6G6Z1110 – Programming Languages: Principles and Design
Dr Ryan Cunningham: room E142 (ryan.cunningham@mmu.ac.uk)

Manchester Metropolitan University

Coffee Language Specs

Reserved Words

import	void	int	float	bool
break	continue	return	if	else
for	in	while	true	false

Whitespace

Spaces, newlines (\n, \r), form feeds (\f), and tabs (\t) are whitespace which may appear between lexical tokens.

Code Comments

Coffee uses single and multi-line comments in the same way as C or Java (see below).

```
// single line

/*

multi-line

*/
```

Operator Precedence

Operator	Description
(type)	Type cast
-	Unary minus
!	Logical not
* / %	Multiply, divide, modulo
+ -	Add, subtract
< <= > >=	Relational
== !=	Equality
&&	Conditional and
||	Conditional or

Manchester Metropolitan University

Backus-Naur Form

<program> → { <import_stmt> | <global_decl> | <method_decl> | <block> }*

<import_stmt> → **import** <id>+, **;**

<global_decl> → <var_decl>

<id> → <alpha> <alpha_num>*

<alpha> → { **a** | **b** | **c** | … | **z** | **A** | **B** | **C** | … | **Z** | **_** }

<alpha_num> → <alpha> | <num>

<num> → { **0** | **1** | **2** | … | **9** }

<var_decl> → <data_type> { <var> [**=** <expr>] }+, **;**

<data_type> → **int**
| **float**
| **bool**

<var> → <id>
| <id> **[** <int_lit> **]**

<int_lit> → <num>+

<method_decl> → <return_type> <id> **(** { <data_type> <id> }*, **)** { <block> | <expr> }

<return_type> → **void** | <data_type>

<block> → **{** { <var_decl> | <block> }* **}**
| <statement>

<statement> → <method_call> **;**
| <location> <assign_op> <expr> **;**
| **if (** <expr> **)** <block> [**else** <block>]
| **for (** <loop_var> **in** { <id> | <limit> } **)** <block>
| **while (** <expr> **)** <block>
| **return** [<expr>] **;**
| **break ;**
| **continue ;**
| **;**

<loop_var> → <id>

<method_call> → <id> **(** <expr>*, **)**

<expr> → **(** <expr> **)**
| **(** <data_type> **)** <expr>
| **-** <expr>
| **!** <expr>

Manchester Metropolitan University

	|	<expr> <op> <expr>
	|	<expr> **?** <expr> **:** <expr>
	|	<literal>
	|	<location>
	|	<method_call>
<op>	→	<assign_op>
	|	<arith_op>
	|	<rel_op>
	|	<equal_op>
	|	<cond_op>
<assign_op>	→	**=**
<arith_op>	→	**+** | **-** | ***** | **/** | **%**
<rel_op>	→	**>** | **>=** | **<** | **<=**
<equal_op>	→	**==** | **!=**
<cond_op>	→	**&&** | **||**
<literal>	→	<int_lit> | <float_lit > | <bool_lit> | <string_lit> | <char_lit>
<float_lit>	→	<num>+ **.** <num>* | <num>* **.** <num>+
<bool_lit>	→	**true** | **false**
<string_lit>	→	**"** <valid_char>* **"**
<valid_char>	→	*a valid C char* (see footnotes)*
<char_lit>	→	**'** <valid_char> **'**
<location>	→	<id> | <id> **[** <expr> **]**
<limit>	→	**[** [<low>] **:** [<high>] [**:** <step>] **]**
<low>	→	<expr>
<high>	→	<expr>
<step>	→	<expr>

** a <valid_char> is any single character which is not a newline, tab, form feed, double or single quote.*
*** a <valid_char> can also be any of the following double characters:* **\n \t \f \r \" \'**.

Manchester Metropolitan University

Code Generation

Coffee generates unoptimised x86-64 GNU Assembly (GAS) code.

Scoping

1. All methods including main have their own stack frame*

2. For loops have their own scope

3. All code blocks (code within curly braces) have their own scope

*stack frames must be 16-byte aligned to use C library functions

Type Precedence

Data Type	Description/Representation
float	64-bit (double precision) float
int	64-bit integer
bool	64-bit integer

6G6Z1110 – Programming Languages: Principles and Design
Dr Ryan Cunningham: room E142 (ryan.cunningham@mmu.ac.uk)

Manchester Metropolitan University

Semantic Rules

The following rules should be implemented and enforced at compile-time.

1. Variables must be declared before use
2. Variable declarations must have unique identifiers in a scope
3. Method declarations (including imported methods) must have unique identifiers in a scope
4. Method calls must refer to a declared method with an identical signature (return type, and number and type of parameters)
5. Method calls referring to imported methods must produce a warning to check the argument and return types match that of the imported method
6. Void methods cannot return an expression
7. Non-void methods must return an expression
8. The main method does not require a return statement, but if it has one, it must be of type int
9. Branch statements (if-else) containing return statements do not qualify a method as having a return statement and a warning must be issued unless they appear in both the main branch and the else branch
10. Loops containing return statements do not qualify a method as having a return statement and a warning must be issued
11. The expression in a branch statement must have type bool
12. The expression in a while loop must have type bool
13. The low and high expressions in a limit must have type int
14. Arrays must be declared with size greater than 0
15. The id in a for-loop must reference a declared array variable
16. Arrays cannot be assigned during declaration
17. Char expressions must be coerced to int
18. The expression in an assignment must have type bool, int or float
19. Locations of the from <id> **[** <expr> **]** must refer to a declared array variable
20. In a location, array indices must have type 'int'
21. The expression in unary minus operation must have type int or float
22. The expression in logical not operation must have type bool
23. The expression(s) in an arithmetic operation must have type int or float
24. The expression(s) in a logical operation must have type bool
25. The singular expression in a block (expr) provides a valid return value for a method without requiring the return keyword
26. Methods returning void cannot be used in an expression
27. Break and continue statements must be contained within the body of a loop.