EGRE591 Compiler Project
Part 2 - Syntax Analysis
Spring 2024

1 Syntax Specification of `toyc`

ToyCProgram → { Definition } **EOF**

Definition → Type **identifier** (FunctionDefinition | ;)

Type → *int* | *char*

FunctionDefinition → FunctionHeader FunctionBody
FunctionHeader → ([FormalParamList])
FunctionBody → CompoundStatement
FormalParamList → Type **identifier** { , Type **identifier** }

Statement → ExpressionStatement
| BreakStatement
| CompoundStatement
| IfStatement
| NullStatement
| ReturnStatement
| WhileStatement
| ReadStatement
| WriteStatement
| NewLineStatement

ExpressionStatement → Expression ;

BreakStatement → *break* ;

CompoundStatement → **{** { Type **identifier** ; } { Statement } **}**

IfStatement → *if* (Expression) Statement [*else* Statement]

NullStatement → ;

ReturnStatement → *return* [Expression] ;

WhileStatement → *while* (Expression) Statement

ReadStatement → *read* (**identifier** { , **identifier** }) ;

WriteStatement	→	*write* $\underline{(}$ ActualParameters $\underline{)}$ $\underline{;}$
NewLineStatement	→	*newline* $\underline{;}$
Expression	→	RelopExpression { **assignop** RelopExpression }
RelopExpression	→	SimpleExpression { **relop** SimpleExpression }
SimpleExpression	→	Term { **addop** Term }
Term	→	Primary { **mulop** Primary }
Primary	→	**identifier** [FunctionCall]
		| **number**
		| **stringConstant**
		| **charConstant**
		| $\underline{(}$ Expression $\underline{)}$
		| ($\underline{-}$ | **not**) Primary
FunctionCall	→	$\underline{(}$ [ActualParameters] $\underline{)}$
ActualParameters	→	Expression { $\underline{,}$ Expression }

Question: Is it LL(1)?

2 `toyc` Example Program

```
int isEven(int n){
 if ((n % 2) == 0)
  return 1;
 else
  return 0;
}

int main() {
 int i;
 i = 1;
 while (i <= 10) {
  write(i);
  if ( isEven(i) )
   write(" is even");
  else
   write(" is odd");
  newline;
  i = i + 1;
 }
 return;
}
```

3 Abstract Syntax

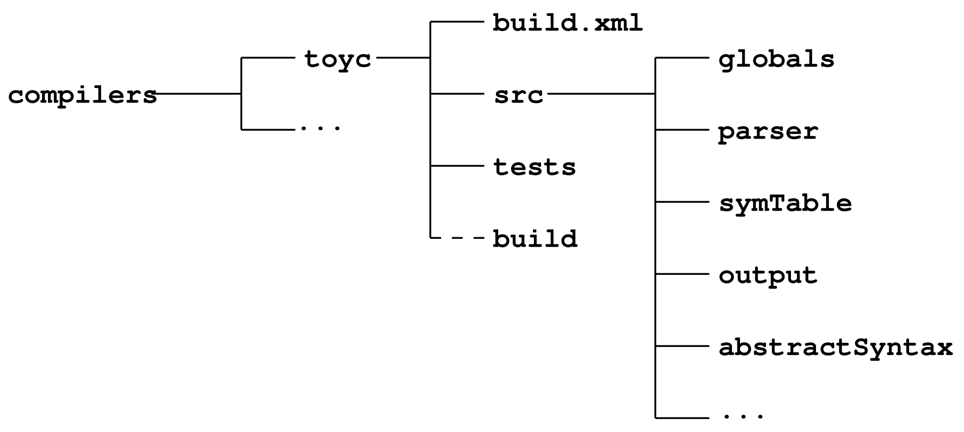
When the appropriate diagnostic flag is set your parser will output an abstract syntax tree for every successful parse.

3.1 Grammar

Program	$\triangleq$	*prog*(Definition*)
Definition	$\triangleq$	*funcDef*(Id, Type, *varDef*(Id$^+$, Type)*, Statement) $\mid$ *varDef*(Id$^+$, Type)
Statement	$\triangleq$	*exprState*(Expression)
		$\mid$ *breakState*()
		$\mid$ *blockState*(*varDef*(Id$^+$, Type)*, Statement*)
		$\mid$ *ifState*(Expression, Statement, Statement?)
		$\mid$ *nullState*()
		$\mid$ *returnState*(Expression?)
		$\mid$ *whileState*(Expression, Statement)
		$\mid$ *readState*(Id$^+$)
		$\mid$ *writeState*(Expression$^+$)
		$\mid$ *newLineState*()
Expression	$\triangleq$	Number $\mid$ Identifier $\mid$ CharLiteral $\mid$ StringLiteral
		$\mid$ *funcCall*(Identifier, Expression*)
		$\mid$ *expr*(Operator, Expression, Expression)
		$\mid$ *minus*(Expression)
		$\mid$ *not*(Expression)
Operator	$\triangleq$	$+ \mid - \mid * \mid / \mid \% \mid$ `||` $\mid$ `&&` $\mid\ <= \mid\ < \mid\ = \mid\ > \mid\ >= \mid\ != $

4 Program Development

4.1 Java Projects



Your base/project directory should be `compilers/toyc` . The purpose of these directories/packages:

src all of your source code

tests the toyc source code files used to test your compiler

build all of your class files; created by '`ant compile`', removed by '`ant clean`'

src/globals contains all definitions needed globally throughout your compiler

src/parser most of the source code for your scanner & parser plus your drivers (e.g. `tc.java`)

src/symTable your symbol table maintenance routines

src/output screen output; code that is called to report compiler messages, including errors & warnings

src/abstractSyntax the code to construct your abstract syntax tree

Following standard practice name your ant buildfile `build.xml` . Your buildfile should have at least 3 targets: `clean`, `compile`, and `test`. Target `compile` depends on `clean` and should create and place all class files in directory `toyc/build/classes`. Issuing an '`ant clean`' should remove the entire subdirectory `toyc/build`.

Target `test` (which should be the ant default target and depend upon `compile`) executes your *parser* driver (main method) located in file `parser/tc.java`. You should define two properties to control the flags and source code file names; these properties must be named `flags` and `source`. You would define these flags from the command line in the following manner: '`ant -Dflags="-verbose" -Dsource="tests/test1.tc" test`' .

4.1.1 Compiler Interfaces

The `compilers` directory contains a couple interfaces that you must implement in your compiler.

Part #2 interface ...	**must be implemented by ...**
`compilers/Parser.java`	`parser/TCparser.java`
`compilers/AbstractSyntax.java`	*various classes – more on this later*

4.2 C++ Projects

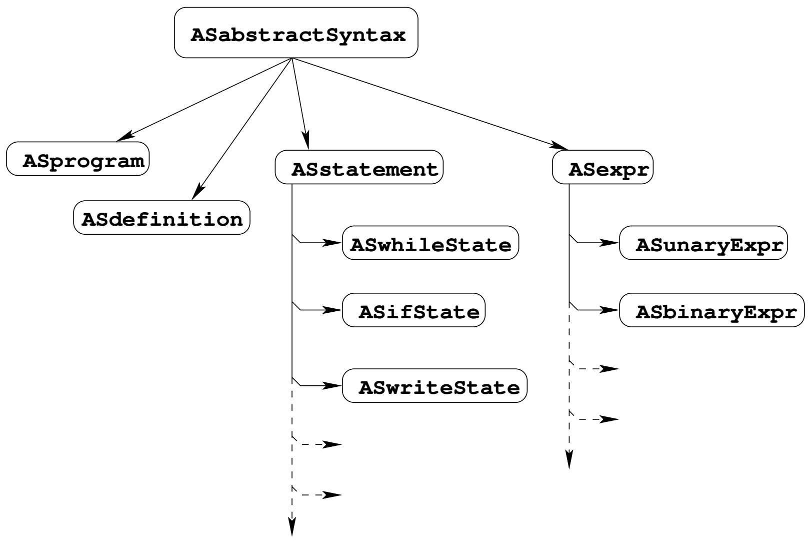
Your directory structure will remain the same as in part1 of the project. Your makefile should have the same targets, but of course it now should create your *parser* executable as `tc`.
Beacuse of the flatter structure of your C++ project (i.e. you do not organize portions of your compiler in directories/packages like in Java), it often is nice to organize sections by naming convention; some suggestions:

Section ...	Naming convention ...
scanner,parser	`TC*.h`, `TC*.cpp`
abstract syntax	`AS*.h`, `AS*.cpp`
code generation	`CG*.h`, `CG*.cpp`
JVM instructions	(in subdirectories)

You are free, however, to name your files in any manner you like. Note that this part of the project will only involve your parser and generation of an abstract syntax tree.

4.2.1 Abstract Syntax Class Hierarchies

Based on the abstract syntax given above, here are some suggestions on the class hierarchies you might define in your code:



5 Command Line Arguments

Your program must provide a certain level of control from the command-line, as specified below.

```
options:
   -help           display a usage message
   -output <file>  specifies target file name
   -class  <file>  specifies class file name
   -debug <level>  display messages that aid in tracing the
                   compilation process. If level is:
                      0 - all messages
                      1 - scanner messages only
                      2 - parser messages only
                      3 - code generation messages only
   -abstract       dump the abstract syntax tree
   -symbol         dump the symbol table(s)
   -code           dump the generated program
   -verbose        display all information
   -version        display the program version

liberty:~/compilers/toyc/>
```

All debug flags should be set to off by default. Note that for this part of the project you will not need to define options '`-output`', '`-class`', '`-debug 3`', or '`-code`'. Your scanner debug messages should report every token and lexeme returned in the following manner, one message per line:

```
[SCANNER] token STRING "hello"
```

When turned on, your parser debug messages should report the entering and exiting of all production methods, i.e.

```
[PARSER] entering WhileStatement
[PARSER] exiting WhileStatement
```

All messages must be printed as soon as possible during the recognizing of tokens and syntax (i.e. tokens are reported as soon as they are isolated by the scanner and your parser entering and exiting messages should be printed to the screen immediately upon entering or exiting a parsing method).

6 Prettyprinting

You must prettyprint your abstract syntax tree along the lines of the example given in the next section.

Here is a Java example prettyprinting class (which would be part of the `abstractSyntax` package):

```
class PrettyPrint {

static int pos = 0;
final static int INDENTSIZE=2;

public static String spaces() {
 String s = "";
 for (int i = 1; i <= pos; i++)
  s += " ";
 return s;
} // method

public static void indent() { pos += INDENTSIZE; }
public static void outdent() { pos -= INDENTSIZE; }

} // class
```

7 An Example Run

Please note that this is *not* an example run of toyc!

```
liberty:~/toy/Parser/> java Parser.toypas -thisIsNotValid
Command line error - '-thisIsNotValid' is an unrecognized command line option

Usage: java Parser.toypas [options] input_file

where options include:
   -help           display this usage message
   -output <file>  specifies target file name
   -class  <file>  specifies class file name
   -debug <level>  display messages that aid in tracing the
                   compilation process. If level is:
                      0 - all messages
                      1 - scanner messages only
                      2 - parser messages only
   -abstract       dump the abstract syntax tree
   -symbol         dump the symbol table(s)
   -code           dump the generated program
   -verbose        display all information
   -version        display the program version

Send bug reports to dresler@vcu.edu
liberty:~/toy/Parser/>
liberty:~/toy/Parser/> cat test10.toy
var a,b;

write("Enter an upper bound for loop: ");
read(a);
b = 1;
while b <= a do
begin
 write(b); write(" ");
 b = b + 1;
end;
newline;
liberty:~/toy/Parser/>
liberty:~/toy/Parser/> ls test10.j
ls: test10.j: No such file or directory
liberty:~/toy/Parser/> java Parser.toypas test10.toy
liberty:~/toy/Parser/> ls test10.j
test10.j
liberty:~/toy/Parser/>
liberty:~/toy/Parser/> java Parser.toypas -debug 0 test10.toy
[SCANNER] token VAR var
[SCANNER] token ID a
[SCANNER] token COMMA ,
[PARSER] reducing declarations
[SCANNER] token ID b
[SCANNER] token SEMICOLON ;
[PARSER] reducing declarations
[SCANNER] token WRITE write
[PARSER] reducing declarationList
[SCANNER] token LPAREN (
[SCANNER] token STRING "Enter an upper bound for loop: "
[SCANNER] token RPAREN )
[PARSER] reducing string
[PARSER] reducing expr
[SCANNER] token SEMICOLON ;
[PARSER] reducing writeStatement
[PARSER] reducing statement
[PARSER] reducing statementList
[SCANNER] token READ read
[SCANNER] token LPAREN (
[SCANNER] token ID a
[SCANNER] token RPAREN )
[PARSER] reducing identifier
[SCANNER] token SEMICOLON ;
[PARSER] reducing readStatement
[PARSER] reducing statement
[PARSER] reducing statementList
[SCANNER] token ID b
[SCANNER] token ASSIGN =
[PARSER] reducing identifier
[SCANNER] token NUMBER 1
[SCANNER] token SEMICOLON ;
[PARSER] reducing number
[PARSER] reducing expr
[PARSER] reducing assignStatement
[PARSER] reducing statement
[PARSER] reducing statementList
[SCANNER] token WHILE while
[SCANNER] token ID b
[SCANNER] token LE <=
[PARSER] reducing identifier
[PARSER] reducing expr
[SCANNER] token ID a
[SCANNER] token DO do
[PARSER] reducing identifier
[PARSER] reducing expr
[PARSER] reducing expr
[SCANNER] token BEGIN begin
[SCANNER] token WRITE write
[SCANNER] token LPAREN (
[SCANNER] token ID b
[SCANNER] token RPAREN )
[PARSER] reducing identifier
[PARSER] reducing expr
[SCANNER] token SEMICOLON ;
[PARSER] reducing writeStatement
[PARSER] reducing statement
[PARSER] reducing statementList
[SCANNER] token WRITE write
[SCANNER] token LPAREN (
[SCANNER] token STRING " "
[SCANNER] token RPAREN )
[PARSER] reducing string
[PARSER] reducing expr
[SCANNER] token SEMICOLON ;
[PARSER] reducing writeStatement
```

```
[PARSER] reducing statement
[PARSER] reducing statementList
[SCANNER] token ID b
[SCANNER] token ASSIGN =
[PARSER] reducing identifier
[SCANNER] token ID b
[SCANNER] token PLUS +
[PARSER] reducing identifier
[PARSER] reducing expr
[SCANNER] token NUMBER 1
[SCANNER] token SEMICOLON ;
[PARSER] reducing number
[PARSER] reducing expr
[PARSER] reducing expr
[PARSER] reducing assignStatement
[PARSER] reducing statement
[PARSER] reducing statementList
[SCANNER] token END end
[PARSER] reducing empty
[PARSER] reducing statement
[PARSER] reducing statementList
[SCANNER] token SEMICOLON ;
[PARSER] reducing block
[PARSER] reducing statement
[PARSER] reducing whileStatement
[PARSER] reducing statement
[PARSER] reducing statementList
[SCANNER] token NEWLINE newline
[SCANNER] token SEMICOLON ;
[PARSER] reducing newLineStatement
[PARSER] reducing statement
[PARSER] reducing statementList
[SCANNER] token EOF null
[PARSER] reducing empty
[PARSER] reducing statement
[PARSER] reducing statementList
[PARSER] reducing program
[SCANNER] token EOF null
liberty:~/toy/Parser/>
liberty:~/toy/Parser/> java Parser.toypas -abstract test10.toy

<< Abstract Syntax >>
program(
  sourceCodeFile(test10.toy),
  [
    declaration(a),
    declaration(b)
  ],
  [
    writeStatement(string("Enter an upper bound for loop: ")),
    readStatement(var(a)),
    assignStatement(
      var(b),
      int(1)
    ),
    whileStatement(
      binaryRelExpr(var(b),LE,var(a)),
      block(
        [
          writeStatement(var(b)),
          writeStatement(string(" ")),
          assignStatement(
            var(b),
            BinaryArithExpr(var(b),PLUS,int(1))
          ),
          nullStatement()
        ]
      )
    ),
    newLineStatement(),
    nullStatement()
  ]
)
liberty:~/toy/Parser/>
liberty:~/toy/Parser/> jasmin test10.j
Generated: test10.class
liberty:~/toy/Parser/> java test10
Enter an upper bound for loop: 5
1 2 3 4 5
liberty:~/toy/Parser/>
```

8 Syntax and Semantic Errors

The first syntax error encountered in the source program should be reported followed by the halting of your parser. Errors should be reported in the following manner:

```
21: if x > 3 ) write("correct answer");
       ^ '(' expected
```

(i.e. list the line containing the error preceded by the line number followed by a line pointing to and reporting the type of error encountered).

Do not detect or report any semantic errors; this will be handled in the next part of the project.

9 Deliverables

Submit Part #2 of the project via Gradescope. You should submit your *complete* `part2` directory tree structure as a zip file. This zip file should unzip into a directory named `part2` containing all of your subdirectories and your file '`part2/README.txt`'. Make sure your name(s) are prominently displayed at the beginning of *all* files. Assignments are due at the beginning of class on their due date.

Due date: Wednesday March 27

Code Report: Monday March 11