COMP9024 17s2

Assignment 1
Student Records

Data Structures and Algorithms

Change Log

We may make minor changes to the spec to address/clarify some outstanding issues. These may require minimal changes in your design/code, if at all. Students are strongly encouraged to check the change log regularly.

14 August

- Submission instruction (`give` command) updated

Version 1: Released on 11 August 2017

Objectives

The assignment aims to give you more experience with

- dynamic data structures, specifically dynamic arrays and linked lists
- implementing (abstract) data types
- asymptotic runtime analysis

Admin

Marks	4 marks for stage 1 (correctness) 2 marks for stage 2 (correctness) 2 marks for stage 3 (correctness) 1 mark for complexity analysis (stages 2 & 3 only) 1 mark for style ——————— Total: 10 marks
Due	23:59 on **Wednesday** 30 August (week 6)
Late	1.5 marks (15%) off the ceiling per day late (e.g. if you are 25 hours late, your maximum possible mark is 7)

Aim

Your task is to write a program for reading and processing a collection of student records (stage 1) and for maintaining them (stages 2 & 3). You will have to complete wrapper code to test your implementations, and make sure to test a variety of conditions. You will also have to provide the time complexity of your functions for maintaining the student records.

Provided Code

A *zip file* is a way of compressing and packaging files. We have provided a zip file as a base for you to begin the assignment. You can download it here: `src.zip`

When you `unzip` the package, you will find six files in your current directory:

- `main.c`: The incomplete main program.
- `studentRecord.h`: A header file that lists the functions in the student record implementation. *Do not change.*
- `studentRecord.c`: An incomplete library of functions for reading and writing single student records.
- `studentLL.h`: A header file that lists the functions in the linked list implementation. *Do not change.*
- `studentLL.c`: An incomplete library of functions for maintaining a linked list of student records.
- `Makefile`: Used to make an executable `main` from the main program and the two modules.

Stage 1 – Dynamic Array (4 marks)

For stage 1, you will need to implement functions in `studentRecord.c` and write code in `main.c` for a program that

1. accepts a positive number *n* as a single command line argument
2. uses a ***dynamic array*** to store *n* student records from user input
3. prints all student records
4. prints some statistical information

For each student, your program should prompt the user to

- `"Enter student ID: "` (expecting a 7-digit number)
- `"Enter credit points: "` (expecting a number between 2 and 480)
- `"Enter WAM: "` (a floating point number between 50 and 100 — we assume that the WAM is calculated from passed courses only)

Your program should detect any invalid input and, where necessary, ask the user to re-enter data with the message `"Not valid. Enter a valid value: "`.

The output should be as follows:

- for each student,
  - an opening line `"------------------------"`
  - a line `"Student zID: "` followed by their zID
  - a line `"Credit points: "` followed by their total credit points
  - a line `"Level of performance: "` followed by a grade (PS, CR, DN or HD) indicating their overall level of performance
    - based on their WAM
    - by applying UNSW's Grade Definition to the WAM
  - a closing line `"------------------------"`
- a line `"Average WAM: "` followed by the average WAM across all students
- a line `"Weighted average WAM: "` followed by the *weighted* average WAM across all students
  - where each WAM is weighed by the total credit points of the student

Example

Here is an example to show the desired behaviour of your program for stage 1:

```
prompt$ ./main 2
Enter student ID: 123456
```

```
Not valid. Enter a valid value: 1234567
Enter credit points: 6
Enter WAM: 65
Enter student ID: 1111111
Enter credit points: 12
Enter WAM: 101.5
Not valid. Enter a valid value: 95
------------------------
Student zID: z1234567
Credits: 6
Level of performance: CR
------------------------
------------------------
Student zID: z1111111
Credits: 12
Level of performance: HD
------------------------
Average WAM: 80.000
Weighted average WAM: 85.000
```

Note:

- Student records are printed in the order in which they have been entered.
- You may assume that no student ID is entered twice.
- (Weighted) average WAM gets rounded to third decimal place when printed.

Stage 2 – Linked List (2 marks)

Aim: For stage 2, you will implement most of the functions in `studentLL.c`, analyse the time complexity of your implementation and add code to `main.c`.

When you run the main program without command line argument, you will be presented with these options:

```
prompt$ ./main
Enter command (a,f,g,p,q, h for Help)>
```

If you type **`h`** for Help, you will see a more detailed list of commands:

```
A - Add student record
F - Find student record
G - Get statistics
H - Help
P - Print all records
Q - Quit
```

When you type **`q`** the program exits.

For this stage, you are to use a ***dynamic linked list*** to implement the following commands:

1. `A` – prompts the user to input valid data for a student record as above, inserts the record into the linked list and outputs "`Student record added.`"
2. `G` – outputs the total number of records, average WAM and weighted average WAM across all student records in the list
3. `P` – prints all student records

4. Your program should include a time complexity analysis, in Big-Oh notation, for your

functions in `studentLL.c`:

- The size *n* of the input is given by the length of the linked list (i.e. the number of student records).
- Each function should be preceded by two comment lines:

```
// Time complexity: O(...)
// Explanation: ...
```

- You only need to provide time complexity for the functions that are also defined in `studentLL.h`.

Example

Here is an example to show the desired behaviour of your program for stage 2:

```
prompt$ ./main
Enter command (a,f,g,p,q, h for Help)> a
Enter student ID: 3333333
Enter credit points: 12
Enter WAM: 8.95
Not valid. Enter a valid value: 89.5
Student record added.
Enter command (a,f,g,p,q, h for Help)> a
Enter student ID: 1111111
Enter credit points: 15
Enter WAM: 74.5
Student record added.
Enter command (a,f,g,p,q, h for Help)> g
Number of records: 2
Average WAM: 82.000
Average weighted WAM: 81.167
Enter command (a,f,g,p,q, h for Help)> p
------------------------
Student zID: z1111111
Credits: 15
Level of performance: CR
------------------------
------------------------
Student zID: z3333333
Credits: 12
Level of performance: HD
------------------------
Enter command (a,f,g,p,q, h for Help)> q
Bye.
```

Note:

- As before, you can assume that all student IDs will be different.
- For stage 2, new records should always be added to the *beginning* of the list
  - so they are displayed in reverse order when printed.

Stage 3 – Ordered Linked List (2 marks)

Aim: For stage 3, you will modify and complete the implementation in `studentLL.c` and also complete `main.c`:

1. Extend the functionality of the command `A` (adding a student record):

- *New* records should now be inserted into the list in *ascending order* of the student ID.
- *Existing* records should be updated with new data, and the message "`Student record updated.`" printed.

2. Implement the command `F` to
   a. prompt the user to input a 7-digit student ID
   b. output the student's record if it is in the list, and otherwise print the message "`No record found.`".
3. Your program `studentLL.c` should include a time complexity analysis, in Big-Oh notation, for all your functions that are also defined in `studentLL.h`.

Example

Here is an example to show the desired behaviour of your program for stage 3:

```
prompt$ ./main
Enter command (a,f,g,p,q, h for Help)> a
Enter student ID: 1111111
Enter credit points: 15
Enter WAM: 77
Student record added.
Enter command (a,f,g,p,q, h for Help)> f
Enter student ID: 333333
Not valid. Enter a valid value: 3333333
No record found.
Enter command (a,f,g,p,q, h for Help)> a
Enter student ID: 3333333
Enter credit points: 12
Enter WAM: 85
Student record added.
Enter command (a,f,g,p,q, h for Help)> p
------------------------
Student zID: z1111111
Credits: 15
Level of performance: DN
------------------------
------------------------
Student zID: z3333333
Credits: 12
Level of performance: HD
------------------------
Enter command (a,f,g,p,q, h for Help)> f
Enter student ID: 3333333
------------------------
Student zID: z3333333
Credits: 12
Level of performance: HD
------------------------
Enter command (a,f,g,p,q, h for Help)> a
Enter student ID: 3333333
Enter credit points: 12
Enter WAM: 85
Student record updated.
Enter command (a,f,g,p,q, h for Help)> g
Number of records: 2
Average WAM: 81.000
Average weighted WAM: 80.556
Enter command (a,f,g,p,q, h for Help)> q
Bye.
```

Testing

We have created a script that can automatically test your program. To run this test you can execute the `dryrun` *program for the corresponding assignment, i.e.* `assn1`*. It expects to find the programs* `main.c, studentRecord.c, studentLL.c` *in the current directory. You can use dryrun as follows:*

```
prompt$ ~cs9024/bin/dryrun assn1
```

Please note: Passing the dryrun tests does not guarantee that your program is correct. You should thoroughly test your program with your own test cases.

Submit

For this project you will only need to submit program files (.c files). You can either submit through WebCMS3 or use the submit command

```
prompt$ give cs9024 assn1 main.c studentRecord.c studentLL.c
```

Do not forget to add the time complexity to your source code file `studentLL.c`.

You can submit as many times as you like — later submissions will overwrite earlier ones. You can check that your submission has been received on WebCMS3 or by using the following command:

```
prompt$ 9024 classrun -check
```

Marking

This project will be marked on functionality in the first instance, so it is very important that the output of your program be ***exactly*** correct as shown in the examples above. Submissions which score very low on the automarking will be looked at by a human and may receive a few marks, provided the code is well-structured and commented.

Programs that generate compilation errors will receive a very low mark, no matter what other virtues they may have. In general, a program that attempts a substantial part of the job but does that part correctly will receive more marks than one attempting to do the entire job but with many errors.

The main objective of this assignment is to give you more experience with dynamic data structures. You will receive 0 marks for stage 1 if your program is not using a dynamic array, and 0 marks for stage 2 & 3 if your program is not using a dynamic linked list.

Plagiarism

Group submissions will not be allowed. Your program must be entirely your own work. Plagiarism detection software will be used to compare all submissions pairwise (including submissions for similar projects in previous years, if applicable) and serious penalties will be applied, particularly in the case of repeat offences.

Do not copy from others; do not allow anyone to see your code, not even after the deadline

Please refer to the on-line sources to help you understand what plagiarism is and how it is dealt with at UNSW:

- Plagiarism and Academic Integrity
- UNSW Plagiarism Procedure

Help

See FAQ for some hints on how to get started and for troubleshooting tips.

Finally …

Have fun! Michael