

# National University of Singapore

## CS4212 Project Assignment 2

AY 2018/2019 Semester 1

Due Date: 19<sup>th</sup> October 2018 (Friday, 23:59 Hrs)

This assignment aims to complete the construction of the front end of a JLiTE compiler.

You are given the Java code for constructing the parse trees for JLiTE programs. Your tasks are therefore:

1. To perform semantics processing to ensure that the JLiTE programs are semantically sound.
2. To perform intermediate code generation for JLiTE programs.

We encourage you to use your own parser and lexer that you have done in Assignment 1. Contact the TA Mr Nguyễn Thanh Toàn if you have any problem. Until now, just **one** student has some difficulties.

## 1 Resources Provided

1. A parser that takes in JLiTE programs and transforms them into parse trees. This is given in the form of CUP and JFlex codes, available in the provided package.
2. A typing rules specification the type soundness of JLiTE language (given in Appendix A.)
3. A specification for the IR3 language (given in Appendix B.)
4. Some sample JLiTE code. (One such JLiTE code together with its corresponding IR3 code is given in Appendix C, the rest are given in the package.)

## 2 Your Tasks

Given a JLiTE program as input, your developed system is required to generate two pieces of information:

1. **Static Checker:** Your code will perform static semantics processing of JLiTE programs.
2. **Intermediate Code Generator:** Your code will generate IR3-code for those JLiTE programs which successfully pass your *static checker* tests.
3. **Submission:** You are to submit your entire code in a self-contained ZIP in IVLE. The ZIP file should be such that after unzipping, one can compile your code with a single make command - make *no assumption* about what special packages are already installed on the user's machine other than the usual make and GCC. Note that the deadline will not be extended unless you have special reasons.

## 2.1 Static Checker

The *Static checker* should assume that JLite language follows **static scope** discipline, and performs the following static checks on JLite programs:

1. **Distinct-Name Checking:** This ensures that:

- (a) No two fields in a class can have the same name.
- (b) No two classes can be declared in a program with the same (class) name.
- (c) No two parameters in a method declaration can have the same name.
- (d) No two methods within a class declaration can have the same name.

*You have the option for allowing **overloaded methods** to occur in a class declaration. You will however need to accompany your code with words describing how you handle overloaded methods. There is a **bonus of 5 marks** for correct handling of overloaded method declaration, and clear description of how you handle it.*

2. **Type checking:** This makes sure that uses of program constructs conform to how the constructs have been declared. The rules defining the soundness of type checking is described by a set of typing rules, which is presented in Appendix A.

For submitting your code for assessment, your static checker needs to:

- 1. produce syntax trees – for JLite programs that pass your checker – for use by the intermediate code generator;
- 2. report, as clearly as possible, any compilation errors arisen due to semantic errors in the JLite programs. (There is no need for you to modify the existing parser to handle syntactic errors.)
- 3. produce necessary type information required for use by intermediate code generation as well as the backend of the compiler.

Relating to the last point above, you can produce this piece of information as a separate data structure, or you attach this information to the parse tree.

## 2.2 IR Generator

The second program you are going to construct, called “IR3-Gen”, takes in a data structure (aka., parse tree) that has passed the static checks, and that contains type information. IR3-Gen aims to generate three-address code of the form called IR3.

We normally think of the code generated to be a string of characters. However, in this project, your IR3-Gen does not directly generate the character string representing the intermediate code. Rather, it generates a data structure as specified by the *IR3* grammar, as shown in Appendix B. As a data structure, we can further take it as input and transform it to some low-level machine code (as will be done in Assignment 3). Certainly, generating code from the IR3 data will not be difficult – you just need to use the affiliated set of display functions; or you might want to alter these display functions so that they can generate fascinating displays of IR3 codes.

### 3 Deliverable

#### Testing Your Programs

We have provided some sample JLITE programs for you to do your preliminary testing. Nevertheless, you are required to create some sample programs to test your product. Put into a subfolder these sample programs that you have created, and show their corresponding results.

Create a readme file to describe how one can make use of your test suites to validate the correctness of your programs.

Appendix C provides a sample JLITE program and the corresponding IR3 code.

#### Submission of your product

Please submit a zipped item, named after you, containing the following documents to IVLE CS4212 website under the “Project Assignment 2 Submission” folder.

1. Your complete project and an instruction on how to run it. It is encouraged to use a Makefile as in Assignment 1.
2. Five (or more) sample programs that you have tried on your product
3. A document, named **Project Assignment 2 Readme**, describing your product, the content of your submission, and any important information which you would like to share with us, and which you think might gain you more marks.

## A Typing Rules for JLITE Programs

### A.1 Class Descriptor, Type Environment and Legitimate Types

The *type environment* comprises two entities: *class descriptor*  $\mathcal{C}$  and *local environment*  $\Gamma$ .

1. *Class Descriptor*  $\mathcal{C}$  maps a class name to a triple consisting of: its class name, the types of each of its field declarations, and the signatures of each of its method declarations. Note that, in case the class declared is **not** inherited from any other class, the type of its parent attribute will be  $\top$ .

$$\begin{aligned} \mathcal{C} &:: \langle cname \rangle \rightarrow (\langle fds \rangle \times \langle msigs \rangle) \\ \langle fds \rangle &= \langle id \rangle \rightarrow T \\ \langle msigs \rangle &= \langle id \rangle \rightarrow ([T] \times T) \end{aligned}$$

2. *Local environment*  $\Gamma$  maps variables declared locally to their types.

$$\Gamma \quad :: \quad \langle id \rangle \rightarrow T$$

The type class  $T$  is declared as follows:

$$T = \langle cname \rangle \cup \{int, bool, void\}.$$

where  $\langle cname \rangle$  ranges over all class names declared.

We begin type checking of a JLITE program  $P$  by filling in the class descriptor with all type information declared in the program; we call this process *initialize*( $P$ ). This includes class names, class attribute declarations and method signatures occurred in each class. On the other hand, the initial local environment contains empty information (or it may contain some global type information, such as the types of the primitive operations).

In addition, we assume that all types *referred to* in the program must be *legitimate*; That is, they can be found in the type class  $T$ ; failing so will lead to type errors during compilation time.

## A.2 Type Checking A JLITE Program

A JLITE program is well typed if the type checker returns `isOk` upon return.

$$\begin{array}{c} P = \text{mainC } C_1 \dots C_n \quad \Gamma = [] \\ \mathcal{C} = \text{initialize}(P) \quad \langle \mathcal{C}, \Gamma \rangle \vdash \text{mainC } \text{isOk} \\ \frac{\langle \mathcal{C}, \Gamma \rangle \vdash C_i \text{ isOK} \quad \forall i \in \{1, \dots, n\}}{\vdash \text{mainC } C_1 \dots C_n \text{ isOK}} \quad [\text{JLITE-Pgm}] \end{array}$$

## A.3 Type Checking Classes

Before type checking all the methods declared in a class, the local environment is set to include respectively the types/signatures associated with the attributes/methods declared in the class, and the types for the special identifiers *this*.

$$\begin{array}{c} ((a_1 \mapsto t_1, \dots, a_n \mapsto t_n), (ms_1, \dots, ms_k)) = \mathcal{C}(c) \\ ms_i = (m_i \mapsto ([t_{i1}; \dots; t_{ip}], tr_i)) \quad \forall i \in \{1, \dots, k\} \\ \Gamma = [this \mapsto c, a_n \mapsto t_n, \dots, a_1 \mapsto t_1, ms_k, \dots, ms_1] \\ \frac{\langle \mathcal{C}, \Gamma \rangle \vdash md_i \text{ isOK} \quad \forall i \in \{1, \dots, k\}}{\langle \mathcal{C}, \_ \rangle \vdash \text{class } c \{t_1 a_1; \dots; t_n a_n; md_1; \dots, md_k\} \text{ isOK}} \quad [\text{CDecl}] \end{array}$$

## A.4 Environment Operations

We define here several notations describing the operations on a local environment,  $\Gamma$ .

- *Environment Lookup*: Given  $\Gamma = [(v_1, t_1); \dots; (v_n, t_n)]$ , looking up the type of a variable/attribute  $x$  from  $\Gamma$  is defined by:

$$\begin{aligned} \Gamma(x) &= \text{match } \Gamma \text{ with} \\ &\quad | ((v, t) :: \Gamma') \rightarrow \begin{array}{l} \text{if } (v = x) \text{ then Some } t \\ \text{else } \Gamma'(x) \end{array} \\ &\quad | [] \rightarrow \text{None} \end{aligned}$$

- *Environment Augmentation*: When we wish to add more associate pairs into the existing environment  $\Gamma$ , we always add them to the front of the association list. Given  $\Gamma = [(v_1, t_1); \dots; (v_n, t_n)]$ , we have

$$\begin{aligned} \Gamma[] &= \Gamma \\ \Gamma[v'_1 \mapsto t'_1] &= \Gamma' \text{ where } \Gamma' = [(v'_1, t'_1); (v_1, t_1); \dots; (v_n, t_n)] \\ \Gamma[v'_1 \mapsto t'_1, v'_2 \mapsto t'_2, \dots, v'_m \mapsto t'_m] &= (((\Gamma[v'_1 \mapsto t'_1])[v'_2 \mapsto t'_2]) \dots)[v'_m \mapsto t'_m] \end{aligned}$$

## A.5 Type Checking Method Declaration

When type checking a method declaration, the local environment is augmented with the types of the parameters declared in the method, **and the return type** of the method, associated with and unique special identifier *Ret*.

$$\begin{array}{c} \text{Some } ([t_1; \dots; t_n], t_0) = \Gamma(m) \\ \langle \mathcal{C}, \Gamma[v_1 \mapsto t_1, \dots, v_n \mapsto t_n, \text{Ret} \mapsto t_0] \rangle \vdash_S S : t \\ \hline \langle \mathcal{C}, \Gamma \rangle \vdash t_0 m(t_1 v_1, \dots, t_n v_n) S \text{ isOK} \end{array} \quad [\text{MDecl}]$$

## A.6 Type Checking Statements

Type checking of statements is specified by the judgment:  $\langle \mathcal{C}, \Gamma \rangle \vdash_S S : t$ .

$$\frac{\langle \mathcal{C}, \Gamma[v_1 \mapsto t_1, \dots, v_n \mapsto t_n] \rangle \vdash_S S : t}{\langle \mathcal{C}, \Gamma \rangle \vdash_S \{ t_1 v_1; \dots; t_n v_n; S \} : t} \quad [\text{Block}]$$

$$\frac{\langle \mathcal{C}, \Gamma \rangle \vdash_S S_1 : t_1 \quad \langle \mathcal{C}, \Gamma \rangle \vdash_S S_2 : t_2}{\langle \mathcal{C}, \Gamma \rangle \vdash_S S_1; S_2 : t_2} \quad [\text{Seq}]$$

$$\frac{\langle \mathcal{C}, \Gamma \rangle \vdash_E e : t_e \quad \Gamma(v) = \text{Some } t_v \quad t_e = t_v}{\langle \mathcal{C}, \Gamma \rangle \vdash_S v = e : \text{void}} \quad [\text{VarAss}]$$

$$\frac{\langle \mathcal{C}, \Gamma \rangle \vdash_E e_1 : c_1 \quad \langle \mathcal{C}, \Gamma \rangle \vdash_E e_2 : t_2 \quad \langle fds, \_ \rangle = \mathcal{C}(c_1) \quad \text{Some } t_a = fds(a) \quad t_a = t_2}{\langle \mathcal{C}, \Gamma \rangle \vdash_S e_1.a = e_2 : void} \quad [\text{FdAss}]$$

$$\frac{\langle \mathcal{C}, \Gamma \rangle \vdash_S S_1 : t_1 \quad \langle \mathcal{C}, \Gamma \rangle \vdash_S S_2 : t_2 \quad \langle \mathcal{C}, \Gamma \rangle \vdash_E e : bool \quad t_1 = t_2}{\langle \mathcal{C}, \Gamma \rangle \vdash_S \text{if } e \text{ then } S_1 \text{ else } S_2 : t_2} \quad [\text{Cond}]$$

$$\frac{\langle \mathcal{C}, \Gamma \rangle \vdash_E e : bool \quad \langle \mathcal{C}, \Gamma \rangle \vdash_S S : t}{\langle \mathcal{C}, \Gamma \rangle \vdash_S \text{while } e \{ S \} : t} \quad [\text{While}]$$

$$\frac{\Gamma(v) = \text{Some } t \quad t \in \{int, bool, string\}}{\langle \mathcal{C}, \Gamma \rangle \vdash_S \text{readln } v : void} \quad [\text{Read}]$$

$$\frac{\langle \mathcal{C}, \Gamma \rangle \vdash_E e : t \quad t \in \{int, bool, string\}}{\langle \mathcal{C}, \Gamma \rangle \vdash_S \text{println } e : void} \quad [\text{Print}]$$

$$\frac{\langle \mathcal{C}, \Gamma \rangle \vdash_E e_i : t'_i \quad \forall i \in \{1, \dots, p\} \quad \text{Some } ([t_1; \dots; t_p], t_r) = \langle \mathcal{C}, \Gamma \rangle(m) \quad \frac{t'_i = t_i \quad \forall i \in \{1, \dots, p\}}{\langle \mathcal{C}, \Gamma \rangle \vdash_S m(e_1, \dots, e_p) : t_r}}{\langle \mathcal{C}, \Gamma \rangle \vdash_S m(e_1, \dots, e_p) : t_r} \quad [\text{SLocalCall}]$$

$$\frac{\langle \mathcal{C}, \Gamma \rangle \vdash_E e_i : t'_i \quad \forall i \in \{1, \dots, p\} \quad \langle \mathcal{C}, \Gamma \rangle \vdash_E e_0 : c_0 \quad \langle \_, \_, \Gamma_{MS} \rangle = \mathcal{C}(c_0) \quad \text{Some } ([t_1; \dots; t_p], t_r) = \Gamma_{MS}(m) \quad \frac{t'_i = t_i \quad \forall i \in \{1, \dots, p\}}{\langle \mathcal{C}, \Gamma \rangle \vdash_S e_0.m(e_1, \dots, e_p) : t_r}}{\langle \mathcal{C}, \Gamma \rangle \vdash_S e_0.m(e_1, \dots, e_p) : t_r} \quad [\text{SGlobalCall}]$$

$$\frac{\langle \mathcal{C}, \Gamma \rangle \vdash_E e : t \quad t' = \Gamma(Ret) \quad t = t'}{\langle \mathcal{C}, \Gamma \rangle \vdash_S \text{return } e : t'} \quad [\text{Ret-T}]$$

$$\frac{\Gamma(Ret) = void}{\langle \mathcal{C}, \Gamma \rangle \vdash_S \text{return} : void} \quad [\text{Ret-Void}]$$

## A.7 Type Checking Expressions

Type checking of expressions is specified by the judgment:  $\langle \mathcal{C}, \Gamma \rangle \vdash_E e : t$ .

$$\frac{v \in \langle Id \rangle \cup \{\mathbf{this}\} \quad \Gamma(v) = \mathbf{Some } t}{\langle \mathcal{C}, \Gamma \rangle \vdash_E v : t} \quad [\text{Id}]$$

$$\frac{}{\langle \mathcal{C}, \Gamma \rangle \vdash_E 1 : int} \quad [\text{Integers}]$$

$$\frac{}{\langle \mathcal{C}, \Gamma \rangle \vdash_E \mathbf{true} : bool} \quad [\text{Booleans}]$$

$$\frac{\langle \mathcal{C}, \Gamma \rangle \vdash_E e : c \quad \langle fds, \_ \rangle = \mathcal{C}(c) \quad \mathbf{Some } t_a = fds(a)}{\langle \mathcal{C}, \Gamma \rangle \vdash_E e.a : t_a} \quad [\text{Field}]$$

$$\frac{\langle fds, ms \rangle \in \mathcal{C}(c)}{\langle \mathcal{C}, \Gamma \rangle \vdash_E \mathbf{new } c() : c} \quad [\text{New}]$$

$$\frac{\begin{array}{l} \langle \mathcal{C}, \Gamma \rangle \vdash_E e_i : t'_i \quad \forall i \in \{1, \dots, p\} \\ \mathbf{Some } ([t_1; \dots; t_p], t_r) = \Gamma(m) \\ t'_i = t_i \quad \forall i \in \{1, \dots, p\} \end{array}}{\langle \mathcal{C}, \Gamma \rangle \vdash_E m(e_1, \dots, e_p) : t_r} \quad [\text{LocalCall}]$$

$$\frac{\begin{array}{l} \langle \mathcal{C}, \Gamma \rangle \vdash_E e_i : t'_i \quad \forall i \in \{1, \dots, p\} \\ \langle \mathcal{C}, \Gamma \rangle \vdash_E e_0 : c_0 \quad \langle \_, mds \rangle = \mathcal{C}(c_0) \\ \mathbf{Some } ([t_1; \dots; t_p], t_r) = mds(m) \\ t'_i = t_i \quad \forall i \in \{1, \dots, p\} \end{array}}{\langle \mathcal{C}, \Gamma \rangle \vdash_S e_0.m(e_1, \dots, e_p) : t_r} \quad [\text{GlobalCall}]$$

$$\frac{\langle \mathcal{C}, \Gamma \rangle \vdash_E e_1 : int \quad \langle \mathcal{C}, \Gamma \rangle \vdash_E e_2 : int \quad aop \in \{+, -, *, /\}}{\langle \mathcal{C}, \Gamma \rangle \vdash_E e_1 aop e_2 : int} \quad [\text{Arith}]$$

$$\frac{\langle \mathcal{C}, \Gamma \rangle \vdash_E e_1 : int \quad \langle \mathcal{C}, \Gamma \rangle \vdash_E e_2 : int \quad rop \in \{<, >, <=, >=, ==, !=\}}{\langle \mathcal{C}, \Gamma \rangle \vdash_E e_1 rop e_2 : bool} \quad [\text{Rel}]$$

$$\frac{\langle \mathcal{C}, \Gamma \rangle \vdash_E e_1 : bool \quad \langle \mathcal{C}, \Gamma \rangle \vdash_E e_2 : bool \quad bop \in \{||, \&\&\}}{\langle \mathcal{C}, \Gamma \rangle \vdash_E e_1 bop e_2 : bool} \quad [\text{Bool}]$$

## B Syntactic Specification of IR3

### B.1 Lexical Issues

- $\langle CName3 \rangle \in$  **Class Names**. It is defined similarly as  $\langle CName \rangle$  in JLite.
- $\langle id3 \rangle \in$  **Identifiers**. This includes  $\langle id \rangle$  as defined in JLite. In addition, it also includes **this** (referring to specific object), as well as newly generated temporary variables, which are of the format  $\_ [a-z] + [0-9] +$ .

### B.2 Syntactic Specification

$\langle Program3 \rangle \rightarrow \langle CData3 \rangle^* \langle CMtd3 \rangle^*$   
 $\langle CData3 \rangle \rightarrow \text{Data3 } \langle CName3 \rangle \{ \langle VarDecl3 \rangle^* \}$   
 $\langle VarDecl3 \rangle \rightarrow \langle Type3 \rangle \langle id3 \rangle ;$   
 $\langle CMtd3 \rangle \rightarrow \langle Type3 \rangle \langle id3 \rangle ( \langle FmlList3 \rangle ) \langle MdBdy3 \rangle$   
 $\langle FmlList3 \rangle \rightarrow \langle CName3 \rangle \text{ this } \langle FmlL13 \rangle$   
 $\langle FmlL13 \rangle \rightarrow , \langle Type3 \rangle \langle id3 \rangle \langle FmlRest3 \rangle^* \mid \epsilon$   
 $\langle FmlRest3 \rangle \rightarrow , \langle Type3 \rangle \langle id3 \rangle$   
 $\langle Type3 \rangle \rightarrow \text{Int} \mid \text{Bool} \mid \text{String} \mid \text{Void} \mid \langle CName3 \rangle$   
 $\langle MdBdy3 \rangle \rightarrow \{ \langle VarDecl3 \rangle^* \langle Stmt3 \rangle^+ \}$   
 $\langle Stmt3 \rangle \rightarrow \langle Label3 \rangle : \mid \text{if } ( \langle RelExp3 \rangle ) \text{ goto } \langle Label3 \rangle ; \mid \text{goto } \langle Label3 \rangle ;$   
 $\mid \text{readln } ( \langle id3 \rangle ) ; \mid \text{println } ( \langle idc3 \rangle ) ;$   
 $\mid \langle Type3 \rangle \langle id3 \rangle = \langle Exp3 \rangle ; \mid \langle id3 \rangle = \langle Exp3 \rangle ; \mid \langle id3 \rangle . \langle id3 \rangle = \langle Exp3 \rangle ;$   
 $\mid \langle id3 \rangle ( \langle VList3 \rangle ) ;$   
 $\mid \text{return } \langle id3 \rangle ; \mid \text{return} ;$   
 $\langle RelExp3 \rangle \rightarrow \langle idc3 \rangle \langle Relop3 \rangle \langle idc3 \rangle \mid \langle idc3 \rangle$   
 $\langle Exp3 \rangle \rightarrow \langle idc3 \rangle \langle Bop3 \rangle \langle idc3 \rangle \mid \langle Uop3 \rangle \langle idc3 \rangle \mid \langle id3 \rangle . \langle id3 \rangle \mid \langle idc3 \rangle$   
 $\mid \langle id3 \rangle ( \langle VList3 \rangle ) \mid \text{new } \langle CName3 \rangle ( )$   
 $\langle Relop3 \rangle \rightarrow < \mid > \mid <= \mid >= \mid == \mid !=$   
 $\langle Bop3 \rangle \rightarrow \langle Relop3 \rangle \mid || \mid \&\& \mid * \mid / \mid + \mid -$   
 $\langle Uop3 \rangle \rightarrow ! \mid -$   
 $\langle VList3 \rangle \rightarrow \langle idc3 \rangle \langle VRest3 \rangle^* \mid \epsilon$   
 $\langle VRest3 \rangle \rightarrow , \langle idc3 \rangle$   
 $\langle idc3 \rangle \rightarrow \langle id3 \rangle \mid \langle Const \rangle$   
 $\langle Const \rangle \rightarrow \text{true} \mid \text{false} \mid \text{INTEGER\_LITERAL} \mid \text{STRING\_LITERAL} \mid \text{NULL}$



## C Sample JLITE Program and Corresponding IR3 Code Generated

Following is a sample JLITE program:

```
class MainC {

    Void main () {
        Functional fo ;
        Int i;
        Int j ;

        readln(i) ;
        if (i > 0) {
            fo = new Functional() ;
            j = fo.f(i) ;
            println(j) ;
        }
        else {
            println("Error") ;
        }
        return ;
    }
}

class Functional {
    Int a;

    Int f (Int b){
        return 3;
    }
}
```

Following is a possible IR3 code generated for the above-mentioned JLiTE program.

```
===== CData3 =====

class MainC{
}

class Functional{
    Int a;
}

===== CMtd3 =====

void main(MainC this){
    Functional fo;
    Int i;
    Int j;
    readln(i);
    If(i > 0) goto 1;
    println("Error");
    goto 2;
Label 1:
    fo = new Functional();
    j = %Functional_0(fo,i);
    println(j);
Label 2:
    Return;
}

Int %Functional_0(Functional this,Int b){
    Int _t1;
    _t1=3;
    Return _t1;
}

=====fx== End of IR3 Program =====
```