Language Specification

Patrick Vatterott

May 5, 2013

1 Basic Types

1.1 Identifier

1.2 Numeric Literal

Numeric literals can be expressed in two ways:

1) **Decimal:** Decimal literals are expressed as base 10 integers. Formally:

   $'0' | '1'..'9' ('0'..'9')*$

2) **Double:** Double literals are expressed in base 10 and must contain a decimal. Formally:

   $('0'..'9')+ '.' ('0'..'9')*$

1.3 Identifier

A variable/function name, or identifier, takes the form:

$LETTER (LETTER | '0'..'9')*;$

$LETTER$ is defined as: $('\$' | 'A'..'Z' | 'a'..'z' | '_');$

The placeholder ¡id¿ is used to indicate an identifier.

2 Functions

This language supports function definitions and calls that are non-recursive and in the same file. Function definitions are of the form:

<type> <id>(<params>) {

}

Where ¡params¿ is a comma separated list of ¡type¿ ¡id¿ declarations. For example:

```
int foo(int a, double b, int c) {

}
```

Functions do not need to be declared above where they are used. The file has a comprehensive function namespace.

Recursive and mutually recursive functions are not allowed. The graph of function calls with main as the root node must form a directed acyclic graph (DAG).

3 Expressions

3.1 Expressions

An expression can be an identifier, a literal, or a binary expression. Binary expressions take the form:

```
(<id>|<literal>) <op> (<id>|<literal>)
```

<	>	<=	>=	==	!=
+	−				
*	/				

3.2 Assignment Expressions

We support assignments of the form:

```
<id> = <expression>;
<id> += <expression>;
<id> -= <expression>;
<id> *= <expression>;
<id> /= <expression>;
```

Expressions of this form will be referred to with the placeholder :

```
<assignment>
```

4 Control Flow

4.1 For

```
for (<assignment>; <expression>; <assignment>) {

}
```

4.2 While

```
while (<expression>) {

}
```

4.3 If

If statements

```
if (<expression>) {

}
```

If statements can also contain an else clause (but no elseif clauses)

```
if (<expression>) {

} else {

}
```