# CSE2050 - Programming in a Second Language
# Assignment 3: Linked List Manipulation

February 27, 2018

## 1 Submission

- Due Date: **March 2nd, 11:59pm**
- Deliverable: **listGCD.cpp**
- Submit on **Canvas**.

## 2 Description

In this assignment, we expand the problem of computing the GCD of a sequence of integers stored in a linked list (as discussed in *Assignment 2*) by providing the functionality of adding and removing numbers (nodes) to/from that list. The goal is to build a Singly Linked List (with a node structure similar to the one used in *Assignment 2*) to store a number of integers (given), and calculating the GCD of those numbers. Providing the following:

- New numbers can be added at a specific location in the list.
- Numbers can be removed from the list (either by providing their location, or their value).

## 3 Implementation details

### 3.1 Input

The program reads input from a text file. The name of the input file is provided as the only command line parameter (only the name of the file will be provided, not the full path).

The input file contains sequences of positive integers on each line. The first integer $\boldsymbol{x}$ in each sequence determines the operation to be performed, and the number of the following integers in this sequence. possible formats:

| *input* | description |
|---|---|
| **1** $\boldsymbol{i}$ $\boldsymbol{v}$ | add node at index i, with value v |
| **2** $\boldsymbol{i}$ | remove node at index i |
| **3** $\boldsymbol{v}$ | remove first encountered node with value v |
| **0** | quit |

For example, the input line: **1 3 4** tells the program to add a node (the first number, $\boldsymbol{x} = 1$) at index 3 (the second number, $\boldsymbol{i} = 3$), with a value 4 (third number, $\boldsymbol{v} = 4$).

The program ends if the line is **0** (no other values follow).

The number of these sequences is not known in advance, so your program should read all the sequences until it encounters a **zero**, then it stops reading (the zero value is not stored in the list).

It is important to make sure the loop used to read these values ends properly. Endless loops are considered failed cases (0 points).

```
./listGCD filename.txt
a b c
d e
f g
h i j
...
```

## 3.2 Output

- For each sequence of numbers, the program prints out the following two lines:
  - The first line shows the list's GCD (single spaces). An example:

    ```
    List GCD: 15
    ```

  - The second line shows the contents of the list (separated by single spaces) printed in the same order of input (the first number entered on the left). An example:

    ```
    List contents: 14 15 16
    ```

- If the list is empty (for example, after removing all the nodes), show the following:

  ```
  List GCD: 0
  List contents:
  ```

- On exit (when receiving the value 0), there is no output.

## 3.3 General notes

- Only implementations based on *Dynamic Memory Allocation* will receive full points.
- implement the *node* as a *class* with *value* as private field (you will need public *getter* and *setter* methods to access it). It is also recommended to use *constructor* and *destructor* methods.
- You may use any algorithm you prefer to calculate the GCD of the list.
- The index $\boldsymbol{i}$ is used to determine the node's location in the list (starts from 0 for the $1^{st}$ node), and $\boldsymbol{v}$ represents the actual value for that node.
- When adding a new node ($op\_code = 1$), the index $\boldsymbol{i}$ has the following meanings:
  - if $\boldsymbol{i}$ is 0, *add first*. The new node will be prepended to the beginning of the current list (it becomes the first node), then *head* will point to the newly added node.
  - if $\boldsymbol{i} \geq \boldsymbol{N}$ (where $\boldsymbol{N}$ is the size of the list, i.e., the number of items currently in the list), *add last*. The new created node will be added to the end of the list.
  - otherwise, the new created node will be inserted at the specified index (becomes the $i^{th}$ node).
- When removing a node giving its index $\boldsymbol{i}$ ($op\_code = 2$), if the index does not exist ($\boldsymbol{i} \geq \boldsymbol{N}$), do nothing (do not remove, do not show error message).
- When removing a node giving its value $\boldsymbol{v}$ ($op\_code = 3$), remove the first occurrence of that value in the list (the first encountered node with value $\boldsymbol{v}$). If the value does not exist, do nothing (do not remove, do not show error message).
- Managing *Dynamic Memory Allocation* properly (20% of this assignment's grade) requires avoiding issues like memory leaks. Depending on your OS and IDE, there are some tools that can help identify some of these issues (like cppcheck).
- Same notes from previous assignments apply here as well (correct deliverable name, runs on **code01.fit.edu**, output format is important, and make sure you use the correct input source for data)
- Make sure you know where your IDE looks for files (the default directory). In our testing, the input files will be in the same folder as your compiled files (the executable).

## 3.4 Example

The following example, shows the program's responses to input (all the responses are shown in order):

- Running the program

```
./listGCD smaple1.txt
```

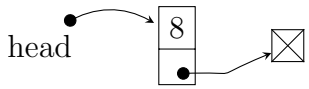
- Contents of *sample1.txt*

```
1 0 8
1 0 12
1 1 2
3 12
2 1
2 5
2 0
0
```

- Input line 1

```
1 0 8
```

This will create the following linked list,,



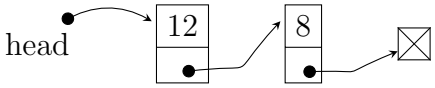

- Output 1

```
List GCD: 8
List contents: 8
```

- Input line 2

```
1 0 12
```

Inserting the value 12 at location 0 (first) changes the linked list as follows:



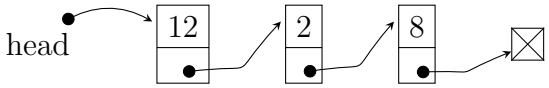

- Output 2

```
List GCD: 4
List contents: 12 8
```

- Input line 3

```
1 1 2
```

Value 2 will be inserted at index 1 (second node):



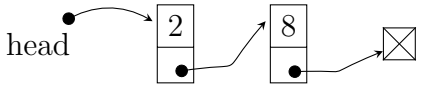

- Output 3

```
List GCD: 2
List contents: 12 2 8
```

- Input line 4

```
3 12
```

Delete first node that has value 12:



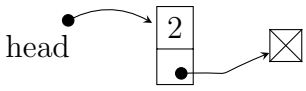

- Output 4

```
List GCD: 2
List contents: 2 8
```

- Input line 5

```
2 1
```

Deleting the second node (index 1):



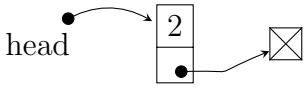

- Output 5

```
List GCD: 2
List contents: 2
```

- Input line 6

```
2 5
```

Attempting to delete node at index 5 (an invalid index) will not actually change the list.




- Output 6

```
List GCD: 2
List contents: 2
```

- Input line 7

```
2 0
```

This deletes the remaining node (the list becomes empty).

- Output 7

```
List GCD: 0
List contents:
```

- Input line 8

```
0
```

0 ends the program (no printing).

# 4 Sample input/output

## 4.1 Sample 1

**Input file contents**

```
1 0 16
1 1 18
1 1 12
2 1
1 2 14
3 12
3 5
1 0 1
0
```

### Output

```
List GCD: 16
List contents: 16
List GCD: 2
List contents: 16 18
List GCD: 2
List contents: 16 12 18
List GCD: 2
List contents: 16 18
List GCD: 2
List contents: 16 18 14
List GCD: 2
List contents: 16 18 14
List GCD: 2
List contents: 16 18 14
List GCD: 1
List contents: 1 16 18 14
```

## 4.2 Sample 2

### Input

```
1 0 24
1 1 12
1 2 6
1 2 18
1 4 3
3 12
2 2
0
```

### Output

```
List GCD: 24
List contents: 24
List GCD: 12
List contents: 24 12
List GCD: 6
List contents: 24 12 6
List GCD: 6
List contents: 24 12 18 6
List GCD: 3
List contents: 24 12 18 6 3
List GCD: 3
List contents: 24 18 6 3
List GCD: 3
```

```
List contents: 24 18 3
```

## 4.3 Sample 3

### Input

```
1 0 25
1 0 8
3 17
2 3
2 2
3 6
1 0 9
2 5
0
```

### Output

```
List GCD: 25
List contents: 25
List GCD: 1
List contents: 8 25
List GCD: 1
List contents: 8 25
List GCD: 1
List contents: 8 25
List GCD: 1
List contents: 8 25
List GCD: 1
List contents: 8 25
List GCD: 1
List contents: 9 8 25
List GCD: 1
List contents: 9 8 25
```

## 4.4 Sample 4

### Input

```
1 0 2
1 0 2
3 2
1 0 6
1 8 90
2 1
2 2
```

```
2 1
2 1
2 3
1 4 23
3 23
0
```

**Output**

```
List GCD: 2
List contents: 2
List GCD: 2
List contents: 2 2
List GCD: 2
List contents: 2
List GCD: 2
List contents: 6 2
List GCD: 2
List contents: 6 2 90
List GCD: 6
List contents: 6 90
List GCD: 6
List contents: 6 90
List GCD: 6
List contents: 6
List GCD: 6
List contents: 6
List GCD: 6
List contents: 6
List GCD: 1
List contents: 6 23
List GCD: 6
List contents: 6
```

# 5 Rubric

| Criterion | Possible points | Excellent (max. points) | Satisfactory (partial points) | Unsatisfactory (no points) |
|---|---|---|---|---|
| Compiles | 5 % | 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 | some functions implemented | |
| Dynamic Memory allocation | 20% | implemented using pointers/Dynamic Memory allocation | | implemented using other memory access methods |
| Avoiding memory leaks | 15% | freeing memory correctly for all pointers & dynamic memory allocation operations | | code has memory leaks |
| Test cases | 25% | passes all test cases (correct output in the correct format) | passes some test cases | fails all test cases, or has an endless loop. |