CHIL - CSS HTML Integrated Language

Programming Languages and Translators - Fall 2013

Authors:
Gil Chen-Zion - gc2466
Ami Kumar - ak3284
Annania Melaku - amm2324
Isaac White - iaw2105

Professor:
Prof. Stephen A. Edwards

CONTENTS

1. INTRODUCTION

CHIL is an HTML and CSS combined website generator targeted specifically for the novel web developer. Instead of learning both the complexities of HTML and the complexities of CSS, CHIL allows the developer to create functional, stylistic, and dynamic web pages through the mastery of only one language. This language will combine both content and styling in one language so that the user isn't constantly switching between files. Furthermore, it provides the ability to programmatically loop through different types of element creation, or define more complex elements made up of basic types provided by the language.

2. LEXICAL CONVENTIONS

2.1 Comments

Multi-line comments begin with @> and end with <@. A single-line comment is preceded by @. The nesting of comments is allowed.

2.2 Primitive Types

Key Word	**Description**
`int`	non-decimal number
`string`	sequence of characters
`float`	decimal number
`boolean`	true or false value
`element`	fundamental container that can contain text or images
`style`	characteristics of an element
`array`	list of primitives of a single type

2.2.1 Integers

A data type consisting of whole number values.

Examples:
```
x = 5
y = 0
z = -20
```

2.2.2 Strings

A constant representing character strings.

Examples:
```
x = “Hello World”
y = “abc”
```

String Concatenation

Strings are combined using + operator.

Examples:
```
x = “Hello”
y = “World”
z = x+y         (“Hello World”)
or
z = “Hello” + “World”
```

2.2.3 Floats

A numeric data type containing a decimal. Examples of floats are:

```
125.3
3.0
3.
```

2.2.4 Booleans

Booleans are incorporated in the use of boolean operators. The use of a boolean operator in comparison expressions return one of two predefined constants: true or false.

Boolean operators used in comparisons:
```
==, !=, =<, =>, <, >
```

Examples:
```
x = 3
y = 3
z = 1

x == y     (evaluates to true)
x > y      (evaluates to false)
x > z      (evaluates to true)
```

```
z != y     (evaluates to true)
y => x     (evaluates to true)
```

2.2.5 Elements

Elements are the fundamental container for any object which is intended to eventually be rendered as HTML. An element's attributes determine which tag will eventually be associated with it.

Example:

```
someElementName = {
      contents: "I'm an element.",
      style: ${
      nameDeclaredByProgram: someValue
      },
      children:
      @> children not provided here, since it often makes sense to
      define them as another variable or add them on to defined
      elements later via the someElementName.children[] = operator. <@
}
```

2.2.6 Styles

All elements have a property called style, which is provided without any allowed values. Page.addStyle() can be used to add possible values to the list for allowed properties, which will tell the compiler that the keywords are permitted.

Styles are processed by by the Page.toHtml() method, which the programmer is responsible for providing. If no Page.toHtml() method is set, then the compiler will use a provided fallback that is not accessible to the programmer. If Page.toHtml() produces invalid output, the compiler will also fallback to the built in, inaccessible method. Style is defined at the time of Element declaration, but styles may be added by referencing Element.style.someStyleName. Style elements are declared with ${} syntax, using the equals operator.

Examples:

```
@>In this example, keywords have been previously provided by a
standard library<@

3pxBorderTop = ${
      border: "top,3px,solid",
      innerSpacing: "top, .25",
      outerSpacing: "top ,.25"
}
```

2.2.7 Arrays

An array is a list of objects or primitive types. All elements of an array must be of the same type. Arrays can be concatenated by using the + operator. Any array that is used in an addition operation will take precedence for returning the output as an array. However, since arrays may contain only a single type, this operation may fail if not used with matching types. Values can also be added to the end of the array using `varName[] = someVal`

Examples:

```
r = []                      (initialize an empty array)
x = [“hello”, “world”]           (array of Strings)
y = [42,34,42]              (array of type int)
r[] = 34 @add 34 to the end of the r array
s = [42] @make a new array called s
z = r + s

@> z = [34,42] <@
```

2.3 Identifiers

Identifiers consist of uppercase and lowercase letters, numbers, and underscore (_). The first letter cannot be a digit or an underscore (_). Two different identifiers cannot have the same name.

2.4 Keywords

```
element
style
page
if
endif
else
elseif
for
endfor
rtn
fn
endfn
page
```

3. EXPRESSIONS

3.1 Operators

Function Name	**Description**
`=`	assignment operator
`++,--`	increment and decrement
`+, -, *, /`	basic arithmetic operators
`==, !=`	comparison operators, by value
`<, <=, >, >=`	inequality operators, by value
`%`	modulus
`“”`	denotes a string
`(,)`	contains an expression
`&&, ||`	logical operators

3.2 Increment Operators

Increment and decrement are unary operators that work on a single integer. The increment operators follow an integer value expression, as follows:

```
expr++
```

`++`	increment
`--`	decrement

3.3 Multiplicative Operators

The multiplicative operators are `*` and `/`. Both can be used on integer or float value expressions. Multiplication and division operators are formatted as follows:

```
expr * expr
expr / expr
```

`*`	multiplicaton

/	divison

3.4 Additive Operators

The additive operators are + and -, and are also used on integer or float value expressions. The format of addition and subtraction is similar to that of multiplication and division.

```
expr + expr
expr - expr
```

+	addition
-	subtraction

3.5 Inequality Operators

Inequality operators compare two integer or two float expressions. The inequality operators are `<, >, <=, and >=`. An expression with an equality has the following format:

```
expr < expr
```

<	less than
>	greater than
<=	less than or equal to
>=	greater than or equal to

3.6 Comparison Operators

Two integer or float operands can be compared by the operators == and `!=`. For example:

```
expr == expr
```

==	equal to
!=	not equal to

3.7 Logical Operators

The two logical operators are `&&` and `||`. Two boolean value expressions can be combined by either AND or OR as follows:

```
(3 < 4) && (4 < 5)        evaluates to true
(3 > 5) || (6 != 7)       evaluates to false
```

`&&`	and
`||`	or

3.8 Precedence

The following list shows the precedence of all the operators, from highest to lowest precedence:

```
Parenthesis
Increment operators
Multiplicative operators
Additive operators
Inequality operators
Comparison operators
Logical operators
```

4. FUNCTIONS

4.1 Function Call

Functions will be called by their identifier, with arguments specified in parenthesis and separated by commas. Member functions will be called after an object and separated by a period. For example, for the function "func":

```
func(1, 2)
a.func(1, 2)
```

4.2 Recursion

Recursive functions cannot be defined. No function can call itself recursively.

4.3 Built-in Functions

`toHtml()` converts the program into HTML. If no `toHtml()` is defined, a default `toHtml()` is used. Otherwise, the new `toHtml()` is used for the conversion.

*for functions to convert types see Section 7. Type Conversions

5. DECLARATIONS

5.1 Variable Declaration

Variables are declared by their identifier and assigned values using =. Integers just include digits and floats contain a single decimal point. Strings are identified by opening and closing quotes and arrays are denoted by opening and closing square brackets. The following are examples of variable declarations:

```
a = “Hello World”
b = 42
c = [“hello”, “world”]
d = [42,34,42]
e = [a]
```

5.2 Function Declaration

Functions are declared by the keyword `fn` and an identifier, which is used to refer to the function throughout the program. Parameters are passed to the function in parenthesis and separated by commas. The body of a function is closed by the `endfn` keyword. The format of a function declaration is as follows:

```
fn identifier(parameter list)
      @ body of function
endfn
```

5.3 Scoping

Variables declared within the program scope can be accessed by all functions in the file. Variables declared within a statement (`if` or `for`) or function have a scope that lasts within the body of the statement or function, starting at the variable declaration and ending at `endif`, `endfor`, or `endfn`.

6. STATEMENTS

6.1 Conditional Statement

An if statement begins with the `if` keyword and ends with `endif`. If is followed by a condition contained in parenthesis. The body of the if statement is followed by endif, as follows:

```
if (condition)
      @ body of if
endif
```

6.2 For Loop

A for loop starts with `for` and ends in `endfor`. For is followed by three expressions enclosed in parenthesis and separated by semicolons. The first expression is evaluated before the body of the for loop. The body of the loop is executed whenever the second expression evaluates to true. The last expression is evaluated after each iteration of the loop. The format of the for loop is as follows:

```
for (expr; expr; expr)
     @ body of loop
endfor
```

6.3 Return Statements

All functions return a data type using the `rtn` keyword. If the keyword is not included the function will return `undefined.`

```
fn identifier(parameter list)
     @ body of function
     @ optional return statement
     rtn type
endfn
```

7. TYPE CONVERSIONS

The following functions take one argument of the type they are converting from and return a value of the type they are converting to:

`stf`	string to float
`itf`	int to float
`fti`	float to int
`sti`	string to int
`its`	int to string
`fts`	float to string

8. OUTPUT

Once compiled, each program will output both an HTML and a CSS file.

Example:
`http://www.columbia.edu/~iaw2105/plt/`