Web Site Monitoring

Due at 11:59pm on Monday, 16 April 2018

1 Introduction

Each Web server routinely logs accesses from other Web servers and browsers. The log is a text file in which each line contains a date and a hostname. Each date is logged in the format `dd/mm/yyyy`. Each hostname ends with a 2-letter country code such as `uk` or `fr` (or a 3-letter code such as `com`) preceded by a dot/period/full-stop ('.'). The final token in a hostname is usually called the "top level domain", or TLD for short. The log might look like this:

```
05/11/1999 www.intel.com
12/12/1999 www.dcs.gla.ac.uk
05/11/2001 www.mit.edu
31/12/1999 www.cms.rgu.ac.uk
25/12/1999 www.informatik.tum.de
01/04/2000 www.wiley.uk
01/01/1999 www.fiat.it
14/02/2000 www.valentine.com
```

A new DHS[1] regulation requires that we track access by country, being able to demonstrate the percentage of accesses from each country over a given time period. Given the number of queries that will originate from `.com`, `.edu`, or other TLDs for which the country is not explicitly known, the politicians have allowed that tracking accesses by TLD is sufficient to satisfy the regulation. If the period of interest is 01/08/1999 to 31/07/2000, given the above log, the output from the program should look like this:

```
33.33 com
16.67 de
50.00 uk
```

Since the program is to execute on a Linux platform, there is no requirement that the summary statistics be output in any particular order, as we can pipe the output of the program into sort to yield the ordering desired.

2 Specification

Given a start date, an end date, and one or more log files, the program is to determine the percentage of access from each TLD during that period, outputting the final percentages on standard output, as shown above.

Hostnames, and therefore, top level domain names, are case-insensitive. Therefore, accesses by `X.Y.UK` and `a.b.uk` are both accesses from the same TLD.

[1] Department of Homeland Security

3 Design

In Canvas/Files/Projects/Project0/start.tgz, I am providing you with the source file for main(), and header files for two abstract data types – date.h and tldmap.h.[2]

3.1 date.h

```
#ifndef _DATE_H_INCLUDED_
#define _DATE_H_INCLUDED_

typedef struct date Date;
struct date {
    void *self;            /* instance-specific data */

/*
 * duplicate() creates a duplicate of `d'
 * returns pointer to new Date structure if successful,
 *         NULL if not (memory allocation failure)
 */
    const Date *(*duplicate)(const Date *d);

/*
 * compare() compares two dates, returning <0, 0, >0 if
 * date1<date2, date1==date2, date1>date2, respectively
 */
    int (*compare)(const Date *date1, const Date *date2);

/*
 * destroy() returns any storage associated with `d' to the system
 */
    void (*destroy)(const Date *d);
};

/*
 * Date_create() creates a Date structure from `datestr`
 * `datestr' is expected to be of the form "dd/mm/yyyy"
 * returns pointer to Date structure if successful,
 *         NULL if not (syntax error or malloc failure)
 */
const Date *Date_create(char *datestr);

#endif /* _DATE_H_INCLUDED_ */
```

The **`struct date`**, and the corresponding typedef **`Date`**, define a dispatch table structure for a date. You can only manipulate one of these structures using the defined methods.

The constructor for this ADT is **`Date_create()`**; it converts a **`datestring`** in the format "dd/mm/yyyy" to a **`Date`** structure. You will have to use **`malloc()`** to allocate this **`Date`** structure to return to the user.

The **`duplicate()`** method is known as a copy constructor; it duplicates the **`Date`** argument on the heap (using **`malloc()`**) and returns it to the user.

[2] Note that these header files follow the style of ADTs described in Chapter 5 of Canvas/Files/Miscellaneous/Intro-Linux-C-ADTs.pdf. You must familiarize yourself with the material in that chapter to be able to do this project.

The **`compare()`** method compares two **`Date`** structures, returning `<0, 0, >0` if `date1<date2`, `date1==date2`, `date1>date2`, respectively.

The **`destroy()`** method returns the heap storage associated with the **`Date`** structure.

tldmap.h

```
#ifndef _TLDMAP_H_INCLUDED_
#define _TLDMAP_H_INCLUDED_

#include "date.h"
#include "iterator.h"

typedef struct tldmap TLDMap;
typedef struct tldnode TLDNode;

/*
 * TLDMap_create generates a map structure for storing counts against
 * top level domains (TLDs)
 *
 * creates an empty TLDMap
 * returns a pointer to the dispatch table if successful, NULL if not
 */
const TLDMap *TLDMap_create(void);

struct tldmap {
    void *self;               /* instance specific data */
/*
 * destroy() destroys the map structure in `tld'
 *
 * all allocated storage associated with the map is returned to the heap
 */
    void (*destroy)(const TLDMap *tld);

/*
 * insert() inserts the key-value pair (theTLD, v) into the map;
 * returns 1 if the pair was added, returns 0 if there already exists an
 * entry corresponding to theTLD
 */
    int (*insert)(const TLDMap *tld, char *theTLD, long v);

/*
 * reassign() replaces the current value corresponding to theTLD in the
 * map with v.
 * returns 1 if the value was replaced, returns 0 if no such key exists
 */
    int (*reassign)(const TLDMap *tld, char *theTLD, long v);

/*
 * lookup() returns the current value associated with theTLD in *v
 * returns 1 if the lookup was successful, returns 0 if no such key exists
 */
    int (*lookup)(const TLDMap *tld, char *theTLD, long *v);

/*
 * itCreate() creates an iterator over the map
 * returns the iterator if successful, NULL if unsuccessful
 */
    const Iterator *(*itCreate)(const TLDMap *tld);
};

/*
 * TLDNode_tldname returns the tld associated with the TLDNode
```

```
 */
char *TLDNode_tldname(TLDNode *node);

/*
 * TLDNode_count returns the value currently associated with that TLD
 */
long TLDNode_count(TLDNode *node);

#endif /* _DATE_H_INCLUDED_ */
```

TLDMap and **TLDNode** are opaque data structures that you can only manipulate using methods and functions in this class.

TLDMap_create() creates a **TLDMap** dispatch table which can be used to store the counts of log entries against TLD strings.

The **destroy()** method returns the heap storage associated with the **TLDMap** structure.

The **insert()** method inserts the key-value pair (theTLD, v) into the map.

The **reassign()** method replaces the current value associated with theTLD in the map with v.

The **lookup()** method returns the current value associated with the TLD in the map in *v.

The **itCreate()** method creates an iterator over the map.

The **TLDNode_tldname()** function returns the string for the TLD represented by a node obtained from the iterator.

The **TLDNode_count()** function returns the number of log entries that were counted for that TLD.

3.2 tldmonitor.c

```
#include "date.h"
#include "tldmap.h"
#include <stdio.h>
#include <string.h>
#include <ctype.h>

#define USAGE "usage: %s begin_datestamp end_datestamp [file] ...\n"
#define ERROR_EXIT 1

static void fold(char *s) {
    int i;

    for (i = 0; s[i] != '\0'; i++) {
        s[i] = tolower((int)s[i]);
    }
}

static char *extract(char *hostname) {
    char *p = strrchr(hostname, '.');
    if (p == NULL)
        p = hostname;
    else
        p++;
    return p;
}

static void process(FILE *fd, const Date *begin, const Date *end,
                    const TLDMap *tld, long *count) {
    char bf[1024], sbf[1024];
    const Date *d;
    char *theTLD;
    while (fgets(bf, sizeof(bf), fd) != NULL) {
        char *q, *p = strchr(bf, ' ');
    if (p == NULL) {
        fprintf(stderr, "Illegal input line: %s", bf);
        return;
    }
    strcpy(sbf, bf);
    *p++ = '\0';
    while (*p == ' ')
        p++;
    q = strchr(p, '\n');
    if (q == NULL) {
        fprintf(stderr, "Illegal input line: %s", sbf);
         return;
    }
    *q = '\0';
    fold(p);
    theTLD = extract(p);
    d = Date_create(bf);
    if (!(d->compare(d, end) > 0 || d->compare(d, begin) < 0)) {
        long value;
        *count += 1;
        if (! tld->lookup(tld, theTLD, &value))
            tld->insert(tld, theTLD, 1L);
        else
            tld->reassign(tld, theTLD, value+1);
    }
    d->destroy(d);
}
```

```
int main(int argc, char *argv[]) {
    const Date *begin, *end;
    int i;
    FILE *fd;
    const TLDMap *tld = NULL;
    const Iterator *it = NULL;
    TLDNode *n;
    double total;
    long count = 0;

    if (argc < 3) {
        fprintf(stderr, USAGE, argv[0]);
        return ERROR_EXIT;
    }
    begin = Date_create(argv[1]);
    if (begin == NULL) {
        fprintf(stderr, "Error processing begin date: %s\n", argv[1]);
        return ERROR_EXIT;
    }
    end = Date_create(argv[2]);
    if (end == NULL) {
        fprintf(stderr, "Error processing end date: %s\n", argv[2]);
        goto error;
    }
    if (begin->compare(begin, end) > 0) {
        fprintf(stderr, "%s > %s\n", argv[1], argv[2]);
    goto error;
    }
    tld = TLDMap_create();
    if (tld == NULL) {
        fprintf(stderr, "Unable to create TLD list\n");
        goto error;
    }
    if (argc == 3)
        process(stdin, begin, end, tld, &count);
    else {
        for (i = 3; i < argc; i++) {
            if (strcmp(argv[i], "-") == 0)
                fd = stdin;
            else
                fd = fopen(argv[i], "r");
            if (fd == NULL) {
                fprintf(stderr, "Unable to open %s\n", argv[i]);
                continue;
            }
            process(fd, begin, end, tld, &count);
            if (fd != stdin)
                fclose(fd);
        }
    }
    total = (double)count;
    it = tld->itCreate(tld);
    if (it == NULL) {
        fprintf(stderr, "Unable to create iterator\n");
        goto error;
    }
    while (it->hasNext(it)) {
        (void) it->next(it, (void **)&n);
        printf("%6.2f %s\n", 100.0 * (double)TLDNode_count(n)/total,
               TLDNode_tldname(n));
    }
    it->destroy(it);
    tld->destroy(tld);
    begin->destroy(begin);
```

```
    end->destroy(end);
    return 0;
error:
    if (it != NULL)    it->destroy(it);
    if (tld != NULL)   tld->destroy(tld);
    if (end != NULL)   end->destroy(end);
    if (begin != NULL) begin->destroy(begin);
    return ERROR_EXIT;
}
```

The main program is invoked as

./tldmonitor begin_date end_date [file] ...

If no file is present in the arguments, stdin will be processed. Additionally, if a filename is the string "-", the program will process stdin at that point.

The mainline functionality of tldmonitor.c consists of the following pseudocode:

```
process the arguments
create a TLD map
if no file args are provided
  process stdin
else for each file in the argument list
  open the file
  process the file
  close the file
create an iterator
while there is another entry in the iterator
  print out the percentage associated with that TLD
destroy the iterator
destroy the TLD map
destroy the Date structures
```

Static functions (**`process, fold, and extract`**) are provided to process the log entries in a particular log file.

4 Implementation

You are to implement date.c and tldmap.c. The implementations must match the function prototypes in the headers listed in section 3 above.

You ***must*** implement tldmap.c as a hash table or a binary search tree.

Your marks for each source file will depend upon its design, implementation, and its ability to perform correctly when executed. tldmonitor will be tested against some VERY LARGE log files to see if you have correctly implemented your map. N.B. If your code does not compile, you will not receive **any** marks for that file. A complete mark scheme is appended to the handout.

Note that you will be heavily penalized if your program leaks heap memory. After you have a working version of the program, you need to test it using "valgrind" to make sure it does not leak heap memory. If "valgrind" indicates ***any*** problems with your code's use

of heap memory, it is usually an indication that you are doing something very wrong that will bite you eventually; you ***must*** fix your code to remove all such problem reports.

In addition to tldmonitor.c, date.h and tldmap.h, I have also provided linux32/tldmapll.o and linux64/tldmapll.o, which are 32-bit and 64-bit versions of a linked list implementation of tldmap.c, on Canvas. This will permit you to test your implementation of date.c against a working, albeit inefficient, implementation of tldmap. I have also provided sample input files and the output that your program should generate for that input file.[3]

5 Submission[4]

You will submit your solutions electronically by uploading a gzipped tar archive via Canvas.

Your TGZ archive should be named "<duckid>-project0.tgz", where "<duckid>" is your duckid. It should contain your "date.c", your "tldmap.c", and a document named "report.pdf" or "report.txt", describing the state of your solution and documenting anything of which we should be aware when marking your submission. Do not include any other files in the archive; in particular, this means that ***UNDER NO CIRCUMSTANCES*** should you change date.h, tldmap.h, or tldmonitor.c. Your submission will be tested against the issued versions of these files; if you change them, then your code will not work correctly and you will lose marks.

These files should be contained in a folder named "<duckid>". Thus, if you upload "jsventek-project0.tgz", then we should see the following when we execute the following command:

```
% tar –ztvf jsventek-project0.tgz
-rw-rw-r-- jsventek/None      3670 2015-03-30 16:30 jsventek/date.c
-rw-rw-r-- jsventek/None      5125 2015-03-30 16:37 jsventek/tldmap.c
-rw-rw-r-- jsventek/None    629454 2015-03-30 16:30 jsventek/report.pdf
```

Each of your source files must start with an "authorship statement", contained in C comments, as follows:

- state your name, your duckid, and the title of the assignment (CIS 415 Project 0)
- state either "This is my own work." or "This is my own work except that …", as appropriate.

We will be compiling your code and testing against an unseen set of log files. We will also be checking for collusion; better to turn in an incomplete solution that is your own than a copy of someone else's work. We have very good tools for detecting collusion.

[3] The following commands should yield ***NO*** output if you have implemented your ADTs correctly:
% ./tldmonitor 01/01/2000 01/09/2013 <small.txt | sort -n | diff - small.out
% ./tldmonitor 01/01/2000 01/09/2013 <large.txt | sort -n | diff - large.out

[4] ***A 20% penalty will be assessed if you do not follow these submission instructions. Section 6 describes how to follow these directions for those that are unsure.***

6 How to create the correct archive to submit

- First, determine your duckid (your uoregon.edu email ***without*** the "@uoregon.edu". In the following, I refer to it as *duckid*.
- Assume that your current working directory is the source directory for project 0 (it does not matter what you call that directory).
- Create a subdirectory named *duckid* in the current working directory by executing the following command in the shell:

  ```
  $ mkdir duckid
  ```

- Create a text file named manifest with the following lines in it

  ```
  duckid/date.c
  duckid/tldmap.c
  duckid/report.pdf
  ```

  (of course, if you are submitting `report.txt`, you should replace the last line with `duckid/report.txt`)
- Now execute the following lines in the shell:

  ```
  $ cp date.c tldmap.c report.pdf duckid
  $# the previous command makes copies of the files in duckid
  $ tar -zcvf duckid-project0.tgz $(cat manifest)
  $ tar -ztvf duckid-project0.tgz
  ```

The last command above should generate a listing that looks like the following:

```
-rw-rw-r-- duckid/<group>   3670 2015-03-30 16:30 duckid/date.c
-rw-rw-r-- duckid/<group>   5125 2015-03-30 16:37 duckid/tldmap.c
-rw-rw-r-- duckid/<group> 629454 2015-03-30 16:30 duckid/report.pdf
```

Marking Scheme for CIS 415, Project 0

Your submission will be marked on a 100 point scale. Substantial emphasis is placed upon **WORKING** submissions, and you will note that a large fraction of the points are reserved for this aspect. It is to your advantage to ensure that whatever you submit compiles, links, and runs correctly. The information returned to you will indicate the number of points awarded for the submission.

You must be sure that your code works correctly on the virtual machine under VirtualBox, regardless of which platform you use for development and testing. Leave enough time in your development to fully test on the virtual machine before submission.

The marking scheme is as follows:

Points	Description
10	Your report – honestly describes the state of your submission
20	Date ADT 6 for workable solution (looks like it should work) 2 if it successfully compiles 2 if it compiles with no warnings 6 if it works correctly (when tested with an unseen driver program) 4 if there are no memory leaks
70	TLDMap ADT 24 for workable solution (looks like it should work) 2 if it successfully compiles 2 if it compiles with no warnings 2 if it successfully links with tldmonitor 2 if it links with no warnings 18 if it works correctly with small.txt and large.txt 6 if it works correctly with 10,000 entry unseen log file 6 if it works correctly with 1,000,000 entry unseen log file 6 if there are no memory leaks

Several things should be noted about the marking schemes:

- Your report needs to be honest. Stating that everything works and then finding that it won't even compile is offensive. The 10 points associated with the report are probably the easiest 10 points you will ever earn as long as you are honest.
- If your solution does not look workable, then the points associated with successful compilation and lack of compilation errors are **not** available to you. This prevents you from handing in a stub implementation for each of the methods in each ADT and receiving points because they compile without errors, but do nothing.
- The points associated with "workable solution" are the maximum number of points that can be awarded. If it is deemed that only part of the solution looks workable, then you will be awarded a portion of the points in that category.