C♭ Language Concrete Syntax

1 C♭ Syntax

In the specification below, orange characters are terminals. Integers, identifiers, and comments are specified using regular expressions (extended with some common notation) while the rest of the language is specified using a context-free grammar.

$$
\begin{aligned}
n \in \mathit{Integer} &::= \text{-}^{?}[\text{0–9}]^{+} \\
\mathit{id} \in \mathit{Identifier} &::= [\text{a–zA–Z}]^{+}[\text{0–9a–zA–Z}]^{\star} \\
\mathit{comment} \in \mathit{Comment} &::= \text{//}.^{\star}\backslash\text{n}
\end{aligned}
$$

An integer is a non-empty sequence of digits, optionally preceded by a negative sign. An identifier is a non-empty sequence of alphanumeric characters that must begin with an alphabetic character. A comment starts with '//' and ends at the next newline.

$$
\begin{aligned}
\mathit{aexp} \in \mathit{ArithmeticExp} ::= {} & n \mid \mathit{id} \mid \mathit{aexp} + \mathit{aexp} \mid \mathit{aexp} - \mathit{aexp} \mid \mathit{aexp} * \mathit{aexp} \mid (\mathit{aexp}) \\
\mathit{rexp} \in \mathit{RelationalExp} ::= {} & \mathit{aexp} < \mathit{aexp} \mid \mathit{aexp} = \mathit{aexp} \mid \mathit{aexp} <= \mathit{aexp} \mid \mathit{rexp} \;\&\&\; \mathit{rexp} \\
& \mid \mathit{rexp} \;||\; \mathit{rexp} \mid !\mathit{rexp} \mid [\mathit{rexp}]
\end{aligned}
$$

An arithmetic expression is an integer, variable, or two arithmetic expressions combined with a plus, minus, or times operator. A relational expression is two arithmetic expressions combined with a less-than, equals, or less-than-or-equal-to operator; or two relational expressions combined with logical **and** or **or**; or a relational expression preceded by **not**. For arithmetic expressions, the user can indicate grouping using parentheses. For relational expressions, the grouping is indicated with brackets–distinguishing the use of brackets helps keep parsing simple, as we are going to see later in class.

$$
\begin{aligned}
\mathit{stmt} \in \mathit{Statement} &::= \mathit{assign} \mid \mathit{whileLoop} \mid \mathit{forLoop} \mid \mathit{cond} \\
\mathit{assign} \in \mathit{Assignment} &::= \mathit{id} := \mathit{aexp}; \mid \mathit{id} := \mathit{call}; \\
\mathit{whileLoop} \in \mathit{WhileLoop} &::= \text{while } \mathit{rexp} \text{ \{ } \mathit{block} \text{ \}} \\
\mathit{forLoop} \in \mathit{ForLoop} &::= \text{for } \mathit{var} \text{ from } \mathit{aexp} \text{ to } \mathit{aexp} \text{ \{ } \mathit{block} \text{ \}} \\
\mathit{cond} \in \mathit{Conditional} &::= \text{if } \mathit{rexp} \text{ \{ } \mathit{block} \text{ \}} \mid \text{if } \mathit{rexp} \text{ \{ } \mathit{block} \text{ \} else \{ } \mathit{block} \text{ \}}
\end{aligned}
$$

A statement is an assignment, a while loop, a for loop, or a conditional. The left-hand side of an assignment can be either an arithmetic expression or a function call. A conditional may or may not have an **else** branch. The for loops are range-based rather than the C-style for loops.

$$
\begin{aligned}
\mathit{type} \in \mathit{Type} &::= \text{int} \\
\mathit{decl} \in \mathit{Declaration} &::= \mathit{type}\ \mathit{id}; \\
\mathit{decls} \in \mathit{Declarations} &::= \epsilon \mid \mathit{decl}\ \mathit{decls} \\
\mathit{stmts} \in \mathit{Statements} &::= \epsilon \mid \mathit{stmt}\ \mathit{stmts} \\
\mathit{block} \in \mathit{Block} &::= \mathit{decls}\ \mathit{stmts}
\end{aligned}
$$

A variable declaration is a type followed by a variable name. A block of statements is a (possibly empty) sequence of declarations followed by a (possibly empty) sequence of statements.

$$
\begin{aligned}
\mathit{call} \in \mathit{FunctionCall} &::= \mathit{id}(\mathit{args}) \mid \mathit{id}() \\
\mathit{args} \in \mathit{Arguments} &::= \mathit{aexp} \mid \mathit{aexp}, \mathit{args}
\end{aligned}
$$

A function call is the name of the function followed by a (possibly empty) sequence of arithmetic expression arguments in parentheses separated by commas.

$$\begin{aligned} \mathit{fundef} \in \mathit{FunctionDef} &::= \text{def } \mathit{id}(\mathit{optparams}) \ : \mathit{type} \ \{ \ \mathit{block} \ \text{return } \mathit{aexp}; \ \} \\ \mathit{params} \in \mathit{Parameters} &::= \mathit{type}\ \mathit{id} \ \mid \ \mathit{type}\ \mathit{id}, \mathit{params} \\ \mathit{optparams} \in \mathit{OptionalParameters} &::= \epsilon \ \mid \ \mathit{params} \end{aligned}$$

A function definition provides the function name; a (possibly empty) sequence of parameters giving the type and name of each parameter, seperated by commas; the function's return type; and the function body consisting of a block of statements followed by a return statement.

$$\begin{aligned} \mathit{fundefs} \in \mathit{FunctionDefs} &::= \epsilon \ \mid \ \mathit{fundef}\ \mathit{fundefs} \\ \mathit{program} \in \mathit{Program} &::= \mathit{fundefs}\ \mathit{block}\ \text{output}\ \mathit{aexp}; \end{aligned}$$

A program consists of a (possibly empty) sequence of function definitions followed by a block of statements followed by an output that will be printed by the program.

2 Example Program

This progam doesn't do anything sensible, it's just an example of the language syntax in use.

```
def foo(int a, int b) : int {
  int x;
  int y;

  x := a + b * (a - b);

  while (y <= x + 3) {
    y := y + 10;
  }

  for (c from a to b * 2) {
    y := y + x;
  }

  return y * 2;
}

def bar() : int { return 42; }

int t1;
int t2;

t1 := bar();
t2 := foo(t1 + 2, t1 * 2);
if (t1 < t2) { t1 := t2; }

output t1 + 42;
```