University of Glasgow | School of Computing Science

Assessed Coursework

Course Name	**Systems Programming**			
Coursework Number	**1a: Website Monitoring**			
Deadline	**Time:**	**16:30**	**Date:**	**5th November 2020**
% Contribution to final course mark	**8%**			
Solo or Group ✓	**Solo**	✓	**Group**	
Anticipated Hours	**20**			
Submission Instructions	**Described in section 5 below.**			
Please Note: This Coursework cannot be Re-Done				

Web Site Monitoring

1 Requirements

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:

```
05/11/2018 www.intel.com
12/12/2018 www.dcs.gla.ac.uk
05/11/2019 www.mit.edu
31/12/2018 www.cms.rgu.ac.uk
25/12/2017 www.informatik.tum.de
01/04/2018 www.wiley.co.uk
01/01/2019 www.fiat.it
14/02/2018 www.valentine.com
```

A new EU regulation requires that we track access by country, being able to demonstrate the percentage of accesses from each country over a given time period. The politicians have allowed that tracking accesses by TLD is sufficient to satisfy the regulation. If the period of interest is 01/12/2017 to 31/12/2018, 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.

3 Design

You are provided with the source file for main() tldmonitor.c, and header files for two abstract data types – date.h and tldlist.h.

3.1 date.h

```
#ifndef _DATE_H_INCLUDED_
#define _DATE_H_INCLUDED_

typedef struct date Date;
```

```
/*
 * 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)
 */
Date *date_create(char *datestr);

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

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

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

#endif /* _DATE_H_INCLUDED_ */
```

The `struct date`, and the corresponding typedef `Date`, define an opaque data structure for a date.

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.

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

`date_compare()` compares two `Date` structures, returning `<0, 0, >0` if `date1<date2`, `date1==date2`, `date1>date2`, respectively.

`date_destroy()` returns the heap storage associated with the `Date` structure.

3.2 tldlist.h

```
#ifndef _TLDLIST_H_INCLUDED_
#define _TLDLIST_H_INCLUDED_

#include "date.h"

typedef struct tldlist TLDList;
typedef struct tldnode TLDNode;
typedef struct tlditerator TLDIterator;

/*
 * tldlist_create generates a list structure for storing counts against
 * top level domains (TLDs)
 *
 * creates a TLDList that is constrained to the `begin' and `end' Date's
 * returns a pointer to the list if successful, NULL if not
 */
TLDList *tldlist_create(Date *begin, Date *end);

/*
 * tldlist_destroy destroys the list structure in `tld'
 *
 * all heap allocated storage associated with the list is returned to the heap
 */
void tldlist_destroy(TLDList *tld);

/*
 * tldlist_add adds the TLD contained in `hostname' to the tldlist if
 * `d' falls in the begin and end dates associated with the list;
 * returns 1 if the entry was counted, 0 if not
 */
int tldlist_add(TLDList *tld, char *hostname, Date *d);

/*
 * tldlist_count returns the number of successful tldlist_add() calls since
 * the creation of the TLDList
 */
long tldlist_count(TLDList *tld);

/*
 * tldlist_iter_create creates an iterator over the TLDList; returns a pointer
 * to the iterator if successful, NULL if not
 */
TLDIterator *tldlist_iter_create(TLDList *tld);

/*
 * tldlist_iter_next returns the next element in the list; returns a pointer
 * to the TLDNode if successful, NULL if no more elements to return
 */
TLDNode *tldlist_iter_next(TLDIterator *iter);

/*
 * tldlist_iter_destroy destroys the iterator specified by `iter'
 */
void tldlist_iter_destroy(TLDIterator *iter);

/*
 * tldnode_tldname returns the tld associated with the TLDNode
 */
char *tldnode_tldname(TLDNode *node);

/*
```

```
 * tldnode_count returns the number of times that a log entry for the
 * corresponding tld was added to the list
 */
long tldnode_count(TLDNode *node);

#endif /* _TLDLIST_H_INCLUDED_ */
```

TLDList, TLDIterator, and TLDNode are opaque data structures that you can only manipulate using methods in this class.

tldlist_create() creates a TLDList which can be used to store the counts of log entries against TLD strings; the begin and end date arguments enable filtering of added entries to be in the preferred date range.

tldlist_destroy() returns the heap storage associated with the TLDList structure.

tldlist_add() will count the log entry if the associated date is within the preferred data range.

tldlist_count() returns the number of log entries that have been counted in the list.

tldlist_iter_create() creates an iterator to enable you to iterate over the entries, independent of the data structure chosen for representing the list.

tldlist_iter_next() returns the next TLDNode in the list, or NULL if there are no more entries.

tldnode_tldname() returns the string for the TLD represented by this node.

tldnode_count() returns the number of log entries that were counted for that TLD.

3.3 *tldmonitor.c*

```
#include "date.h"
#include "tldlist.h"
#include <stdio.h>
#include <string.h>

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

static void process(FILE *fd, TLDList *tld) {
    char bf[1024], sbf[1024];
    Date *d;
    while (fgets(bf, sizeof(bf), fd) != NULL) {
        char *q, *p = strchr(bf, ' ');
      if (! p) {
            fprintf(stderr, "Illegal input line: %s", bf);
          return;
        }
      strcpy(sbf, bf);
      *p++ = '\0';
      while (*p == ' ')
            p++;
      q = strchr(p, '\n');
      if (! q) {
            fprintf(stderr, "Illegal input line: %s", sbf);
          return;
        }
      *q = '\0';
      d = date_create(bf);
      (void) tldlist_add(tld, p, d);
      date_destroy(d);
    }
}

int main(int argc, char *argv[]) {
    Date *begin, *end;
    int i;
    FILE *fd;
    TLDList *tld;
    TLDIterator *it;
    TLDNode *n;
    double total;

    if (argc < 3) {
        fprintf(stderr, USAGE, argv[0]);
        return -1;
    }
    begin = date_create(argv[1]);
    if (! begin) {
        fprintf(stderr, USAGE, argv[0]);
        return -1;
    }
    end = date_create(argv[2]);
    if (! end) {
        fprintf(stderr, USAGE, argv[0]);
        return -1;
    }
    tld = tldlist_create(begin, end);
    if (! tld) {
        fprintf(stderr, "Unable to create TLD list\n");
        return -2;
    }
    if (argc == 3)
        process(stdin, tld);
    else {
        for (i = 3; i < argc; i++) {
            if (strcmp(argv[i], "-") == 0)
                fd = stdin;
            else
                fd = fopen(argv[i], "r");
            if (! fd) {
                fprintf(stderr, "Unable to open %s\n", argv[i]);
                continue;
            }
            process(fd, tld);
            if (fd != stdin)
                fclose(fd);
        }
    }
    total = (double)tldlist_count(tld);
    it = tldlist_iter_create(tld);
    if (! it) {
        fprintf(stderr, "Unable to create iterator\n");
        return -2;
    }
    while ((n = tldlist_iter_next(it))) {
        printf("%6.2f %s\n", 100.0 * (double)tldnode_count(n)/total, tldnode_tldname(n));
    }
    tldlist_iter_destroy(it);
    tldlist_destroy(tld);
    date_destroy(begin);
    date_destroy(end);
    return 0;
}
```

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 list
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 Date structures
destroy the TLDList
```

A static function (process) is provided to process all of the log entries in a particular log file.

4 Implementation

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

Your code must work correctly with the clang compiler (without using any --std flags) on the School stlinux02..08 systems, regardless of which platform you use for development and testing. Leave enough time in your development to fully test on the lab machines before submission.

You have two options for your implementation of tldlist.c: 1) you can use a binary search tree (BST) as the basis of your list; in this case, you can earn at most 70% of the marks for the assignment, or 2) you can use a balanced binary search tree (AVL), based upon the Adelson-Velskii and Landis algorithm; in this case, you can earn all 100% of the marks for the assignment.

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

In addition to tldmonitor.c, date.h and tldlist.h, you are provided with tldlist.o, which is a linked list implementation of tldlist.c. *This will permit you to test your implementation of date.c against a working, albeit inefficient, implementation of tldlist.* You are also provided sample input files and the output that your program should generate for that input file.[1]

5 Submission

You will submit your solutions electronically by submitting the following files on Aropa2: https://aropa2.gla.ac.uk/aropa/

- date.c
- tldlist.c
- report.pdf – A brief report in PDF outlining the state of your solution, and identifying any bugs or limitations you are aware of. It is important that the report is accurate. For example it is offensive to report that everything works when it won't even compile.

If you have produced a binary search tree implementation, then submit to Exercise 1a Website Monitoring (BST). If you have produced a balanced binary search tree (AVL) implementation, then submit to Exercise 1a Website Monitoring (AVL). ***Only submit to one or the other!***

Each of your source files must start with an "authorship statement" as follows:

- state your name, your login, and the title of the assignment (SP Exercise 1a)

[1] The following commands should yield ***NO*** output if you have implemented your ADTs correctly:
```
% ./tldmonitor 01/01/2017 01/09/2020 <small.txt | sort -n | diff - small.out
% ./tldmonitor 01/01/2017 01/09/2020 <large.txt | sort -n | diff - large.out
```

- state either "This is my own work as defined in the Academic Ethics agreement I have signed." or "This is my own work except that …", as appropriate.

You must complete an "Own Work" form via https://webapps.dcs.gla.ac.uk/ETHICS.

Assignments will be checked for collusion; better to turn in an incomplete solution that is your own than a copy of someone else's work. There are very good tools for detecting software plagiarism, e.g. JPLAG or MOSS.

6 Marking Scheme

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.

You can choose to turn in two forms of tldlist:

- if it is implemented using a binary search tree (BST), only 70 of the 100 total points are available to you
- if it is implemented using an Adelson-Velskii Landis (AVL) tree, all 100 total points are available to you.

The BST marking scheme is as follows:

Points	Description
10	Your report – accurately & 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
40	TLDList ADT 12 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 6 if it works correctly with small.txt and large.txt 4 if it works correctly with 10,000 entry unseen log file 4 if it works correctly with 1,000,000 entry unseen log file 6 if there are no memory leaks

The AVL marking scheme is as follows:

Points	Description
10	Your report – accurately & honestly describes the state of your submission
20	<u>Date ADT</u> 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	<u>TLDList ADT</u> 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 4 if it works correctly with 10,000 entry unseen log file 4 if it works correctly with 1,000,000 entry unseen log file 6 if it works correctly with sorted 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 accurate and 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.