Department of Computing and Information Systems

COMP10002 Foundations of Algorithms Semester 2, 2016 Assignment 1

Learning Outcomes

In this project you will demonstrate your understanding of arrays, strings, and functions. You may also use `typedef`s and `struct`s if you wish – and may find the program easier to assemble if you do – but you are not required to use them in order to obtain full marks. Nor do you need to make any use of `malloc()` in this project.

TSV Files

The simple *tab separated format* is a common and convenient way of storing structured data in a text file. The first line of the file stores a set of c "column header" entries all represented as text strings that may contain alphanumeric, blank characters and punctuation characters, but no tab (`'\t'`) or newline (`'\n'`) characters, stored with $c-1$ tab characters between the c entries to denote the breaks between the strings, and with a final newline character at the end of the line. A set of r following rows each store c data items, represented in the same format. For example, consider the data file `test0.tsv`, with tabs and newline characters shown explicitly:

```
Event\tGender\tCountry\tMedal\n
Swimming\tWomens\tNew Zealand\tfirst: gold\n
Swimming\tWomens\tChina\tsecond: silver\n
Swimming\tWomens\tIndonesia\tthird: bronze\n
Cycling\tWomens\tChina\tfirst: gold\n
Cycling\tWomens\tNew Zealand\tsecond: silver\n
Cycling\tWomens\tNew Zealand\tthird: bronze\n
EOF
```

This test file has $c = 4$ columns, headed "`Event`", "`Gender`", "`Country`", and "`Medal`" respectively; and contains $r = 6$ data rows that show (for example) that the country "`New Zealand`" is associated with the medals "`second: silver`" and "`third: bronze`" in the sport "`Cycling`" in connection with the gender "`Womens`". Note that r and c are not stored as part of the file, they are implicit in the arrangement of the tab characters, three per line, and the arrangement of newline characters, one per line. Note also that `EOF` is not actually a character in the file, and it is included here simply to show you that the last actual input line is also ended by a newline (see the FAQ page for more on newlines).

Many different kinds of data can be represented in tsv files including scientific data (for example, weather data, in columns headed date, time, location, temperature, and rainfall); financial data (for example, share transactions with a stock code, a purchase price, a volume, and a sale price); student data (columns for family name, given name, student number, year, semester, subject code, final mark); and so on. In this project you will write functions that manipulate input that is provided in tsv format. The simple test file `test0.tsv` shown above is available on the LMS page, together with some other larger files. Further data will be used to test your program once submissions have closed.

Stage 1 – Reading and Printing (marks up to 8/15)

Write a program that reads a tsv-structured input stream from `stdin` and builds a corresponding internal data structure using a 2d array of strings. You should assume that at most 1,000 input lines will be presented, that each input line contains at most 30 columns of information, and that each entry contains at most 50 characters. As evidence of the correctness of this stage of your program, it should report the number of rows and columns in the input, and list the column headings and the corresponding values from the final input row of the file, using *exactly* the output format that is shown here:

```
mac: ./ass1-soln < test0.tsv
Stage 1 Output
input tsv data has 6 rows and 4 columns
row 6 is:
   1: Event      Cycling
   2: Gender     Womens
   3: Country    New Zealand
   4: Medal      third: bronze
```

Other input files and example output files are linked from the LMS page. Note that the columns are labeled from 1, and that the data rows are also labeled starting with 1.

You may assume throughout that all input files you will be provided with will be "correct", with all rows having the same number of entries; the number of rows and columns being within the specified bounds; the length of each entry being within the specified bound; and with no additional (or missing) newline or tab characters. Your program must read from `stdin` and write to `stdout`.

Distractions to steer clear of: it will be *much* easier to use `getchar()` (or the `mygetchar()` function provided on the FAQ page) than try and get `scanf()` to work; you do *not* need to use `malloc()` or the other similar functions from Chapter 10; you should *not* write your program in such a way that it relies on the test data being in a named location or file; you may *not* use any of the file operations described in Chapter 11 (except for `fflush(stdout)` when debugging, see the FAQ page); and should *not* print a prompt of any sort.

Be sure that you also read the information on the FAQ page about PC and Mac newline differences and the way that different types of editors might create them. These difference have the potential to be very frustrating if you don't allow for them. All of test files that are provided via the LMS will use the PC/DOS convention, that is, with combined `'\r'+'\n'` to indicate newlines, and should work correctly on both PCs and Macs if you use the `mygetchar()` function from the FAQ page.

Stage 2 – Sorting (marks up to 12/15)

Now add further functionality to your program so that it accesses a sequence of integers from the command-line, each of them a value between 1 and c, indicating which column(s) should be used as sort keys. For example, for `test0.txt`, if columns 3 and then 1 are specified, the r data rows in your internal format are to be reordered so that they are sorted first by `Country` (as a string, using `strcmp()`), and then where the `Country` fields are equal, using `Event` as a secondary key. Where two rows have the same values in the selected columns, they should be retained in the same order as they were originally; that is, your sort should be *stable*. The header row should not be sorted. For `test0.txt`, the sorted form for arguments "3 1" should be:

```
Event          Gender         Country        Medal
Cycling        Womens         China          first: gold
Swimming       Womens         China          second: silver
Swimming       Womens         Indonesia      third: bronze
Cycling        Womens         New Zealand    second: silver
Cycling        Womens         New Zealand    third: bronze
Swimming       Womens         New Zealand    first: gold
```

That listing shows the *ordering* for this particular pair of sort keys. For brevity, the demonstration of the correctness of your Stage 2 program will be judged based on the first data row, the middle data row (that is, row $\lceil r/2 \rceil$), and the last data row. The following is what your program should produce when it is submitted:

```
mac: ./ass1-soln 3 1 < test0.tsv
<<<Stage 1 output here, see above>>>
Stage 2 Output
row 1 is:
   1: Event      Cycling
   2: Gender     Womens
   3: Country    China
   4: Medal      first: gold
row 3 is:
   1: Event      Swimming
   2: Gender     Womens
   3: Country    Indonesia
   4: Medal      third: bronze
row 6 is:
   1: Event      Swimming
   2: Gender     Womens
   3: Country    New Zealand
   4: Medal      first: gold
```

Other output examples with different sort keys are shown on the FAQ page. If there are no values supplied on the command line, your program should simply exit after completing Stage 1, without generating any Stage 2 or Stage 3 output. You may assume that there will be no more command-line arguments specified than there are columns in the tsv file.

Distractions to steer clear of: you probably *shouldn't* try and use the system `qsort()` function, and are permitted to make use of the insertionsort algorithm rather than implement your own version of quicksort – this project is about programming in C, not about algorithm implementation, and there won't be a penalty for using insertionsort. Apart from that, keep it simple, and use functions sensibly.

Stage 3 – Hierarchical Reporting (marks up to 15/15)

Now use the sorted tsv values to generate a report that shows counts of the rows that match the selected column combination:

```
mac: ./ass1-soln 3 1 < test0.tsv
<<<Stage 1 and 2 output here, see above>>>
Stage 3 Output
------------------
Country
    Event    Count
------------------
China
    Cycling      1
    Swimming     1
Indonesia
    Swimming     1
New Zealand
    Cycling      2
    Swimming     1
------------------
```

Note how the output appears in the sort order that was the result of Stage 2. Note also that the "Count" column starts one space to the right of the longest entry (including the column header) in the last column getting printed. There are no tabs in the output, and the formatting is achieved through the use of spaces. You should make your output as close to these examples as you can, and identical if possible. Marks will be deducted for discrepancies. The format descriptor `*` will be your friend to help achieve this goal, use `printf("%-*s", num, str)` to print the string `str` left-justified in `num` columns, where `num` is a variable/value calculated while your program is executing.

And as a final hint, you should probably write your program so that in the first instance it generates more extensive output than is shown here. Then, prior to submission, suppress the debugging output by commenting out the extra `printf()` debugging lines. Check carefully that your output matches the examples. (Note that the example output files on the FAQ page use Unix newline formatting, that is, no `CR+LF` combinations.)

The boring stuff...

This project is worth 15% of your final mark. A rubric explaining the marking expectations will be provided on the FAQ page.

You need to submit your program for assessment; detailed instructions on how to do that will be posted on the LMS once submissions are opened. Submission will *not* be done via the LMS; instead you will need to log in to a Unix server and submit your files to a software system known as `submit`. You can (and should) use `submit` **both early and often** – to get used to the way it works, and also to check that your program compiles correctly on our test system, which has some different characteristics to the lab machines. *Failure to follow this simple advice is highly likely to result in tears.* Only the last submission that you make before the deadline will be marked.

You may discuss your work during your workshop, and with others in the class, but what gets typed into your program must be individual work, not copied from anyone else. So, do **not** give hard copy or soft copy of your work to anyone else; do **not** "lend" your "Uni backup" memory stick to others for any reason at all; and do **not** ask others to give you their programs "just so that I can take a look and get some ideas, I won't copy, honest". The best way to help your friends in this regard is to say a very firm "**no**" when they ask for a copy of, or to see, your program, pointing out that your "**no**", and their acceptance of that decision, is the only thing that will preserve your friendship. *A sophisticated program that undertakes deep structural analysis of C code identifying regions of similarity will be run over all submissions in "compare every pair" mode. Students whose programs are so identified will be referred to the Student Center for possible disciplinary action without further warning. This message* ***is*** *the warning.* See `https://academichonesty.unimelb.edu.au` for more information.

Deadline: Programs not submitted by **10:00am on Monday 19 September** will lose penalty marks at the rate of two marks per day or part day late. Students seeking extensions for medical or other "outside my control" reasons should email `ammoffat@unimelb.edu.au` as soon as possible after those circumstances arise. If you attend a GP or other health care professional as a result of illness, be sure to take a Health Professional Report form with you (get it from the Special Consideration section of the Student Portal), you will need this form to be filled out if your illness develops in to something that later requires a Special Consideration application to be lodged. You should scan the HPR form and send it in connection with any non-Special Consideration assignment extension requests.

Marks and a sample solution will be available on the LMS before Tuesday 4 October.

And remember, algorithms are fun!