Compiler Design

Viktorya Harutyunyan

Professor Joey Paquet

ASSIGNMENT #4

Contents

1. Semantic specifications

Note: In this section are stated only the semantic specifications that are not included in the assignment documentation (#3 and #4).

1.1. operation ::= (operator VariableI_int) * | EPS

operation::= (operator Variable_float)* | EPS

operator ::= [+|-|*|/|or|and|not|=|==|<|>|<>|<=|>=]

Variable_int ::= variabletypeinteger

Variable_float ::= variabletypefloat

Explanation=>

a. operation can involve only simple types <int> and <float>. No operation is defined for arrays and user defines types (classes).

b. There is no type conversion between <int> and <float>. Any mixture of types brings up a semantic error.

1.2. FunctionParameterType::= int | float | array | class

Explanation=> any type could be passed to a function as a parameter including <class> , <array> and array of objects

1.3. FunctionReturnType::= int | float

Explanation=> function can return only integer and float values. <returne()> is not defined for complex structures.

1.4. FunctionParameterList ::= (FunctionParameter)*| EPS

FunctionParameter::= variabletypeinteger | variabletypefloat | class | array

Explanation=> theoretically function can contain any number of parameters, but we have to take into account how many functions are called, what are the size of

parameters. For example if we pass array of objects say A[10][10][10], where <A> is of a size 100bayts => the entire array takes 1000 * 100 byte which is being too much compare to the memory limits that moon machine provides.

1.5. variable ::= Variable_int | Variable_float | class | array_int | array_float | array_object

Explanation=> there is no limitation on variables, including arrays of objects.

1.6. Relational operators are used only with <if> statements

1.7. Logical <and, or, not> are used within expressions equally with the <+, -, *,/ > within expressions

1.8. Statements put(exp) and get(exp) are defined only for simple types => no arrays, no objects

1.9 Recursion calls are allowed by the underlying language.

2. Errors

The error messages rated to the semantic checking's are already included in the document <Assignment#3>. In this section are included new errors, that were ignored for <Assignment#3>. Note that error < SEMERR#17:> is removed from error list; according to the implementation of <Assignment#4>, arrays are valid parameters for functions.

Error code	Error text	Error description
SEMERR#17:	Invalid Function Parameter	Removed!
SEMERR#18:	Invalid Function call	Is triggered when function has Invalid number of parameters, or function is called without parentheses, or function is called without parentheses
SEMERR#19:	Invalid operation	Is triggered when a current operation (assignment, addition ...) is not defined for arrays
SEMERR#20:	Invalid operation	Is triggered when <Put> or <get> are used with non simple type. Also, it could be triggered when complete array or object are use in an expression.

3. File System

File output for a current running file is "files\Output".

1. "files\Output _test.m" => is the generated moon code for a current given file
2. "\files\Output\SYNTAX_ANALYZER_SYMBOL_TABLE.txt " => is the symbol table content for a current given file.
3. "\files\Output\SEMANTIC_ANALYZER_ERRORS.txt " => is the output semantic error file for a current given file

Note: the rest of all the files needed for compiler processing is described in previews assignment documentations.

4. Testing

1. <CodeGenerationPart_correctTestCases> => contains test cases to check correctness. Test cases output a moon code that is executable on moon machine. For each test case there are 3 fines with the same prefix. Assume the name of the file is <testCase.txt> , then <testCase_MOON.m> represents the assembly code of that test case and < testCase_SymbolTable> contains information of symbol table.

   To run a test case we have to copy a full path of that test case into a console window when it outputs <Enter file name to compile>

   Note: the names of test cases are self descriptive, each of them has also a description inside of a test case.
   note: some test cases require an input from keyboard to start computation

2. < ErrorCheckingTestCases> => contains test cases to check errors. Mostly this set is related to the syntax specifications stated in this document. Note that most of the type checking part is done in the <Assignment#3>. For each test case there are 2 files. Assume the name of the test case is <testCase.txt>, then <testCase_SymbolTabe.txt> contains its symbol table. Also, at the end of each <testCase.txt> error messages are provided for that specific test case (in the multiline comments (/*...*/) . Test cases in this section generate moon code, but the code is not executable.