Michael Langford (mal2134)
Khai Vuong (kqv1)
Javier Coca (jc2658)
George Sirois (gts2006)

Programming Language Project Proposal
COMS W4115
September 27th, 2005

Our group has decided to implement a simple web-surfing language to help automate the web-browsing experience. Potentially, we will also build in features for form filling, but the primary mission for the language is to allow the user to write a quick script to go to a web page and follow any links or download any files.

Syntactically, we have decided to keep it as much like C as possible, because the syntax of C is well known and has proven to be adequate. Modularity will be achieved via functions, and curly brackets (`{ }`) will allow the programmer to group together multiple statements within the same scope. The `;` symbol will mark the end of a statement.

The primitive data types will include the `int` and `string` types. These are identical to their counterparts in most other languages. In addition, our language will also introduce the `hyperlink` data type, which will contain a url and whatever text that the link contains. The `webfile` type will store a webpage that can be manipulated by the programmer via our language. We would also introduce a collection type, which would act as a group of `hyperlink` data types.

The operators implemented in our language will include the simple mathematical operators, i.e. `+`, `-`, `*`, `/`, and `%`. The = operator will be used for assignment. Whenever a `webfile` type is assigned, it is at that time that the webpage is downloaded in temporary storage. The `==`, `!=`, `<`, `<=`, `>`, and `>=` operators will be used for comparison. The `&&` and `||` operators will be used for the logical AND and OR, respectively. For link traversal, we will use C's pointer notation. For example, to point a webpage to a `hyperlink`, you could use the following statements: `page1 = *link1.` Likewise, the following statement would assign a link to a webpage: `link1 = &page1;`. Another operator that we would like to introduce would be the save operator (`->`). This operator would be applied from a `webfile` to some local file location. For example, the statement `page1 -> "C:\\downloaded";` would save to the `C:\downloaded` the contents of `page1`. The `+` and `–` operators could also be used with the `collection` type to add and remove a `link` from the `collection`. Finally, we would also like to include the `~~` and `!~` operators for comparisons between strings and regular expressions. The statement `(a ~~ b)`, where `a` is a string and `b` is a regular expression would return `1` if `b` matches `a` and `0` if `b` does not match `a`.

Our conditional statements will be limited to the `if(…)…else… statement`. Looping structures will include the `while(…)…` loop and a `foreach(… in …)…` loop.

Our language would also include some built-in functions to aid the programmer:

- `url(hyperlink)` would return the url of the given `hyperlink`
- `text(hyperlink)` would return the text of the given `hyperlink`
- `status(webfile)` would return the status of the http request of the given file (200, 404, etc.)

- `type(webfile)` would return the MIME type of the `webfile`
- `link(webfile, i)` would return the `i`th link on the `webfile` given.
- `link(webfile, regex)` would return the first link that matches the given regular expression on the `webfile`.
- `links(webfile)` would return a collection of all links on the given `webfile`.
- `images(webfile)` would return a collection of all images on the given webpage.
- `title(webfile)` would return the title of the given `webfile` as a `string`.
- `html(webfile)` would return the html source code of the given `webfile`.
- `text(webfile)` just returns the text shown on the given `webfile`.
- `print(string)` prints the given string to the console
- `indexOf(string1, string2, start, length)` returns the index of string2 in string1 in the given bounds
- `substring(string, start, length)` returns a substring from the given string within the given bounds

A sample program written in the above described language follows:

```
void main()
{
   hyperlink currentLink;
   webfile currentPage;

   currentLink = {"http://www.google.com", ""};
   currentPage = *currentLink;

   while((status(currentPage) == 200) && (type(currentPage) == "text/html"))
   {
      foreach(hyperlink image in images(currentPage))
      {
         *image -> "C:\\downloaded_files";
      }
      currentLink = link(currentPage, 0)
      currentPage = *currentLink;
   }

   print("Done!");
}
```

The `main()` function is just like you'd expect from C. A `link` (`currentLink`) and `webfile` (`currentPage`) are declared, and `currentLink` is pointed to Google's website. Then, while `currentLink` points to a web page, `currentPage` is assigned to that page, all of the images are saved to `C:\downloaded_files`, and then `currentLink` is pointed to the first link on `currentPage`.

Another, more complex, example would be as follows:

```
void main()
{
   hyperlink lnk = {"http://www.google.com", ""};
```

```
   // Visit google.com
   visit(lnk, "c:\\data", 0);

   print("Done!");
}

// The visit() func is a recursive function that saves a webpage and all
// of the images on it, if it matches the regular expression ".* Brin.*"
// Then, it recursively calls itself on every webpage that current webpage
// links to a depth of 3 from the original page, if the url of the link
// matches the regular expression ".*about.*" or the link containst he text
// "About Google".
void visit(hyperlink lnk, string dir, int depth)
{
   webfile page1 = *lnk;

   // Check if the page has been downloaded
   if(status(page1) != 200) || (type(page1) != "text/html"))
   {
      print("Download error!");
   }
   else
   {
      // Check if page1 contains text that matched ".* Brin.*"
      if (text(page1) ~~ ".* Brin.*")
      {
         // Save the webpage
         page1 -> dir;

         // Save every image on the webpage
         collection images1 = images(page1);
         foreach (hyperlink img1 in images1)
         {
            *img1 -> dir;
         }
      }

      // Increment the depth
      depth++;

      if (depth < 3)
      {
            // Follow each link whose url matches ".*about.* "
            // or whose text is "About Google"
            collection links1 = links(page1);
            foreach (hyperlink link in links1)
            {
               if ((url(link1) ~~ ".*about.*") ||
                   (text(link1) == "About Google"))
               {
                  visit(link1, dir, depth);
               }
            }
      }
   }
}
```