EGRE591 Compiler Project
Part 3 - Semantic Analysis & Code Generation
Spring 2024

The final phase of the project will involve semantic analysis and generating code for the Java Virtual Machine. We will not generate code for all of `toyc`; in particular, you should:

- restrict yourself to one data type: `int`
- you need not generate code for `char` literals or break statements
- you *may* implement functions/function calls for 10% extra credit; note that functions/function calls must work perfectly to receive the extra credit points
  - you need not implement functions nested deeper than one level
  - you need not implement nested blocks

You may represent boolean values as integers with 0 representing false and any non-zero value representing true.

1 Semantic Analysis

All semantic errors should be reported to the user followed by the termination of the compiler. You should check for and report violations of the following rules:

1. All identifiers must be *declared* before they are used (referenced).
2. Identifiers can have only one binding per scope (i.e. they can not be *redefined* within a given scope).
3. Subroutine parameters and/or return types must agree in number and type.
4. All expressions must be type compatible.
5. Division by zero is undefined.

Scope rules for toyc are essentially the same as with C/Java.

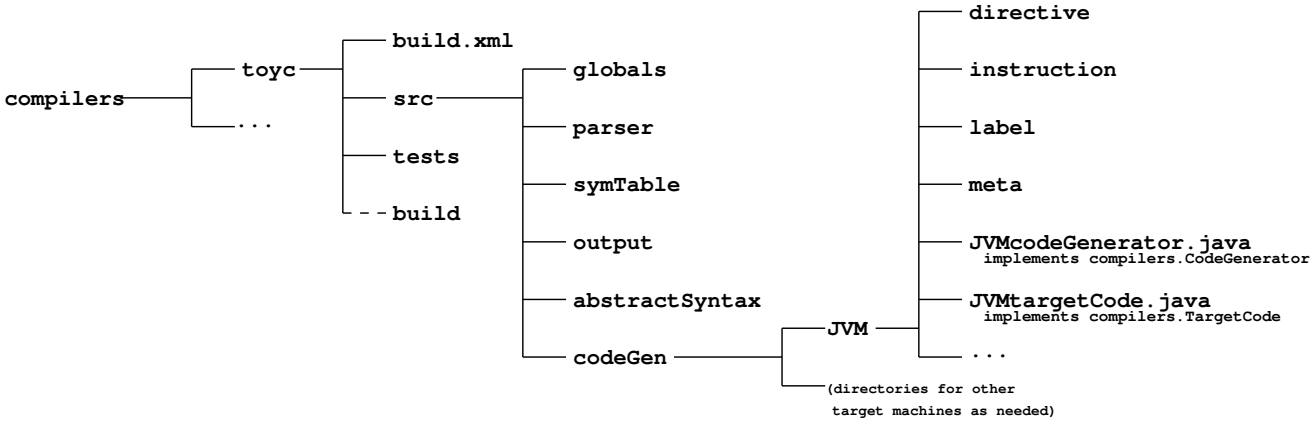
2 Symbol Table

You will need a symbol table for this portion of the assignment. The Java/C++ specifications given below are *suggestions* – you need not implement your symbol table in this manner. Note also that these specifications are not necessarily complete.

3 Program Development

3.1 Java Projects

3.1.1 Project Structure



3.1.2 SymTable.java

```
package symTable;
public class SymTable{

// Constructor
public SymTable();

// Instance Methods
public Symbol insert(String id,int type) throws SymbolAlreadyDeclared;
public Symbol insert(String id) throws SymbolAlreadyDeclared;
public Symbol find(String id) throws SymbolNotFound;
public String toString();
}
```

3.1.3 Symbol.java

```
package symTable;
public class Symbol{

// Public Instance Variables
public String name;
public int type;
public int offset;

// Constructors
public Symbol();
public Symbol(String name);
```

```
public Symbol(String name,int offset);
public Symbol(String name,int offset, int type);

// Instance Methods
public boolean equals(Symbol item);
public String toString();
}
```

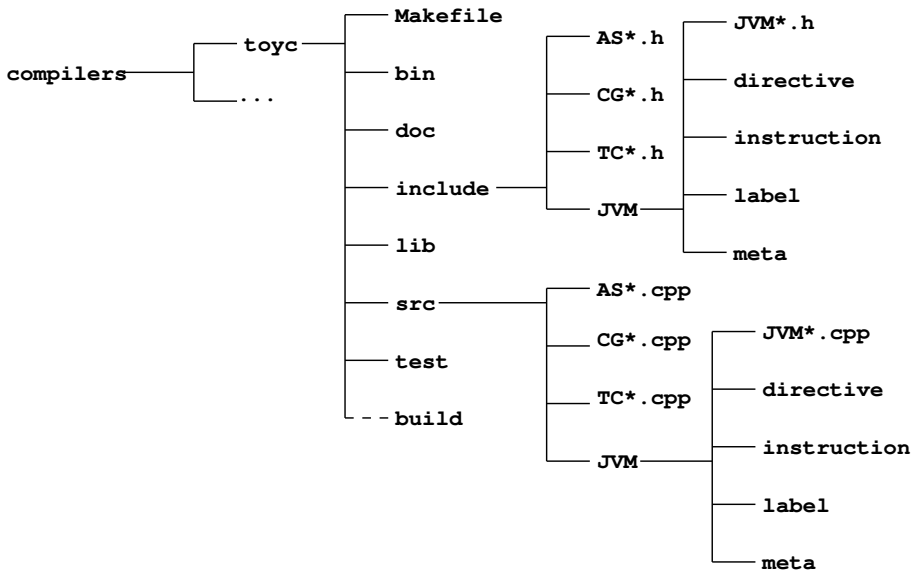
3.1.4 Code Generation

Code generation involves translating an `compilers.AbstractSyntax` object (i.e. an AST) into a `compilers.TargetCode` object.

3.2 C++ Projects

3.2.1 Project Structure

Your file directory structure should not change from the parser stage of your project.



3.2.2 `TCsymbol.h`

From the `toycalc` example compiler.

```
#include <iostream>

#ifndef TCSYMBOL_H
#define TCSYMBOL_H

namespace toycalc {

  enum symType { VAR, LABEL, OFFSET, NO_TYPE };
```

```
  class TCsymbol {
  public:
    TCsymbol();
    TCsymbol(std::string, enum symType);

    std::string getId();
    void setId(std::string);
    enum symType getType();
    void setType(enum symType);
    int getOffset();
    void setOffset(int);

    std::string toString();

  private:
    std::string id;
    int offset;
    enum symType type;

  };

}

#endif
```

3.2.3 TCsymTable.h

From the `toycalc` example compiler.

```
#ifndef TCSYMTABLE_H
#define TCSYMTABLE_H

#include "TCsymbol.h"
#include "TCtoken.h"

#define MAX_SYMS 100 // arbitrary

namespace toycalc {
  class TCsymTable {
  public:
    TCsymTable();

    int add(TCsymbol *);
    int find(std::string);
    int find(TCsymbol*);
```

```
    TCsymbol *getSym(int);
    TCsymbol *getSym(TCtoken*);
    int getSize();
    std::string toString();

  private:
    TCsymbol *symTable[MAX_SYMS];
    int size;
  };

}

#endif
```

4 Front End/Back End Implementation

Parts #1 and #2 of the assignment consisted of writing the *front-end* (concerned with the source-language) of the compiler; in this assignment you will write the *back-end* (which is concerned with the target code and machine). You must implement your code so their is a clean separation between the two parts, i.e.

```
   // front end - Parts #1, #2
   tokenBuff = getToken(); // getToken --> scanner
   AbstractTree ast = ToyCProgram();

   //--------------------------------------
   // back end - Part #3
   generateJasminCode(ast);
```

5 Command Line Arguments

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

```
liberty:~/compilers/toyc/> java -cp build/classes:../interfaces.jar parser.tc -help
Usage: java [classpath] parser.tc [options] toyc_source_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
```

```
                    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

Send bug reports to dresler@vcu.edu
liberty:~/compilers/toyc/>
```

All debug flags should be set to off by default. Note that for this part of the project you will need to add the definitions for options '`-output`', '`-class`', '`-debug 3`', and '`-code`'. All previously defined flags should continue to work as specified.

All messages must be printed as soon as possible during code generation (i.e. you should report all code that generated as soon as it is produced).

6 The Part 3 Assignment

Turn-in the following via Gradescope in the usual manner.

- Your *complete* directory tree structure as a zip file named `part3.zip` . As in Part 2, this zip file should unzip into a directory named `part3` and should contain all of your sub-directories. 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.
- As before when unzipped the text file `part3/README.txt` should explain how to build, clean, and run your project.
- Your compiler should by default produce Jasmin files with the same name as your `toyc` source code file, e.g. if your `toyc` program is contained in file `test1.tc`, then you should by default generate Jasmin file '`test1.j`'. The command line directive `output` can overrule this default.
- Partial credit will be given should you not finish generating code for all constructs in the source language. If you do not complete the entire language, *you must explicitly indicate at the top of your* `README.txt` *file which sections you did not finish.* In addition if you completed the extra credit for the project you must also specify this at the beginning of `README.txt` .

Due date: Wednesday, April 24^{th}

Code Report: Monday, April 15^{th}