**תרגיל 4**

**1. מימוש בונוסים**

בתרגיל מימשנו את בונוסים 1 + 2 באופן מלא.

**2. מבנה הקוד**

בתרגיל זה הוספנו את החבילות, IC.lir, IC.lirAST.

**2א. IC.lir.lirAST** – תת חבילה של IC.lir, מבטאת את מבנה התוכנית בשפת LIR (אובייקטים של ה-AST).

היררכיה מרכזית:

**LirNode**: base class, represents a lir node in the ast

- **ArrayLengthNode** : represents an array length node lir instruction [ ArrayLength ]

- **BinaryInstructionNode**: represents a binary instruction [ Add,Sub, And, … ]

- **CompareNode**: represents a lir compare instruction

- **DispatchTableNode**: represents a dispatch table node (DV pointer, followed by method labels)

- Immediate: represents a lir immediate (32 bit signed value), used to represent integers, null type(0) and Boolean type (0 for false, 1 for true)

- **JumpNode**: represents a lir jump instruction (unconditional jump)

Sub classes: **JumpFalse**, **JumpTrueNode**, **JumpG**, **jumpGE**, **jumpL**, **jumpLE**

-**LabelNode**: represents a node in the AST that holds a lir label only

-**MethodCallNode**: represents an abstract lir call instruction

Sub classes:

* **LibraryCallNode** : represents a call to a library method
* **StaticCallNode**: represents a static call to a user defined method
* **VirtualCallNode**: represents a virtual call

-Memory: represents a variable that is stored in memory [Parameter, Local var]

Sub classes:

* **Label**: string name are considered memory locations in Lir
* **ThisNode**: represents the current object [“this” lir keyword]

-**LirProgram**: represents a lir program, contains methods, exit and entry labels, collection lir method objects, collection of dispatch table lir nodes and a collection of string literal definitions.

-**MethodNode**: represents a lir method construct. Contains the method label, set of lir instructions that define the method

- **MoveArrayNode**: represents a MoveArray lir instruction

Sub classes:

* **LoadArrayNode** : load from array instruction
* **StoreArrayNode**: store into array instruction

-**MoveFieldNode**: represents a MoveField lir instruction

Sub classes:

* **LoadField**: load from a field instruction
* **StoreField**: store into a field instruction

-**MoveNode**: represents a Move instruction, as defined in the spec file

-**Reg**: represents a lir register, each register is defined by its index. For example, register

Number 5 will be represented as “R5”.

-**RegWithIndex**: represents the pair Register[Index] used in move array instructions

-**RegWithOffset**: represents the pair Register.offset used in move field instructions

-**ReturnNode**: represents the lir return instruction

-**StringLiteralNode**: represents a string literal definition line in the lir program (the pair, string name label and the actual string literal, as appears in the top of the program)

- **UnaryInstructionNode**: represents an unary lir instructions [Inc, Dec, … ]

מחלקות נוספות בחבילה זו:

**lirBinaryOp** – [enum] represents the possible lir binary operations

**lirUnaryOp** – [enum] represents the possible lir unary operations

**LirVisitor** – a visitor for the lir ast, will be used in next assignment.

כמו כן, חשוב לציין כי כל אובייקט LirNode מגדיר מתודה **emit(**) שמייצרת את הקוד הLIR כמחרוזת.

**2ב. IC.lir** – חבילה מטפלת בכל מה שקשור לשפת LIR, בייצוג המחלקות, ביצירת labels ושמות משתנים, בייצוג dispatch tables ולבסוף בתרגום ה-AST לAST מתאים בשפת LIR.

מחלקות עיקריות:

1. **ClassLayout**:

Represents the layout of an IC class: maps from methods and fields to their offsets inside the class object. This class also provides methods to generate the dynamic dispatch table for the class.

2. **ClassLayoutManager**:

Provides a convenient interface for the lir translator to access class information. The class gathers information about all the classes in the program, builds their dispatch tables, provides quick methods to fetch method/field offsets and class’ sizes.

3. **DispatchTableEntry**:

Represents a single entry in the dynamic dispatch table. Holds a method name, name of the class that defines it and the offset of the method inside the dispatch table.

4. **LabelGenerator**:

Generates labels for string literals, methods, class DV pointers, library methods, static user defined methods, custom run time check methods, virtual methods, and custom labels for other purposes (like control, if, while, etc). All labels are unique and created only once (singletons)

5. **VariableNameGenerator**

Allocates variable names for a method. All names are unique and defined only once (singleton). The class also handles duplicate names.

6. **RuntimeChecks**

This class provides convenient methods to fetch method call objects to runtime checks and also provides their implementation to inject to the main program.

7. **LirTranslator**

Visitor class that traverses the given IC AST and constructs a LIR ast.

**3. אופן הפעולה:**

המחלקה המרכזית בפרויקט, IC.Compiler, מפעילה את המבקר המרכזי, IC.lir.LirTranslator, לצורך בניית ה-AST של תוכנית ה-LIR המתקבלת. לאחר-מכן באמצעות מתודת emit (שמוגדרת בכל LirNode) אנו מייצרים את התוכנית ה-LIR.

**4. טסטים:**

**Test1.txt**

מכיל בדיקה של class יחיד. בדיקת פונקציות סטטיות ווירטואליות. בדיקת if else, while.. בנוסף מכיל בדיקת מערך, יצריה וכתיבה והוצאת מידע.

**Test2.txt**

מכיל בדיקה של class יחיד. בדיקת פונקציות וירטואליות וגישה לשדות באמצעות הפונקציות ובאמצעות גישה ישירה. שימוש בפונקציות ספרייה.

**Test3.txt**

מכיל בדיקה של שני קלאסים. בודק ירושות. Overloading and new function in the extended class

**Test4.txt**

מכיל בדיקה של break.

**Test5.txt**

מכיל בדיקה של אופרנדים && ||

**Semantics.txt 1 – 6**

מכילים בדיקות של כל השגיאות הסמנטיות המופיעות בקובץ ה spec:

