CSE2050 - Programming in a Second Language
Assignment 2: Dynamic Memory

September 28, 2017

1 Submission

- Due Date: **October 6th, 11:59pm**
- Deliverable: **numberList.cpp** (case sensitive)
- Submit on **Canvas**.

2 Description

This problem is primarily about computing GCD and LCM of a sequence of integers stored in a linked list. In this assignment, we will focus on classes and Dynamic Memory Allocation. The goal is to build Singly Linked List with a variable number of integers, and perform some mathematical operations on those numbers (as described below) using `C++` .

The linked list consists of a number of identical *Nodes*, each of which contains (as shown in Figure 1,) a pointer (**next*) that points to the next node. The "next" pointer of the last node has a value of *nullptr* (represented below in Figure 2 with the right most node with an X).

The *head* pointer keeps track of the first node in the list, and we can reach the other nodes by following the *next pointers.

Figure 1: A node

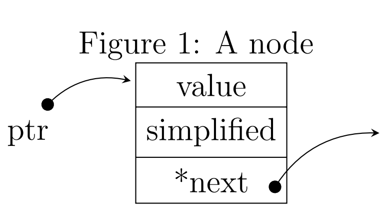
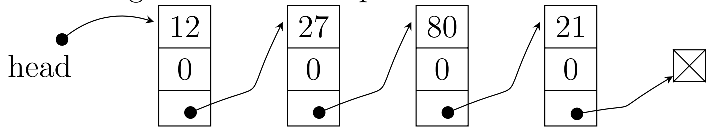


Figure 2: An example of a linked list



3 Important Notes

Here are some points that we consider when grading your submission..

- Submit only your source code (.cpp and .h files). No other files are needed (or graded!).
- Make sure your code runs on our server (**code01.fit.edu**). Your program will be compiled using the following flags:

```
g++ -Wall -g -std=c++11 -o numberList numberList.cpp
```

- To help the automated grading system to correctly read the results of your code, please make sure you follow the format shown in the examples below (including extra spaces and/or extra characters in the output will cause your results to be flagged as incorrect, which will then have to be re-graded manually). Please do NOT print out any prompts (like "Enter a number",.. etc) or other comments (like "the result is",..etc).
- Please make sure your program handles the input correctly. The problem description for each assignment provides details about the method to use (e.g., from a file, or from the command-line,..etc) and the format of that input. If, for example, the input is to be provided from the command-line (as in this project), but your program can only read input from standard input, it will fail our tests (will affect your grade).

4 Implementation details

4.1 Input

A variable number of integers is provided from the command line, as follows:

```
./numberList a b c d ...
```

The program receives these integers and uses them to create the linked list. Note that, for the purpose of this assignment, the order you choose to store these numbers in the list does not affect the results (why?).

More details on reading and handling command line arguments, can be found on cplusplus.com

4.2 Output

For each sequence of numbers from command line, the program prints out the following two lines:

- The first line contains (separate the values with a single space):

    1. the size of the list (number of nodes),
    2. the product of all numbers,
    3. the list's GCD, and
    4. the list's LCM.

- The second line contains the simplified numbers (separated by single spaces) printed in the reverse order (despite what we mentioned in class about the order not affecting the results, these numbers are required to be printed in the reverse order).

Please note that the automated test cases are evaluated based on a character-by-character comparison (i.e., don't print other things, just the values mentioned above).

4.3 General notes

- Practicing how to find manually the GCD and LCM for lists of numbers can help greatly implementing the most efficient (and the correct!) algorithm to solve these problems.

- Only implementations based on *Dynamic Memory allocation* will receive full points.

- The *value* field stores the number read from input, and the *simplified* field is to store the simplified version of this number, with respect to the common GCD of the list (initialized to zero).

- implement the *node* as a *class* with both *value* and *simplified* as private fields (you will need public *getter* and *setter* methods to access these fields).

  More on *encapsulation* and *data hiding* can be found here

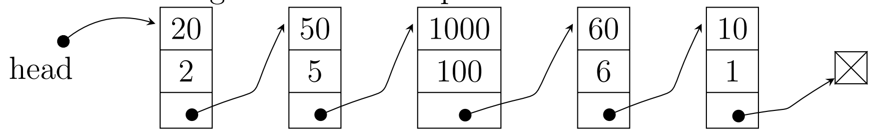
- You may use any algorithm you prefer to calculate the GCD and the LCM of the list.

  LCM of multipe numbers (i.e., >2) can be computed sequentially using an approach similar to this:

  lcm(a,b,c) = lcm(a,lcm(b,c))

- After creating the linked list with the provided integers (stored in reverse order, i.e., *head* points to the the node with the last integer from command line), The following operations are required:

Figure 3: An example of a linked list



- Get the size of the list (number of nodes in it),
- Get the product of all the numbers,
- Calculate the list GCD (i.e., for all the numbers in the list),
- Simplify all the numbers in the list (with respect to the common GCD), and
- Calculate the list LCM (i.e., for all the numbers in the list).

4.4 Example

For example, running the program with the following arguments..

```
   ./numberList 10 60 1000 50 20
```

Will create the linked list in Figure 3:

Note that, in this example:

1. the size of the list is 5
2. the product of all numbers is 600,000,000
3. the list's GCD is 10
4. the list's LCM is 3000

And these values are shown in the required order and format in the corresponding output sample below:

```
5 600000000 10 3000
2 5 100 6 1
```

5 Sample input/output

Each of the following examples shows the command used to execute the program (with the input values) on the first line, and the results printed by the program for that input on the following two lines.. Note also that, in this implementation, the list is stored (and printed) in the reverse order of input.

```
./numberList 4 8 2 6
4 384 2 24
3 1 4 2
```

```
./numberList 3 300
2 900 3 300
100 1
```

```
./numberList 42 7 21 63 640
5 248935680 1 40320
640 63 21 7 42
```

```
./numberList 20 100 5 30 120 2500 45 70
8 2093692928 5 315000
14 9 500 24 6 1 20 4
```

```
./numberList 1 2 3 4 5 6 7 8 9 10
10 3628800 1 2520
10 9 8 7 6 5 4 3 2 1
```

```
./numberList 10 20 7 48 32
5 2150400 1 3360
32 48 7 20 10
```

```
./numberList 17 11 13 19
4 46189 1 46189
19 13 11 17
```

6 Rubric

Criterion	Possible points	Excellent (max. points)	Satisfactory (partial points)	Unsatisfactory (no points)
Delivery	10%	on time, using correct file name (see top)		wrong file name
Compiles	10%	compiles on code01.fit.edu with no errors		does not compile
Runs	10%	runs on code01.fit.edu with no run-time errors (missing files,..etc)		does not execute
Completion	25%	all functions implemented		
Dynamic Memory allocation	20%	implemented using pointers/Dynamic Memory allocation	implemented using other memory access methods	
Test cases	25%	passes all test cases (correct output in the correct format)	passes some test cases (correct output in the wrong format, or contains extra characters, ...etc	fails all test cases