EC327 - Spring 2025 - Midterm Extra Credit

Background

This is an optional, extra credit assignment. It will be auto-graded on a scale from 0 to 12 and the points will be added to your midterm exam grade, without going over 100.

Important

Please make sure your code compiles and runs as intended on the engineering grid. Code that does not compile will NOT be graded and will receive a 0.

Submission Instructions

You will submit this assignment as single .zip file with the following name:

<first-name>_<last-name>-examextra.zip

So for example:

ed_solovey-examextra.zip.

Your zip file should contain two files for problem-1 and one file for problem-2 in this assignment. Those files should be named like this:

examextra_problem1.h

<first-name>_<last-name>-examextra-1.cpp

<first-name>_<last-name>-examextra-2.cpp

Note on Collaboration

You are welcome to talk to your classmates about the assignment and discuss high level ideas. However, all code that you submit must be your own. We will run code similarity tools against your submissions and will reach out with questions if anything is flagged as suspicious.

Problem 1 - Function Overloading (6 points)

Submission Instructions

NOTE: For this problem, unlike most other problems you have worked on, you will submit a heads file and an implementation file. They should be of the form:

examextra_problem1.h and

<first-name>_<last-name>-examextra-1.cpp

So for example, my files would be:

examextra_problem1.h and **ed_solovey-examextra-1.cpp**

The include line from your **<first-name>_<last-name>-examextra-1.cpp** should be:

#include "examextra_problem1.h"

Actual Problem

Take a look at the following block of code:

```cpp
#include <iostream>

int average(int a, int b) {
    return (a + b) / 2;
}

int main() {
    int num1 = 5;
    int num2 = 10;
    float num3 = 7.5;
    double num4 = 12.8;

    std::cout << average(num1, num2) << std::endl;
    std::cout << average(num1, num3) << std::endl;
    std::cout << average(num3, num2) << std::endl;
    std::cout << average(num4, num1) << std::endl;
    std::cout << average(num3, num4) << std::endl;
    std::cout << average(num4, num4) << std::endl;

    return 0;
}
```

If the only implementation of the **average** function is the one shown above, then the result of executing the **main** function would be:

```
7
6
8
8
9
12
```

However, your goal is to make the output look like this:

```
7
6.25
8.75
8.9
10.15
12.8
```

You should accomplish this by overloading the **average** function as many times as needed.

The declarations of your overloaded signatures should go in **<first-name>_<last-name>-examextra-1.h** and the implementations in **<first-name>_<last-name>-examextra-1.cpp**

You should follow the following rules:

- If any argument is a double, **average** should return a double.
    - otherwise, if any argument is a float, **average** should return a float.
        - otherwise **average** should return an int.

Problem 2 - Linked Lists (6 points)

Submission Instructions

Your solution to this problem should contribute a single **cpp** file to your overall **examextra** zip. The **cpp** file should follow the following format:

<first-name>_<last-name>-examextra-2.cpp

So for example, my file would be:

ed_solovey-examextra-2.cpp

Actual Problem

Take a look at the header file provided below. Your task is to provide implementations of the three functions in your **<first-name>_<last-name>-examextra-2.cpp** file according to the documentation provided for each of the functions.

```
#ifndef EXAMEXTRA_PROBLEM2_H
#define EXAMEXTRA_PROBLEM2_H

/**
 * Struct representing a node in the linked list.
 */
struct Node {
    int data;   // The value stored in the node
    Node* next;  // Pointer to the next node
};

/**
 * Inserts a value into the linked list at a given position.
 * If the position
 * is invalid, either zero or smaller, or out of bounds for the size of the
 * current linked list, the linked list should remain unchanged.
 * @param head Pointer to the head of the linked list.
 * @param value The integer value to insert.
```

```
 * @param position The position to insert the value at (1-based index).
 * @return the head of the resulting linked list.
 */
Node* insertAtPosition(Node* head, int value, int position);

/**
* Removes the node at a given position from the linked list.
* If the position
 * is invalid, either zero or smaller, or out of bounds for the size of the
 * current linked list, the linked list should remain unchanged.
 * @param head Pointer to the head of the linked list.
 * @param position The position to remove the node at (1-based index).
 * @return the head of the resulting linked list.
 */
Node* deleteAtPosition(Node* head, int position);

/**
 * Finds the greatest product of two values in the linked list.
 * @param head Pointer to the head of the linked list.
 * @return the greatest product, or 0 if the list has fewer than two
elements.
 */
int findGreatestProduct(Node* head);

#endif //EXAMEXTRA_PROBLEM2_H
```