CS1020E: Data Structures and Algorithms I

Tutorial 4 – ADTs, Lists

17 February 2017

# 1. NUS Modules

Have you ever thought about writing a program to calculate your NUS Cumulative Average Point (CAP)? Here, we will be implementing an **NUSModule** ADT, and use an array of **NUSModule** objects to calculate a CAP score.

The **NUSModule** ADT needs to be able to store the module code, the number of modular credits that it is worth, and the module letter grade. In addition, it should be able to get the grade point equivalent of the letter grade. These should all be stored as private attributes within the class, due to encapsulation.

All modules are initially given a letter grade from A+ to F, but it can also be declared as an S/U grade at a later time (assume all modules can be declared as S/U). S and U are special grades. If a module is given any of those grades, that module will contribute 0 modular credits to the CAP.

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Letter | A+ | A | A- | B+ | B | B- | C+ | C | D+ | D | F |
| Point | 5.0 | 5.0 | 4.5 | 4.0 | 3.5 | 3.0 | 2.5 | 2.0 | 1.5 | 1.0 | 0.0 |
| S/U | S | S | S | S | S | S | S | S | U | U | U |

Source: <http://www.eng.nus.edu.sg/ugrad/SI_caps.html>

A skeleton code of the class is provided below.

class NUSModule {

private:

string \_name;

int \_credits;

string \_grade;

double \_point;

// other private attributes and methods here

public:

NUSModule( string name, int credits, string grade = “F” ) // some default grade

: \_name( name ), \_credits( credits ), \_grade( grade )

{ /\* anything else needed? \*/ }

string toString () {

ostringstream oss;

oss << \_name << “(“ << \_credits << “): ” << \_grade << “ = “ << \_point;

return oss.str();

}

void setGrade( string grade ); // sets the grade, and updates the point

// other public methods here

};

(a) List some methods that you think should be included, and specify whether they should be public or private methods.

(b) The method **setGrade** has not been implemented. Implement the method.

(c) Implement a **calcCAP** function that calculates the CAP, given an array of **NUSModules.**

double calcCAP( NUSModule moduleList[ ], int numModules);

# 2. Fractions

C++ contains several numeric data types, e.g. int, double, etc. However, it does not support fractions. We will store our fraction as an irreducible fraction, which means having an integer numerator and denominator with no common factors within them. We also need to implement some arithmetic operations on fractions.

One way of implementing a fraction in C++ is to inherit from the STL **pair< int, int >,** which is defined in the header **< utility >**:

class Fraction : pair<int, int> {

private:

//some private methods here

public:

Fraction( int num, int den) {

first=num;

second=den;

//anything else needed?

}

string getFraction () {

ostringstream oss;

oss << first << "/" << second;

return oss.str();

}

Fraction add(Fraction other);

//adds the current fraction with another fraction, and then returns the result as a fraction

//other public methods here

};

(a) List some of methods that you think should be included, and specify whether they should be public or private methods.

(b) The method **add** has not been defined. Implement the method.

(c) Implement 3 other methods proposed in (a), not including getter/setter methods.

# 3. List ADT Implementations

In the lectures, we have discussed the array and singly linked list implementations for a list ADT. Let us compare their performances. Given a list containing **N** elements, how many elements will be accessed/modified for each implementation when:

(a) Adding an element to the front of the list.  
(b) Adding an element to the back of the list.  
(c) Removing an element from the front of the list.  
(d) Removing an element from the back of the list.  
(e) Retrieve an element by index.

# 4. Singly Linked List

Implement an integer linked list class, with the following methods:

(a) **push** – adds an integer to the front of the list  
(b) **pop** – removes an integer from the front of the list  
(c) **retrieve** – returns the integer stored at position idx  
(d) **remove** – removes the integer stored at position idx   
(e) **sort** – sorts the integers in ascending order  
(f) **constructor** – ensure that there is no improper pointer dereferencing at any point  
(g) **destructor**– ensure that there is no memory leak at the end of the life of a **LinkedList** object.

A skeleton code is provided for you below. You may implement other helper methods.

class LinkedList {

private:

struct Node {

int number;

Node\* next;

};

Node\* \_head;

public:

LinkedList () ;

~LinkedList () ;

void push( int value );

void pop () ;

int retrieve( int idx );

void remove( int idx );

void sort () ;

};