Vowel: Language Reference Manual

The Way for Wordsmiths

Name	UNI
Coby Simler	zys2102
Aidai Beishekeeva	ab5248
Lex Mengenhauser	am4958
Vikram Rajan	vjr2123

Vowel: Language Reference Manual **1**
- Introduction 2
- Data Types 2
  - Primitive Data Types 2
  - Data Structures 2
  - Type Conversion and Casting 2
  - Structs 3
- Variables 3
  - Comments 4
- Operators 4
  - General operators 4
  - String operators 4
- Functions 6
  - Built in functions 7
- Scope 7
- Lexical Conventions 7
  - Keywords 7
  - Expressions 7
  - Unary expressions 8
  - Conditional x operators 8
- Sequencing and Control Flow 8
  - Sequencing 8
  - Loops 8
- Example Program 9

Introduction

Vowel is a high-level, imperative, statically-typed programming language intended to be used to iterate over, operate on, and manipulate large text inputs. To do this, Vowel introduces an extended collection of string operators and supports a novel control flow syntax tailored for text processing. The language has Python-style control flow and function definition syntax with Java-like typing. Vowel enables programmers to quickly analyze, manipulate, and map functions

Data Types

Primitive Data Types

Type	**Description**	**Example Declaration.**
int	A 32-bit integer value.	int i = 42;
string	String is a sequence of one or more consecutive ASCII characters surrounded by quotation marks (“).	string vowel = “consonant”;
bool	An 8-bit boolean value.	bool columbia = true;

Data Structures

array	A type-specific array of ASCII characters of predetermined length.	int ex[5] = [1,2,3,4,5];

Type Conversion and Casting

The Vowel language does not allow for any implicit type promotion or casting. Instead, programmers may use the below methods to express a given type to another desired type.

String → Int

- Strings may be converted into ints via the int(x) command. The string input into this command must only consist of characters between 0-9 (i.e. regex -?([‘0’-’9’])+). Leading zeros are removed from the input expression. For example, int(“145”) would evaluate to 145.

Int → String

- Ints may be converted to strings via the str(x) command. The int input is returned as a string represented semantically by ASCII characters. For example, str(42) would evaluate to 42.

Bool → String

- Booleans may be converted to strings via the str(x) command. The bool inputs (either True or False) are returned as strings, where str(True) returned “True” and str(False) returns “False.”

Structs

Structs, like in C, are collections of variables under a single data type. Structs (structures) are declared using the keyword `struct` and the name of the struct followed by parenthesis `().` Structs in Vowel must be declared with accompanying member variables. Functions cannot be placed inside of structs, member variables are accessed by the struct variable name followed by a period and the name of the member variable the programmer wants to access. Unlike structs in C, Vowel has no concept of a struct *identifier* or *tag*; each struct is treated as a distinct identity regardless of the types of its member variables.

For example, structs can be declared as follows:

```
struct citation {
    string sentence = "To the well-organized mind, death is but the next
                       great adventure.";
    string title = "Harry Potter and the Sorcerer's Stone";
    string author = "J.K. Rowling.";
    int year_published  = 1997;
};
citation.year_published;
→1997
```

Variables

Variable Naming

Variable names are sequences of alphanumeric characters (letters and digits), including the underscore ‘_’. Variable names may start with an underscore, but not with a number, and cannot include any symbols. Furthermore, no variable can have the same name as any reserved word.

Variable Declaration
To declare a variable, first state the data type, then an appropriate variable name followed by the assignment operator, ‘=’, and then the associated value. Declared variables must be initialized in place; Vowel does not allow declaration without initialization.

```
int x = 4;
string temp = “Hello World!”;
```

Comments

All comments will be opened with /* and will comment out everything until a closing */ is found. If no closing */ is found, an error will be thrown. There is no single-line comment syntax, as everything will follow this open and close comment convention.

```
/* Single-line comment */

/* Multi-line comment
This is still commented */
```

Operators

General operators

```
= assignment    integers, strings, bool
+ sum           integers, strings
- difference    integers, strings
* product       integers
/ quotient      integers
% modulo        integers
+= increment    integers, string
-= decrement    integers
```

String operators

Plus Operator

For ints, the plus operator will return the sum of the two values. For strings, the plus operator ‘+’ when used on two strings will concatenate them and return the combined string.

```
string first = “Hello ”;
string second = “world!”;
string new = first + second;
print(new); // prints: Hello World!
```

Minus Operator

For ints, the minus operator will perform the difference of the numbers as an int,
The minus operator ‘-’ produces an array of words that appear in the first string but not the second string. The order of the output will follow the order of how those words appear in the first string.

```
string first = “this is a random sentence.”;
string second = “another random string is this”;
string result[] = first - second;
print(result); // prints: [“a”, “sentence”]
```

Slice Operator
The slice operator ‘[“delim”; start: end]’ when used on a string produces the subset of the original string based on the indices in the brackets and the delimiter provided. This operator takes three parameters: a delimiter, start index and end index. The first parameter, delimiter, will be whatever the user defines and describes how the string will be split. The second parameter describes the starting index, where 0 represents the first word in the string (inclusive), and the third parameter describes the ending index and is not inclusive. The operation will return as many results as possible if the index is larger than the number of results. The *end* value can be left out, in which case the operator returns all values until the end of the input.

```
string varname = "This is a random sentence with some words.";
string sliced = varname[" "; 1:4];
print(sliced); // prints: “a random sentence”
```

Union Operator
The union operator ‘|’ produces an array of all words that appear in either or both strings and removes multiple occurences. In this case, words are defined as consecutive strings of ASCII text separated by a combination of any punctuation and spaces.

```
string first = “this is a random sentence.”;
string second = “another random string is this”;
string result[] = first | second;
print(result);
// prints: [“this”, “is”, “a”, “random”, “sentence”, “another”,
“string”]
```

Intersection Operator
The intersection operator ‘&’ produces an array of words that appear in both strings. The order of the strings in the array will be the same order as they appear in the first string.

```
string first = “this is a random sentence.”;
string second = “another random string is this”;
string result[] = first & second;
print(result); // prints: [“this”, “is”, “random”]
```

Functions

Functions in Vowel are run sequentially. The syntax of a function definition is to have the return type of the function, the name of the function, followed by parentheses containing zero or more comma separated arguments formatted as `type name` pairs. Functions that do not return anything are type `void` which is a reserved keyword in Vowel. Arguments in Vowel are passed by value meaning a copy of them is made to be passed to the function. The body of the function is contained by curly braces that open after the function argument’s closing parenthesis, and close after the last line of the function definition.

Program definition that takes a string in as an argument and prints it out:

```
string myString = "Hello World!";
string myFunc(string s){ /*program definition*/
      return s;
 }
```

Functions are called by using the function name followed by parenthesis including the required number of arguments dictated by the function definition. Vowel does not support functionality to include more arguments than dictated in the function definition. Calling the function above looks like:

```
myFunc(myString);
→ Hello World!
```

Variables can be set to contain the results of functions so long as they have the same type as the functions return value. Using the examples above, this looks like:

```
string w = myFunc(myString);
print(w);
→ Hello World!
```

Built in functions

- `len()` - used to find the length of a string

Scope

Scope is bounded by the use of curly braces `{}` as Vowel is a block structured language. The lifetime of a specific variable or function is determined by the scope of the block it is within.

Lexical Conventions

Keywords

Keywords are lower-case sequence of characters reserved for use in Vowel

```
int, bool, string, array, for, if else, struct, continue, break,
print, return, true, false
```

Expressions

An expression is a series of operations and function calls applied to operands. It may consist of one or more operands and zero or more operators.

Example:

```
3 + 4; int x = int a = 3; x + a; 10;
```

Unary expressions

Unary expressions are applied to one operand.

```
!   negate       bool
-   negative     integers
```

Conditional x operators

```
>  greater than              integers, strings
<  less than                 integers, strings
>= greater than or equal     integers, strings
<= less than or equal        integers, strings
== equal                     integers, strings, arrays, bool
!= not equal                 integers, strings, arrays, bool
and                          bool
or                           bool
```

Strings
Like in Python, the <, >, <=, >=, ==, and != operators compare the input strings alphabetically and return a boolean value.

```
string a = “a”;
string b = “b”;
bool x = a < b; /* true */
```

Sequencing and Control Flow

Sequencing

Expressions in Vowel are separated by the “;” character and are evaluated left to right. Every sequence in Vowel must terminate in a “;” character.

Loops

Vowel offers two types of loops: while loops and for loops.

While loops are constructed precisely like while loops in C:

```
while (condition == true){
     /* do something */
}
```

For loops have two types of constructions: explicit sequencing and implicit sequencing.

Explicit Sequencing:
For loops with explicit sequencing are written like Java or C. This gives the developer the flexibility to define precisely the terms of the For loop. The syntax is in the structure:

for (initialize variable ; conditional expression; post-loop expression) { / body of loop */ };*

For example:

```
for (int i = 0; i < 42; i++) {
     print(i);
}
```

Control flow keywords:
The *continue* keyword forces the next iteration of the loop to take place. In for loops with explicit sequencing, this leads to the conditional statement being evaluated. In for loops with implicit sequencing, this leads to the next valid element in the sequence (if there is one) to be loaded up.

The *break* keyword forces the termination of the loop altogether, leading to the end of the loop expression.

Example Program

```
string getLongestSharedWord(string a, string b){
    string intersection[] = a & b;
    int longestWorldLen = 0;
    int index = 0;
    for (int i = 0; i < intersection.length; i++){
        int wordlen = len(intersection[i]);
        if (wordlen > longestWorldLen){
             longestWorldLen = wordlen; index = i;
        }
    }
    return intersection[index];
}


string grimmsFirstEdition = "Mr. Fox, who had nine tails, surreptitiously
                             climbed out of the grated window and into the
                             front yard.";
string grimmsSecondEdition = "Mr. Fox had nine tails—a feature which gave
                              him great pride. He secretly slipped out of
                              the slanted window frame into the garden.";

string first_sliced = grimmsFirstEdition["surreptitiously";0:]; /*
surreptitiously climbed out of the grated window and into the front yard.
*/
string second_sliced = grimmsSecondEdition["secretly";0:];  /* secretly
slipped out of the slanted window frame into the garden. */

string lsw = getLongestSharedWord(first_sliced, second_sliced);
print(lsw); /* prints “window” */
```