Contents

1 Lexical Conventions

1.1 White Space

Aside from its function as token separator, white space has no affect on the output or execution of the WCL compiler. White space consists of a combination of any of the following characters:

`"\n"`	`newline`
`"\r"`	`carriage return`
`"\t"`	`tab`
`" "`	`space`

1.2 Comments

WCL comment blocks can be bookended by a beginning "/*" and a final "*/". Alternatively, anything following "//" on a line is considered as a comment.

1.3 Identifiers

Identifier consist of a letter followed by a sequence of letter, digits and underscores. Upper and lower case letters are considered as different characters.

1.4 Keywords

WCL reserves the following identifiers as keywords, and cannot not be used otherwise:

```
else    float   function        if      int
new     return  string          while   void
Image   Link    Paragraph       Table   WebPage
```

1.5 Punctuation

WCL strictly adheres to the following punctuation conventions:

`( )`	`surround comparisons`
`{ }`	`surround statement blocks`
`[ ]`	`index of arrays`
`;`	`end of statement`
`.`	`object reference`
`,`	`list elements seperator`

1.6 Operators

WCL uses various operators for mathematics and logical operations. The following operators are supported:

```
+       -       *       /
<       >       ==
&       |       %
```

2 Types

WCL supports three basic data types: integers, floating point numbers and strings. A data type for single characters is not available, but can instead implemented by a string containing a single character.

2.1 Integers

Integers consist of one or more digits and are optionally signed. They are stored as a 32-bit block and can range in value from -2147483648 to 2147483647.

2.2 Floating Point Numbers

Floating point numbers consist of an integer part, a decimal point, an 'e' and an optionally signed integer exponent. The integer and fraction parts both consist of a sequence of digits. Either the integer part or the fraction part (but not both) may be omitted; either the decimal point or the e and the exponent (but not both) may be omitted. Like integers, floating point numbers are stored as 32-bit blocks. They can range in value from 1.4E-45 to 3.4028235E+38 with both positive or negative signings. They can also take on the positive of negative values of infinity.

2.3 Strings

A string is a sequence of characters, and is declared within a set of double quotes. Within a quote, a double quote is represented by a slash followed by double quotes.

3 Syntax Notation

In the syntax notation used in this manual, syntactic categories are indicate by *italics*, while literal words and characters are in `typewriter type`. Optional parts are indicated with a subscript "opt". For example,

$$expression_{opt}$$

represents an optional expression.

4 Objects

There exist five basic object types in WCL, each with its own set of methods.

4.1 WebPage

`WebPage` *`web-page-name`* `= new WebPage;`
The WebPage object is the object used to contain the contents of an entire webpage. There can be numerous instances of this object within a WCL file and each can be output (printed) to a separate file.

The WebPage object contains the following methods:

4.1.1 Add

`WebPage.Add(`*object*`);`
The `WepPage.Add` method takes as its input a single object (of types: Image, Link, Paragraph or Table) and inserts it into the webpage after the object most recently inserted there. Only a reference to (and not a copy of) the object is inserted and if the object is changed before the `Print` method is called, the updated object will be output to the html page.

4.1.2 Print

`WebPage.Print(`*filename*`);`
The `WebPage.Print` method creates a webpage file from the WebPage object. The webpage is stored as *filename*`.html` in the present working directory.

4.2 Paragraph

`Paragraph` *`paragraph-name`* `= new Paragraph;`

The Paragraph object contains the following methods:

4.2.1 Add

`Paragraph.Add(`*text*`);`
The `Paragraph.Add` method appends the argument string to the existing paragraph.

4.3 Link

`Link` *`Link-name`* `= new Link((`*text*|*image*`),` *url*`);`
The Link constructor takes two arguments, *text* or *image* and *url* and sets them as the values for the Link *value* and *url* variables. When the `WebPage.Print` method is used, the *value* variable becomes the text or image that when clicked on, directs the user to the location specified by the *url* variable. Optionally, the Link object can be declared with the following statement that leaves the variables *value* and *url* uninitialized:

`Link` *`Link-name`*`;`

4.3.1 setText

`Link.setText(`*text*`);`
The `Link.setText` method sets *text* as the text of the link as it will be displayed on the webpage. This method can be called numerous times, as the link text is variable.

4.3.2 setImage

`Link.setImage(`*location*`);`
The `Link.setText` method sets *location* as the location of the image of the link to be displayed on the webpage. This method can be called numerous times, as the link image is variable.

4.3.3 setURL

`Link.setURL(`*url*`);`
The `Link.setURL` method sets the *URL* variable as the address of the document to link to. This method can be called numerous times, as *URL* is variable.

4.4 Image

`Image` *`image-name`* `= new Image(`*location*`);`
The Image constructor takes a string, *location*, and sets it as the image-source variable that specifies the location of the image. Optionally, the Image object can be declared with the following statement that leaves the value of the image-source variable uninitialized:

`Image` *`image-name`*`;`

4.4.1 Set

`Image.set(`*location*`);`
The `Image.set` sets the image-source variable of the object to *location*. The image-source variable is the address of the image to be displayed.

4.5 Table

`Table` *`Table-name`* `= new Table;`
A Table object is constructed empty. It contains the following method:

4.5.1 Set

Table.Set(tableArray);
The `Table.Set` method sets the *tableArray* array as the contents of the table.

4.5.2 Add

Table.Add(rowArray);
The `Table.Add` method adds the *rowArray* array to the contents of the table, after the already-existing rows.

5 Declarations

All variables must be declared before being referenced. They take on the following format:

declaration-type identifier (= assignment)$_{opt}$;

The declaration type and identifier are both required, whereas the assignment of a value to the identifier is optional. Every identifier must have a unique corresponding declaration type.

Aside from an identifier's type, it also has a lexical scope in which it can be referenced. That scope begins from the point it is declared and terminates at the end of the section it is nested in. For example, the lexical scope of variables declared within a function will begin every time the function is called and terminate at the end of that function call. At the end of an identifier's scope, the content of the variable becomes discarded and the identifier may be used once again.

To prevent ambiguity in overlapping scopes, variables may not be declared within an inner scope that have the same identifier as a variable in an outer scope.

6 Functions

Functions can be declared in the following manner:

`function` ***`return-type function-name`***(*parameters*$_{opt}$) {*declaration-list*$_{opt}$, *statement-list*}

A function declaration consists of a function identifier, followed by a list of arguments enclosed by "(" and ")". The list of arguments contains zero or more arguments separated by ",". Follows is a list of declarations and statements enclosed by "{" and "}". The function must return a value of the return type or nothing if the value is void.

7 Expressions

Expressions are the values that are assigned to variables or input into function calls. Some simple expressions include identifiers, constants and function return values. The type of the expression is the type of the identifier, constant or function return value. More complex expressions are are composed of multiple expressions, yet have a single value when evaluated. These expressions have the value of the combination of the contained expressions. The types of the simple expressions contained and the operands separating them dictates how the expressions are combined. For example, string expressions can be concatenated whereas arithmetic expressions can be evaluated by addition, subtraction, etc., depending on the operands used.

7.1 (expression)

An expression can be enclosed in parenthesis, where its value is the value of the contained expression. Parenthesis can be added to clarify precedence or to make code more readable.

8 Statements

Statements are the basic structure of the WCL file and can be grouped individually or in statement blocks. Every statements is executed sequentially, progressively adding to and expanding a webpage or its contents. Each statement must end with a ";".

8.1 Statement Blocks

A statement block is a series of one or more statements that is enclosed by "{" and "}". Within a statement block, each statement is executed in order.

8.2 Assignments

In an assignment statement, variables can be assigned values of their type as follows:

variable-name `=` *right-value*

The *right-value* must be an expression as defined below.

8.3 Conditional Statements

Conditional statements are of the form:

`if(`*condition*`)` *statement*`;` `(else` *statement*`)`$_{opt}$

In this case, the statement is executed only if the condition is evaluated to be true. The else section is optional and its statement section is executed when the condition is evaluated to be false.

8.4 Iterated Statements

A *while* loop is an iterated conditional statement where the statement is executed so long as the condition remains true. It takes the form:

`while(`*condition*`)` *statement*`;`

Since the condition is checked before the execution of each round, it is possible that the statement will never get executed.

9 Simple Example

The following example creates a simple webpage that contains a single table. The table is first created and filled in, and is then inserted into the webpage object. Finally, the webpage is saved as the file "index.html".

```
WebPage index = new WebPage();

// Create a table
Table myTable = new Table();

// Declare some variables
int i = 0;
int j = 0;
int num = 0;
int size = 10;
int tmpArray[size];

// Fill the table with values
while (i < 5)
{
    // Initialize array with values
    j = 0;

    while (j < size)
    {
        if (j % 2 == 1)
        {
                num = j * i;
        }
        else
        {
                num = j * i * 2;
        }

        tmpArray[j] = num;
        j = j + 1;
    }

    // Add the array to the table
    myTable.Add(tmpArray);
    i = i + 1;
}

WebPage.Add(myTable);
WebPage.Print("index.html");
```