**Program Structure and Patterns**

**General Program Structure**

All programs must contain three sections: a begin section, a pattern section, and an end section. Programs must contain these in the order given and may not have more than one begin or end section. Following are a syntactic outline of such a structure and descriptions of each section.

Syntactic Outline:

begin{

}

[/\*pattern\*/]{

}

end{

}

Begin Section: executed first. Functions and variables may be declared in this section and defined and made available to other sections. In other words, variables and functions defined in this section are visible in the pattern and end sections. The section may be empty and all normal syntax previously defined should be followed.

Pattern Section: executed in order that they are defined, this section has one or more pattern blocks that begin with a CSS or Regex expression in brackets and are followed by an optional-empty action. The patterns must align with the syntax provided in the following pattern sections. The action will be executed after all instances of such patterns have been found in the given file. The action block should contain normal syntax previously defined and all variables defined will be visible in the end section. Functions may not be defined in this section nor can patterns be nested. Relevant information returned by the pattern match can be accessed using the *this* data structure. See below for further information on access to this structure.

*this*: a table of relevant objects returned by the pattern. Objects may be of type string, int, double, or table and can be accessed using the index operator this[i]. Moreover the this data structure has the following standard functions:

.first\_element(): returns an object representing the first element

.length(): returns the length of data structure (i.e. the number of objects that match the provided pattern)

.last\_element(): returns an object representing the last element

End Section: executed last after all pattern sections have been executed, this section may have include new functions and variables as well as references to previously defined (in begin or pattern sections) functions and variables. New functions and variable only are visible in the end section and normal syntax previously defined should be implemented.

**Pattern**

Patterns utilize valid Regex or CSS selector syntax and are defined with special offset characters within brackets before an action block in the pattern section of the program. Please see the following sections for a discussion on what a comprises a valid CSS selector and Regex expression.

CSS Selector:

HAWK implements a limited syntax of the standard W3 definition of CSS selectors. A CSS selector is used to find a particular element in an HTML document using the handles and operations defined below (please see the CSS and HTML language reference manuals for more). Moreover, a CSS selector pattern must be offset with *@* symbols on both sides and be contained within a bracket before an action section.

[@…@]

CSS selector operations may be combined, but must follow the law of the descendent, which states that operations following other operations must operate on elements that are children (elements that are contained by parent elements) of the previously found parent element. Valid CSS selector syntax and operations are as follows (all operations adhere to the law of the descendent):

\* : selects all elements

#id: selects all elements with an id attribute that matches the provided string

Example: #first-name will select all elements with attribute id=“first-name”

.class: selects all elements with a class attribute that matches the provided string

Example: *.first-name* will select all elements with attribute *class=“first-name”*

element: selects all elements that have the provided tag name. Please see the HTML language reference manual for a complete list of valid HTML element tags.

Example: *p* will select for all paragraph elements

element1 element2: law of the descendent. This pattern will select for all *element2’*s that are child elements of *element1*

Example: div #first-name will select for all elements with id attribute, id=“first-name,” that are children of div elements

element1 > element2: strict law of the descendent. This pattern will select for the elements that are direction children of the parent element. In other words, these children cannot be grandchildren (elements nested within other elements)

element1 + element2: selects the immediate child (the one child that is directly below the parent), *element2*, of the parent element, *element1*.

element1 ~ element2: selects every element, *element2*, that is preceded by *element1*

[attribute]: selects all elements with the given attribute.

Example: [title] will select all elements with a title attribute

[attribute op value]: selects all elements that have an attribute with a value that evaluates the expression to true.

[attribute = value]: selects elements that have attribute value equal to provided value.

[attribute ~= value]: selects elements that have an attribute value that contains provided value.

[attribute |= value]: selects elements that have an attribute value that begins with provided value.

[attribute $= value]: selects elements that have an attribute value that ends with provided value.

[attribute \*= value]: selects elements that have an attribute value that contains provided value.

Please see CSS and HTML language reference manuals for additional explanation of each selector pattern.

Regex Patterns:

HAWK implements a limited version of the standard regex expression syntax of the AWK language. A regex expression is used to find a particular pattern in an text document using the operations defined below (please see the POSIX and AWK language reference manuals for more). Moreover, a regex expression pattern must be offset with */* symbols on both sides and be contained within a bracket before an action section.

[/…/]

Regex expression operations may be combined and standard regex expression operator precedence will be assumed (see POSIX reference manual and note the order of the operators below) or order may be defined using parentheses. Below are a description of the operators implemented:

‘c’ : represents a single character.

\_ : represents any character.

eof: represents the end of file character.

“string” : represents a literal string of characters.

‘a’ - ‘z’: represents a range of characters. Must be contained within a pair of brackets.

[‘b’ ‘c’] : evaluates to true if current character matches any character provided within brackets.

^ : compliments a set of characters.

(pattern) : evaluates pattern inside parentheses before patterns outsides. Parentheses suggest an order of evaluation.

pattern \* : represents the kleene closure of a pattern with zero or more of the pattern present.

pattern + : represents the kleene closure of a pattern with one or more of the pattern present.

pattern ? : represents a pattern that is optional.

pattern1 pattern2: represents a pattern followed by a pattern.

pattern1 | pattern2: represents either pattern1 or pattern2

Please see the POSIX and AWK language reference manuals for additional explanation of each regex expression operator.