COMP6421 Assignment 4

Current Version: 0.1
Date: Apr. 8, 2012
Author: Yuan Tao
Instructor: Dr. Joey Paquet

Concordia University

Table of Contents

1. Semantics of the language

1.1. Grammar

No change to the grammar defined in the assignment 2.

1.2. Data types

- integer
  Range: -2147483647 ~ 2147483647
- real
  Range: -2147483.647 ~ 2147483.647
  The scale of decimal is 3.
- class
  The data type of its member variable can be other class.

1.3. Data type conversion

The conversion between integer and real are performed automatically by the complier. A warning message will be issued as well. The conversion happens for the following cases:

- Expressions
- Assignment statement
- Passing parameters
- Function return

1.4. Array

- There is no limitation for the size of each dimension or the number of dimensions.
- The type of the array element can be integer, real and class.
- Array is not a data type. It is a group of variables, which have same data type, store consecutively in somewhere of the memory.
- When using an array variable, the number of dimensions should be the same as the number when it is defined. Otherwise, an error message will be issued.

1.5. Function

- entry point of execution
  The “program” function.
- parameters
  - number
    No limitation. It is based on the size of the memory allocated by Moon simulator.
  - type
    integer, real, class
  - the way of passing parameters
    If the type of parameter is integer or real, it passed by value.
    If the type of parameter is class, it passed by reference, which means the modification of the member variables of the class within the function will take effect outside function.
- return type
  integer or real.
  class is not supported.
- Recursive call
  Any free function or class member function can call itself.
- Function overloading
  Functions with same name but different number of parameters or different type of parameter can be identified by the complier.

1.6. Scope

- Local variables of ‘program’ function are visible to free functions.
- Except the local variable of ‘program’ function, each identifier declared in a given scope can only be used inside this scope. In cases where a sub-scope contains a re-declaration of an identifier, the local definition overrides the higher-level one.
- Each identifier (function, variable or parameter) should be declared exactly once in a given scope.

1.7. Other

- Except the function return, variables of class are allowed anywhere a variable declaration is allowed, including as a parameter to a class member function, local variables of a class member function.
- Array variables are allowed anywhere as the grammar specifies.
- Functions are allowed to be called anywhere as the grammar specifies.

2. Semantic Errors & Warnings

The following are the semantic errors can be found by this program:

- Undeclared identifier
- Undeclared type
- Variable redefinition
- Class redefinition
- Function redefinition
- 'xxx' is not a member of 'yyy' // class yyy does not have member xxx
- left of '.xxx' must have class type
- Function parameters mismatch
- Cannot convert from 'xxx' to 'yyy'
- Type should be integer or real // for addOp, addOp and multOp
- Invalid array index type: xxx
- Invalid array dimensions: xxx // added in assignment 4.

The following are the semantic warnings defined by this program:

- Convert from 'integer' to 'real' // addOp, addOp and multOp
- Convert from 'real' to 'integer' // assignOp
- Convert parameter x from 'real' to 'integer' // x is the sequence number of the parameter
- Convert parameter x from 'integer' to 'real'

3. Design

3.1. Usage of Registers

- R15
  Function return address
- R14
  Stack pointer
- R13
  Used by stack operations: push and pop
- R12
  Used when loading a float number or print a float number
- R11
  Used to calculate the memory offset for array and class
- R10
  Used to store the memory offset when operating a class member variable
- R3 & R2 & R1
  Used by expressions

3.2. Numbers

As the size limitation of the Moon simulator instructions, if the number exceeds the range of the 16-bit K, the number cannot assign to K directly. So the numbers, including floating numbers, are all stored in the data section of the ASM code.

3.3. Stack

The top of the stack is topaddr - 4. (topaddr is a reserved symbol by Moon simulator.)
R14 points to the address of last element of the stack.

3.4. Function definition

Pop the parameters from the stack in the reverse order.
If parameter is passed by reference, when loading its data, it needs to load again.

3.5. Function calling

3.5.1. Before calling itself

Push the data of all parameters which are passed by value into the stack.
Push the data of all local variables into the stack. If it is an array or class, push all words of the memory into stack.

3.5.2. Before calling the function

Push R15, which stores the return address, into the stack.
Push the parameters into the stack one by one.

3.5.3. After calling the function

Pop R15 from the stack.

3.5.4. After calling the itself

Pop the data of all local variables from the stack in the reverse order.
Pop the data of all parameters which are passed by value from the stack in the reverse order.

3.6. Floating Number

- Fixed-bits representation
  The most significant 24 bits are used to store the integer part of the number. The left 8 bits are used to store the fractional part of the number.
  This way has some problems:
  It is not easy to perform the carrier number.
  8 bits fractional number is too easy to exceed to a negative number.

  So finally, the following way is used for the complier.
- Multiplication
  Multiplying all floating numbers by 1000.
  If two floating number do multiplication, dividing the result by 1000.
  If two floating number do division, multiply the first number by 1000 before operating.

3.7. Array

- Get the start address of the array variable.
- Then to calculate the offset of the array element.

1) Before parsing '['
2) Push offset
3) After parsing ']'
4) Pop offset
5) Calculate the offset.

3.8. Class

3.8.1. Using variable with class type

Calculate the memory offset, including the offset of array.

3.8.2. Using member variable

Get the address of this class variable.
Calculate the offset.

3.8.3. Member function definition

Pop the address of the class variable.

3.8.4. Before calling the member function

Push the address of the class variable.

3.9. Read & write

Separately call 'getint' and 'putint' ASM functions which has been provided by Moon simulator in the newlib.m file.
As a result, the read function does not support to get a floating number.
For write function, before calling putint, it needs to convert the floating number to two integer numbers.

3.10. Complier optimization

This complier is developed in a hurry because of studying for the coming final exams. Some part of the semantic actions is not well designed, especially:

- Information of the variable of class type
  If it is not well organized, it is difficult to calculate the memory offset for class members.
- Class member function calling
  How to easily get the memory address of the member variable?
- Load and store the data of variable
  The key is to calculate the memory offset.

4. How to run

1. Install JRE (v5.0 or higher)
2. Extract COMP6421Ass3.zip to $DIR
3. cd $DIR\COMP6421Ass3
4. $ java -jar COMP6421Ass3.jar

The results of symbol tables (xxx_result.txt), error messages (xxx_error.txt) and ASM code (xxx_asm.m) are stored under $DIR\output for each input file.

5. cd $DIR\output
6. moon newlib.m xxx_asm.m

Note that there are already several testing files under $DIR\input:

- t0_syntax_err.txt
  Do not try to moon.exe this file, because it is just used to show all the possible error and warning messages the complier can be found.
  Run the following test cases with Moon simulator:
- t1_expr_func.txt
- t2_float.txt
- t3_array.txt
- t4_passbyaddr.txt
- t5_class_array.txt
- t6_class_func.txt
- t7_class_func.txt

If you use your own test cases, after running step 4, please check the xxx_error.txt file to see if there is any error happens. If have, please correct the grammar mistakes according to the messages before executing step 6.

Appendix A: Revision History

Version	Date	Author	Remark
V0.1	Apr. 8, 2012	Yuan Tao	Draft