The University of Melbourne
School of Computing and Information Systems
COMP10002 Foundations of Algorithms
Semester 1, 2020
Assignment 2
Due: 4pm Tuesday 2nd June 2020

1 Learning Outcomes

In this assignment you will demonstrate your understanding of dynamic memory allocation, linked data structures, and search algorithms. You will further extend your skills in program design and implementation.

2 The Story...

There are around 16 million credit cards on issue in Australia,[1] and the number is over 1 billion worldwide.[2] This is a goldmine for cybercriminals who make unauthorised payment to obtain goods or services (i.e., *credit card fraud*). Worldwide losses from card fraud is expected to reach US$31 billion in 2020.[3] Banks and card companies are strongly motivated to develop anti-fraud technologies. They prevented two-thirds of the attempted card fraud in the UK in 2018,[4] but this is a never-ending battle.

There are various anti-fraud algorithms. The core of those algorithms are rules and statistics (machine learning algorithms) to classify whether a transaction is abnormal and likely to be fraudulent. For example, a transaction well beyond the credit limit of a card is likely to be fraudulent, and so are two transactions of the same card issued at almost the same time but from two different cities.

3 Your Task

In this assignment, you will write a program to process credit card and transaction records and identify fraudulent transactions. *Note that you do not need to be an anti-fraud expert to complete this assignment.* You are given a list of credit card records and transactions to be processed in the following format.

```
3tu2iywy 10000 800
ceww0p66 150 100
v0xyil5t 3000 500
vb3dtxp0 5000 300
xnwxw8tl 2000 800
%%%%%%%%%%
v5buvljyh0lg vb3dtxp0 2020:04:07:03:50:45 42
1yuy3noa2uxu xnwxw8tl 2020:04:08:04:16:20 729
gl3utnnwxf49 xnwxw8tl 2020:04:08:05:39:00 670
9mopqy3snk3h ceww0p66 2020:04:15:08:06:49 86
6hjqaydtmrq5 vb3dtxp0 2020:04:15:10:09:50 213
mlgtqk8oo74e ceww0p66 2020:04:15:13:45:29 95
u7s604f0u6bz xnwxw8tl 2020:04:22:15:30:43 799
2siil57yqe5k vb3dtxp0 2020:04:23:17:26:20 573
vzdg2z09t7zp v0xyil5t 2020:04:29:18:03:00 3489
n6lcvjyrhvwt 3tu2iywy 2020:04:29:23:07:00 4592
```

[1] https://www.aph.gov.au/Parliamentary_Business/Committees/Senate/Economics/Credit_Card_Interest/Report/c02
[2] https://wallethub.com/edu/cc/number-of-credit-cards/25532/
[3] https://theconversation.com/credit-card-fraud-what-you-need-to-know-78542
[4] https://en.wikipedia.org/wiki/Credit_card_fraud

The input starts with a list of credit card records *sorted by card ID*. There are **at least 1 and at most 100 cards**. Each credit card occupies one line with three fields separated by a single space: unique card ID (8 alphanumeric characters; no uppercase letters), daily limit (the total amount that can be spent in a day; a positive integer), and transaction limit (the amount that can be spent in a transaction; a positive integer). The transaction limit does not exceed the daily limit.

The line "%%%%%%%%%%" indicates the end of credit card records and the start of transactions.

The transactions are *sorted by date and time*. Each transaction occupies one line with four fields separated by a single space: unique transaction ID (12 alphanumeric characters; no uppercase letters), card ID (8 alphanumeric characters; no uppercase letters; must have appeared in the card records), transaction date and time (with format `year:month:day:hour:minute:second`, two digits for each component), and transaction amount (a positive integer, for simplicity). There is no given limit on the number of transaction lines.

You may assume that the input data is always in valid format. No input format validity checking is needed.

3.1 Stage 1 - Reading One Credit Card Record (Up to 4 Marks)

Your first task is to design a "`struct`" to represent a credit card record. You will then read in a credit card record from the input data, and output it in the following format.

```
=========================Stage 1=========================        /* 25 '='s each side */
Card ID: 3tu2iywy
Daily limit: 10000
Transaction limit: 800
```

3.2 Stage 2 - Reading All Credit Card Records (Up to 8 Marks)

Next, continue to read all credit card records. You need to design a proper data structure to store the records read. An array will do the job nicely. When this stage is done, your program should output: the total number of credit card records read, the average *daily limit* per card (up to two digits after the decimal point, by using "`%.2f`" in `printf()`), and the credit card with the largest *transaction limit* (if there is a tie, print the card with the smallest ID). The output of this stage based on the sample input is as follows.

```
=========================Stage 2=========================
Number of credit cards: 5
Average daily limit: 4030.00
Card with the largest transaction limit: 3tu2iywy
```

3.3 Stage 3 - Reading the Transactions (Up to 12 Marks)

Your third task is to design a `struct` to represent a transaction, read in the transactions, store them in a *linked data structure*, and output their IDs. The output in this stage for the example above should be:

```
=========================Stage 3=========================
v5buvljyh0lg
1yuy3noa2uxu
gl3utnnwxf49
9mopqy3snk3h
6hjqaydtmrq5
mlgtqk8oo74e
u7s604f0u6bz
2siil57yqe5k
vzdg2z09t7zp
n6lcvjyrhvwt
```

Hint: You may modify the linked list implementation in "`listops.c`" *(link to source code available in lecture slides) to store the transactions.* You are free to use other linked data structures (queues, stacks, etc.) if you wish. Make sure to attribute the use of external code properly.

If you are unconfident with linked structures, you may use an array of `struct`'s to store the transactions, assuming at most 20 transactions. If you do so, the full mark of this stage will be reduced from 4 to 2.

3.4 Stage 4 - Checking for Fraudulent Transactions (Up to 15 Marks)

The next stage is to check whether a transaction may be fraudulent. You will go through the transactions. For each transaction, you need to check if it exceeds the transaction limit or the daily limit of the corresponding credit card.

You will use the binary search algorithm (*Hint: You may use the* "`binary_search()`" *function, link to source code available in lecture slides, or* "`bsearch()`" *provided by* "`stdlib.h`") to look up the credit card ID of each transaction from the credit card records read in Stage 2. You may assume that all credit card IDs in the transactions can be found in the credit card records.

A sample output given the input example above is (note a final newline character '`\n`' at the end):

```
========================Stage 4========================
v5buvljyh0lg              IN_BOTH_LIMITS
1yuy3noa2uxu              IN_BOTH_LIMITS
gl3utnnwxf49              IN_BOTH_LIMITS
9mopqy3snk3h              IN_BOTH_LIMITS
6hjqaydtmrq5              IN_BOTH_LIMITS
mlgtqk8oo74e              OVER_DAILY_LIMIT      /* 86 + 95 > 150 */
u7s604f0u6bz              IN_BOTH_LIMITS
2siil57yqe5k              OVER_TRANS_LIMIT      /* 573 > 300 */
vzdg2z09t7zp              OVER_BOTH_LIMITS      /* 3489 > 3000 and 500 */
n6lcvjyrhvwt              OVER_TRANS_LIMIT      /* 4592 > 800 */
```

Note also that you should only go through the transaction list *once*, that is, you need to design an algorithm with an `O(nlogm)` average time complexity for this stage, given `n` transactions and `m` credit card records. **At the end of your submission file, you need to add a comment that states the average time complexity of your algorithm, and explains why it has that time complexity.**

If the time complexity analysis is missing, or the average time complexity of your algorithm is higher than `O(nlogm)`, the full mark of this stage will be reduced from 3 to 2.

Hint: To achieve the target time complexity, you may need to modify the credit card `struct` *to store additional information.*

4 Submission and Assessment

This assignment is worth 15% of the final mark. A detailed marking scheme will be provided on the Canvas.

You need to submit your code in Grok Learning (`https://groklearning.com`) for assessment. Submission will NOT be done via the Canvas. Instead, you will need to:

1. Log in to Grok Learning using your student login details.
2. Navigate to the Assignment 2 module of our subject COMP10002 2020 S1: Foundations of Algorithms.
3. Place your code in the `program.c` tab window.
4. Compile your code by clicking on the `Compile` button.
5. Once the compilation is successful, click on the `Mark` button to submit your code. (You can submit as many times as you want to. *Only the last submission made before the deadline will be marked.*)
6. Two sample tests will be run automatically after you make a submission. Make sure that your submission passes these sample tests.
7. Two hidden tests will be run for marking purpose. Results of these tests will not be available until after the submissions are marked.

You can (and should) submit both **early and often** – to check that your program compiles correctly on our test system, which may have some different characteristics to the lab machines and your own machines.

You will be given a sample test file `test0.txt` and the sample output `test0-output.txt`. You can test your code on your own machine with the following command and compare the output with `test0-output.txt`:

```
mac: ./program < test0.txt    /* Here '<' feeds the data from test0.txt into program */
```

Note that we are using the following command to compile your code on the submission system (we name the source code file `program.c`).

```
gcc -Wall -std=c99 -o program program.c
```

The flag "`-std=c99`" enables the compiler to use a more modern standard of the `C` language – `C99`. To ensure that your submission works properly on the submission system, you should use this command to compile your code on your local machine as well.

You may discuss your work with others, but what gets typed into your program must be individual work, **not** from anyone else. Do **not** give (hard or soft) copy of your work to anyone else; do **not** "lend" your memory stick to others; 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.* See `https://academichonesty.unimelb.edu.au` for more information.

Deadline: Programs not submitted by **4pm Tuesday 2nd June 2020** will lose penalty marks at the rate of 2 marks per day or part day late. Late submissions after 4pm Friday 5th June 2020 will **not** be accepted. Students seeking extensions for medical or other "outside my control" reasons should email the lecturer at jianzhong.qi@unimelb.edu.au. 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 into 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.

Special consideration due to COVID-19: Please refer to the "Special Consideration" section in this page: `https://students.unimelb.edu.au/student-support/coronavirus/information-for-all-students/`

And remember, *Algorithms are fun!*