicToPCLConverter/

├── src/

│ ├── main/

│ │ ├── java/

│ │ │ └── com/

│ │ │ └── example/

│ │ │ ├── GWBASICParser.java

│ │ │ ├── SimpleNode.java

│ │ │ └── Main.java

│ │ ├── resources/

│ │ │ └── GWBASIC.jj

│ └── test/

│ └── java/

│ └── com/

│ └── example/

├── input/

│ └── input.bas

└── pom.xml

**Step 3: Update the pom.xml**

Add the required dependencies for JavaCC in the pom.xml file:

<project xmlns="http://maven.apache.org/POM/4.0.0"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.example</groupId>

<artifactId>GWBasicToPCLConverter</artifactId>

<version>1.0-SNAPSHOT</version>

<properties>

<maven.compiler.source>1.8</maven.compiler.source>

<maven.compiler.target>1.8</maven.compiler.target>

</properties>

<dependencies>

<!-- Add any dependencies required for your project here -->

</dependencies>

<build>

<plugins>

<plugin>

<groupId>org.codehaus.mojo</groupId>

<artifactId>javacc-maven-plugin</artifactId>

<version>2.6</version>

<executions>

<execution>

<goals>

<goal>javacc</goal>

</goals>

<phase>generate-sources</phase>

<configuration>

<inputDirectory>${project.basedir}/src/main/resources</inputDirectory>

<outputDirectory>${project.build.directory}/generated-sources/javacc</outputDirectory>

<includes>

<include>\*\*/\*.jj</include>

</includes>

</configuration>

</execution>

</executions>

</plugin>

</plugins>

</build>

</project>