Next
Language Reference Manual

Ernesto Arreguin (eja2124)
Danny Park (dsp2120)
Morgan Ulinski (mu2189)
Xiaowei Zhang (xz2242)

Contents

1 Introduction

TODO

2 Lexicon

The Next programming language uses a standard grammar and character set. Characters in the source code are grouped into tokens, which can be punctuators, operators, identifiers, keywords, or string literals. The compiler forms the longest possible token from a given string of characters; tokens end when white space is encountered, or when it would not be possible for the next character to be part of the token. White space is defined as space characters, tab characters, return characters, and newline characters.

The compiler processes the source code and identifies tokens and locates error conditions. There are three types of errors:

- Lexical errors occur when the compiler cannot form a legal token from the character stream.
- Syntax errors occur when a legal token can be formed, but the compiler cannot make a legal statement from the tokens.
- Semantic errors, which are grammatically correct and thus pass through the parser, but break another Next rule. For example, it is possible to `kill` a character or item, but not a location.

2.1 Character Set

The Next programming languages accepts standard ASCII characters.

2.2 Identifiers

An identifier is a sequence of characters that represents a name for a:

- Variable
- Location
- Character

- Item
- Action

Rules for identifiers:

- Identifiers consist of a sequence of one or more uppercase or lowercase characters, the digits 0 to 9, and the underscore character (_).
- Identifier names are case sensitive.
- Identifiers cannot begin with a digit or an underscore.
- Keywords are not identifiers.

2.3 Comments

Comments are introduced by /* and ended by */, except within a string literal. Comments cannot be nested. If a comment is started by /*, the next occurrence of */ ends the comment.

2.4 Keywords

Keywords identify statement constructs and specify basic types. Keywords cannot be used as identifiers. The keywords are listed in Table 1.

Table 1: Keywords

`if`	`then`	`else`	`and`	`or`
`start`	`end`	`when`	`choose`	`kill`
`grab`	`hide`	`exists`	`drop`	`output`
`character`	`location`	`action`	`item`	`int`
`string`	`next`			

Keywords are used:

- To qualify a data type (`character, location, action, item, int, string`)

- As part of a statement (`if, then, else, and, or, start, end, when, choose, kill, grab, hide, exists, drop, output`)

2.5 Operators

Operators are tokens that specify an operation on at least on operand and yield a result (a value, side effect, or combination). Operands are expressions. Operators with one operand are unary operators, and operators with two operands are binary operators.

Operators are ranked by precedence, which determines which operators are evaluated before others in a statement.

Some operators are composed of more than one character, and others are single characters.

The single character operators are shown in Table 2.

Table 2: Single-character operators

+	-	*	/	<
>	=			"
.	{	}	(	)

The multiple-character operators are shown in Table 3.

Table 3: Multiple-character operators

>=	<=	==	!=	and	or

2.6 Punctuators

Table 4 shows the punctuators in Next. Each punctuator has its own syntactic and semantic significance. Some characters can either be punctuators or operators; the context specifies the meaning.

Table 4: Punctuators

{ }	Declaration block or compound statement delimiter
( )	Sub-declaration list; also used in expression grouping
,	Sub-declaration separator
;	Statement end
=	Declaration initializer
“ ”	String literal

2.7 String and integer literals

Strings are sequences of zero or more characters. String literals are character strings surrounded by quotation marks. String literals can include any valid character, including white-space characters.

Integers are used to represent whole numbers. Next does not support floating point numbers. Integers are specified by a sequence of decimal digits. The value of the integer is computed in base 10.

3 Basic Concepts

3.1 Blocks

A block is a section of code surrounded by braces { }. Blocks are used to surround compound statements, a set of related statements enclosed in braces. Since Next uses global scope (except in the case of actions), blocks do not affect the scope of a variable.

3.2 Scope

All declarations except actions are made at the beginning of the program, and have global scope. Actions are declared within a choose statement and their scope is that choose statement.

3.3 Side Effects and Sequence Points

Any operation that affects an operand's storage has a side effect. This includes the assignment operation, and operations that alter the items, attributes, etc. within a location or a character.

Sequence points are checkpoints in the program where the compiler ensures that operations in an expression are concluded. The most important example of this is the semicolon that marks the end of a statement. All expressions and side effects are completely evaluated when the semicolon is reached.

4 Data Types

A type is assigned to an object in its declaration. Table 5 shows the basic types that are used in Next.

Table 5: Basic Data Types

integer
string
location
character
item
(attribute)
(action)

5 Declarations

Declarations introduce identifiers (variable names) in the program, and, in the case of the complex types (character, location, item), specify important information about them such as attributes. When an object is declared, space is immediately allocated for it.

5.1 Primitive Types

The primitive types in Next are integer and string. These can stand on their own, or they can serve as attributes within a complex type. Primitive types are declared as follows:

```
int identifier = value
string identifier = value
```

where:

- `identifier` stands in for a variable name.
- `value` stands in for an expression.

5.2 Complex Types

Each of the complex types in Next (item, character, and location) has its own declaration syntax. The declarations are as follows:

```
item identifier = { primitive_declaration_list }

character identifier = { primitive_declaration_list,
                         item_list }

location identifier = { primitive_declaration_list,
                        item_list,
                        character_list }
```

where:

- `identifier` stands in for a variable name.
- `primitive_declaration_list` stands in for a list of attribute declarations in the form `(declaration_1, declaration_2, ... , declaration_n)`, and each `declaration` is in the form described above in the description of primitive types. These represent the attributes (and values for those attributes) for a given item, character, or location.

- `item_list` and `character_list` stand in for lists of item and character variable names, respectively, in the form `(name_1, name_2, ... name_n)`. These represent the list of items a character is carrying or are found in a location, and the characters that are physically in a location.

6 Expressions and Operators

An expression is a sequence of Next operators and operands that produces a value or generates a side effect. The simplest expressions yield values directly, such as ints, strings, and variable names. Other expressions combine operators and subexpressions to produce values. Every expression has a type as well as a value. Operands in expressions must have compatible types.

6.1 Primary Expressions

6.2 Overview of the Next Operators

6.3 Unary Operators

6.4 Binary Operators

7 Statements

7.1 Labeled Statements

7.2 Compound Statements

7.3 Expression Statements

7.4 Selection Statements

7.5 Iteration Statements

8 Examples