TAPE: A File Handling Language

Project Proposal

Tianhua Fang (tf2377)
Alexander Sato (as4628)
Priscilla Wang (pyw2102)
Edwin Chan (cc3919)

Programming Languages and Translators
COMSW 4115 Fall 2016

Sept 28, 2016

Introduction

Motivations

Understanding user data has become increasingly important. To improve their products and services, many companies analyze behaviors and interaction in user data. There are already several tools created for the purpose of data manipulation. For example, Excel is used for storing and retrieving numerical data and SQL is used to communicate with a database of data. However, one thing that is missing is an easy and universal way for users to perform file operations. Each language has their own way of opening, closing, reading, and writing a file, which can be confusing for program developers. Our team aims to develop a programming language that simplifies the process for file operations.

Description

TAPE is a language that allows users to read, write, and manipulate documents easily. By developing this language, we are responding to the shortcomings of the built-in Java functions which handle files. In particular, there are redundancies within existing Java functions which we believe can be consolidated to ease user interface. For example, existing Java functions allow users to read in a file by using buffer stream I/O, stream I/O, or channel I/O; this redundancy can confuse users unfamiliar with Java. Instead, we want to consolidate these functions into one "read" function in TAPE. Furthermore, if a programmer opens and changes a file in a Java program and does not close the file properly in the program, the file does not actually get modified. This is another problem that our team wants to resolve- we want to simplify the process of file manipulation. In short, TAPE is a language designed to rework existing deficiencies and optimize user ease.

Additionally, we want users of TAPE to be able to work with all languages. File manipulation is handled differently in various languages. C's input and output is drastically different from Javas. By using TAPE, the user can just allow it to translate between the two different languages. The user will have the option to personally define certain basic I/O behavior in a separate file. By defining these once, the user no longer has to worry about translation

issues. Afterall, when editing a file source, it shouldn't matter what language you do it in. Whether it's code, a database of numbers, or a story, data is data. How this is done differs slightly from platform to platform, and TAPE aims to simplify that process.

Language Features

Comments

Characters	Description
/* */	Comments

Data Types

Keyword	Description
`int`	32 bit integer
`float`	64 bit float
`char`	character
`word`	text value string literals type `word`
`bool`	boolean value
`void`	null value
`file`	file
`phrase`	string of characters ending with newline
`scope`	string that in a pair of {}, () scope
`after`	after index, pointer, or other keywords like sentence
`before`	before index, pointer, or other keywords like sentence

Built-In Functions

Type	Method and Description
`*file`	**`open`**`(file name)` Opens a file.
`void`	**`close`**`(file name)` Closes a file.
`phrase`	**`readline`**`(file name, int begin, int end)` Reads a file and stores the data in a buffer. Begin and end are optional parameters. If begin is given, it will start reading that many bytes into the file. If end is given, it will stop reading that many bytes into the file. Returns phrase with all newlines omitted and on inserted at the end.
`void`	**`write`**`(location)[inserted text]` Writes inserted text into a set location. Location must use before or after keyword, and then some buffer, either word, phrase, or a C-style string. When writing, the default behavior for before: newline entered after the inserted text. The default behavior for after: newline inserted first, then the inserted text. If the location does not exist, an exception is thrown.
`file`	**`merge`**`(file name1, file name2, string newfile, string path)` Returns a file with a merged version of two files called newfile. The path is an optional parameter. If it is not specified, it creates the file at the current directory. Otherwise, it creates the file at the specified location. If path doesn't exist, it throws an exception.
`file`	**`split`**`(location, string newfile)` Returns a file created from the original file that is split at the location of some word, phrase, or string literal. A new file is created with the second parameter as its name.
`*char`	**`scan`**`(string buffer, char *location)` Searches for `buffer` in file, starting at a given location. Location of 0 would start the search at the beginning of the file. Returns the address of the location of the first letter of the string if it is found. Otherwise, returns null.
`void`	**`rm`**`(file name)` Deletes the file.
`void`	**`delete`**`(word text)` Deletes `text` in the file.
`file`	**`copy`**`(file name)` Returns a copy of the file into the same path with "copy of" added to the beginning of the new file.
`int`	**`count`**`(word target)` Returns the number of times that `target` appears in `file`.

`void`	**`replace`**`(word search_keyword, word new_keyword)` Finds all instances of `search_keyword` and replace them with `new_keyword`.

Operators

+ - * / can be used for mathematical purposes. But in addition to the mathematical functions, TAPE allows these operations to do the following things:

Operation	Type	Description
+	`int, float, string`	+ can be used between two strings for concatenation. This operator will be able to accomplish smart promotions of data types. 1) When adding int and a float, the int will be upgraded. 2) When adding a string with an int, the int will be upgraded into a string.
+	`file`	+ merges two files. (ie. `file1 + file2` will return `file3`, which is a merged file of `file1` and `file2`)
*	*	* is a C-style pointer. When * is immediately preceded by a data type, the compiler will know that it is a pointer.
/	`file`	/ checks whether two files are different. Returns a file that indicates the similarities and differences between the two files.
==	`bool`	== checks whether two files are the the same. Returns `true` if the two files match character by character, returns `false` otherwise.

Loops and Conditionals

a. If-else conditions

```
if (<condition>) {
      <statements>
} else if {
      <statements>
} else {
      <statements>
}
```

b. For loop

```
for (int i = 0; i < 10; i++) {
```

```
        <statement>
    }
```

c. **While loop**

```
    int j = 10;
    while (i < j) {
        i++;
    }
```

Example Code

Combining Two Files

```
/*Program to combine two file with same programming language into a single file called combinedfile */
file original = open(C:/somepath.py);
file target = open(C:/somepath.py);
file result = merge(original, target, "newfile", "C:/user/..");
```

Merge and Move

```
/* let nginx log was saved as every 5 minute, we want to merge all of the log for Sep28th into a single log file, move the rest to a backup folder.  */
merge("log_20160928*", "\n" "C:/nginx/log..", "C://output/path/..");
move("log_20160928*", "C:/nginx/log...", "C:/logbak..." );
```

Print All the Location with the Name "class"

```
/* open the file, then count how many time the word appear then print out the location of the word in the file */
file target = open(C:/user/test.java);
A = target.count("class");
char *ptr=0;
for(int i=0;i<a; i++){
    ptr = scan(target, "class", ptr);
    print(&ptr);
}
```

Writing Text to a File

```
/* open the file first. Then insert the text*/
file target = open(C:/user/test.java);
word myWord = "hello";
write (after myWord)["world"];
```