CSCI3180 Assignment 4: C++ and Python

2024-2025, Term 2

Introduction

In this two-part assignment, you'll use C++ and Python to implement and use two very small linked list libraries. In particular, you will implement and use methods of a `LinkedList` class in both C++ and Python. Instances of this class are meant to represent lists of elements. There are a total of **100 points** for this assignment.

Guidelines

You should follow the below guidelines during completion of the assignment:

1. Modify the files `assignment_4.cpp` and `assigment_4.py` by adding your code and/or comments.
2. You may define **at most one** additional function or method in `assignment_4.cpp`.
3. You may not define any additional functions using the `def` keyword in `assignment_4.py`, but you may use anonymous functions as needed.
4. Do not add any include directives or import statements to either file, i.e., do not use any additional libraries.
5. Do not use Python lists.
6. Do not modify any types, function/method implementations, function/method calls or function/method signatures in the files `assignment_4.cpp` or `assigment_4.py` unless asked to as part of an exercise.

Failure to adhere to any of the guidelines (above and below) may result in a reduction of points from your assignment.

Submission Guidelines

The submission deadline for the assignment is **16 April 2025 at 11:59pm**. There will be a late submission penalty of **1 point per 5-minutes late**, where the amount of time late is rounded *up* to the nearest 5 minutes. You should complete the following by the deadline:

- Ensure that you have completed the declaration at the top of `assignment_4.cpp`.
- On Blackboard, for Assignment 4, attach and submit *only* your completed versions of `assignment_4.cpp` and `assignment_4.py`. (Do not zip or otherwise compress the files.) The files *must* be called `assignment_4.cpp` and `assignment_4.py`, respectively.

You can submit as many times as you want, but **only the latest submission will be graded**. You are encouraged to submit early to prevent any issues!

Part 1: C++ (50 pts)

In this part of the assignment, you will be using C++ to implement and use a small linked list library in `assignment_4.cpp`.

Implementing the `LinkedList` Class

Below is the `LinkedList` class, in which you will implement a couple of methods:

```
template <typename T>
class LinkedList {
private:
    struct Node {
        T data;
        Node* next;
        Node(T val) : data(val), next(nullptr) {}
    };

    Node* head;

public:
    LinkedList() : head(nullptr) {}

    // Method to add a new element to the end of a list
    void append(T value) {
        Node* newNode = new Node(value);
        if (!head) {
            head = newNode;
            return;
        }
        Node* temp = head;
        while (temp->next) {
            temp = temp->next;
        }
        temp->next = newNode;
    }

    // Exercise 1 (10 pts)
    // Method to reverse the linked list
    void reverse() {
        // your code here
    }

    // Exercise 2 (10 pts)
    // Method to apply a function to each element of the list
    void foreach(void (*func)(T)) {
        // your code here
    }

    ~LinkedList() {
        Node* current = head;
        Node* nextNode;
        while (current) {
            nextNode = current->next;
            delete current;
            current = nextNode;
        }
    }
};
```

Exercise 1 (10 pts)

Implement the `reverse` method, which when invoked by an instance of `LinkedList`, reverses the order of elements in the instance.
We expect the following two properties to hold:

- For any `LinkedList` instance `list` that represents a list ℓ, after executing the statements `list.reverse(); list.reverse();`, `list` will represent ℓ.
- For any `LinkedList` instance `list` that represents a list ℓ of a length greater than 1 whose elements are distinct, after running `list.inline();`, `list` should *not* represent ℓ.

Exercise 2 (10 pts)

Implement the `foreach` method, which when invoked by an instance of `LinkedList` with a pointer to function `f` as an argument, applies `f` to each element of the list in order, starting with the head of the list.

Using the `LinkedList` Class: Additional Functions

Exercise 3 (5 pts)

In this exercise, you will use the already-defined `printInt` function, which prints an integer followed by a space to implement the function `printList`. The function `printList` should print out each element of its `LinkedList<int>` argument in order starting from the head of the list, where each element is followed by a space:

```
void printInt(int value) {
    std::cout << value << " ";
}
void printList(LinkedList<int> &list) {
    // your code here
}
```

You *must* use the `foreach` method in the `LinkedList` class to implement it.
If we call `print` with an argument `LinkedList` that represents the list [1, 2, 3, 4], we expect the following to be printed out:

```
1 2 3 4
```

Exercise 4 (5 pts)

In this exercise, you will implement the function `printListInc`, which should, for each element of its `LinkedList<int>` argument, in order starting from the head of the list, print out the result obtained from incrementing the element by 1, followed by a space:

```
void printListInc(LinkedList<int> &list) {
    // your code here
}
```

You *must* use the `foreach` method in the `LinkedList` class to implement it.
If we call `printListInc` with an argument `LinkedList` that represents the list [1, 2, 3, 4], we expect the following to be printed out:

```
2 3 4 5
```

Using the `LinkedList` Class: Representing Lists

The file `assignment_4.cpp` contains a `main` function, in which you will make use of the `LinkedClass` class that you just finished implementing to explore what things are allowed by the C++ type system. For each

exercise, you will be asked to create a `LinkedList` instance so that some calls to `append` do not cause any typing errors. If creation is successful, you should invoke the function or method as described, **adding any casting, address-of (`&`) and dereferencing (`*`) operations *on the instance* as necessary to make the file compile**.

Note the following:

- If you can create the instance, there exists a way of using casting, address-of, and dereferencing operations on the instance in order to make the file compile. Depending on how you create the instance, you might not need to use *any* of these operations on the instance.
- The result of invoking these functions or methods *may yield results that are non-intuitive*. In a language with a less permissive type system, these kinds of results may not be possible to get, but with C++ doesn't prevent these kinds of behaviors!

If it is not possible to create an instance where the calls to `append` compile, you should comment out any invocations of `append` that require instance creation to be successful in order to compile. You should also provide a comment explaining why the result will not compile.

Exercise 5 (5 pts)

For some type τ of your choosing, create a `LinkedList<`τ`>` instance called `linkedList1` so that the following `append` method invocations won't cause a compiler error.

```
linkedList1.append(1);
linkedList1.append(2);
linkedList1.append(3);
linkedList1.append(4);
```

If you have successfully created such an instance, then invoke `printList` on `linkedList1` after the final call to `append` by `linkedList1`. Otherwise, comment out the above calls to `append` by `linkedList1` and add a comment explaining why such an instance cannot be created.

Exercise 6 (5 pts)

For some type τ of your choosing, create a `LinkedList<`τ`>` called `linkedList2` so that the following `append` method invocations won't cause a compiler error.

```
linkedList2.append(1.1);
linkedList2.append(2);
linkedList2.append(3.3);
linkedList2.append(4);
```

If you have successfully created such an instance, then invoke `printListInc` on `linkedList2` after the final call to `append` by `linkedList2`. Otherwise, comment out the above calls to `append` by `linkedList2` and add a comment explaining why such an instance cannot be created.

Exercise 7 (5 pts)

For some type τ of your choosing, create a `LinkedList<`τ`>` called `linkedList3` so that the following `append` method invocations won't cause a compiler error.

```
linkedList3.append(0);
linkedList3.append("zero");
linkedList3.append(1);
```

If you have successfully created such an instance, invoke `reverse` from `linkedList3` after the final call to `append` by `linkedList3`. Otherwise, comment out the above calls to `append` by `linkedList3` and add a comment explaining why such an instance cannot be created.

Exercise 8 (5 pts)

For some type τ of your choosing, create a `LinkedList<τ>` called `linkedList4` so that after running the below code, so that the following `append` method invocations won't cause a compiler error.

```
linkedList4.append(0);
linkedList4.append('z');
linkedList4.append(1);
```

If you have succesfully created such an instance, `printListInc` on `linkedList4` after the final call to `append` by `linkedList4`. Otherwise, comment out the above calls to `append` by `linkedList4` and add a comment explaining why such an instance cannot be created.

Compiling and Running C++

You can compile and run your `assignment_4.cpp` file by using the below command-line instructions. It is assumed that you have installed the `g++` C++ compiler already. Note that if you haven't completed the assignment, it may not compile! Comment out lines as needed to check your partially-completed assignment.

On Windows

```
g++ -std=c++17 -o assignment_4.exe assignment_4.cpp
.\assignment_4.exe
```

On Unix-like Systems

```
g++ -std=c++17 -o assignment_4 assignment_4.cpp
./assignment_4
```

Part 2: Python (50 pts)

In this part of the assignment, you will be using Python to implement and use a small linked list library in `assignment_4.py`.

Implementing the `LinkedList` Class

Below is the `Node` and `LinkedList` class, in which you will implement a couple of methods:

```
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None

class LinkedList:
    def __init__(self):
        self.head = None

    # Method to add a new element to the end of a list
    def append(self, value):
        new_node = Node(value)
        if not self.head:
            self.head = new_node
            return
        current = self.head
        while current.next:
            current = current.next
        current.next = new_node

    # Exercise 9 (10 pts)
```

```
    # Method to reverse the linked list
    def reverse(self):
        # your code here

    # Exercise 10 (10 pts)
    # Method to apply a function to each element of the list
    def foreach(self, func):
        # your code here
```

Exercise 9 (10 pts)

Implement the `reverse` method, which when invoked by an instance of `LinkedList`, reverses the order of elements in the instance.

- For any `LinkedList` instance `linked_list` that represents a list ℓ, after executing the below code snippet `linked_list` will represent ℓ.

```
linked_list.reverse()
linked_list.reverse()
```

- For any `LinkedList` instance `linked_list` that represents a list ℓ of a length greater than 1 whose elements are distinct, after running `linked_list.reverse()`, `linked_list` should *not* represent ℓ.

Exercise 10 (10 pts)

Implement the `foreach` method, which when invoked by an instance of `LinkedList` with a function `f` as an argument, applies `f` to each element of the list in order, starting with the head of the list.

Using the `LinkedList` Class: Additional Functions

Exercise 11 (5 pts)

Implement the function `print_list`, which should print out each element of its `LinkedList` argument, in order starting from the head of the list, where each element is followed by a space:

```
def print_list(linked_list):
  # your code here
```

You *must* use the `foreach` method in the `LinkedList` class to implement it. If we call `print_list` with an argument `LinkedList` that represents the list $[1, 2, 3, 4]$, we expect the following to be printed out:

```
1 2 3 4
```

Exercise 12 (5 pts)

Implement the function `print_list_inc`, which should, for each element of its `LinkedList` argument, in order starting from the head of the list, print out the result obtained from incrementing the element by 1, followed by a space:

```
def print_list_inc(linked_list):
  # your code here
```

You *must* use the `foreach` method in the `LinkedList` class to implement it. If we call `print_list` with an argument `LinkedList` that represents the list $[1, 2, 3, 4]$, we expect the following to be printed out:

```
2 3 4 5
```

Using the `LinkedList` Class: Representing Lists

The file `assignment_4.py` contains a `main` function, in which you will make use of the `LinkedClass` class that you just finished implementing to explore what things are allowed by the Python type system. For each exercise, you will be asked to create a `LinkedList` instance such that some calls to `append` run successfully. If creation is successful, you should invoke the function or method as described, **regardless of whether this causes a typing-related error at runtime**.

If it is not possible to create the instance without error, you should instead comment out any invocations of `append` that require instance creation to be successful in order to compile. You should also provide a comment explaining why the result will cause an error.

Exercise 13 (5 pts)

Create a `LinkedList` called `linked_list_1` so that the following `append` method invocations run without error.

```
linked_list_1.append(1)
linked_list_1.append(2)
linked_list_1.append(3)
linked_list_1.append(4)
```

If you have successfully created such an instance, then invoke `print_list` on `linked_list_1` after the final call to `append` by `linked_list_1`. Otherwise, comment out the above calls to `append` by `linked_list_1` and add a comment explaining why such an instance cannot be created.

Exercise 14 (5 pts)

Create a `LinkedList` called `linked_list_2` so that the following `append` method invocations run without error.

```
linked_list_2.append(1.1)
linked_list_2.append(2)
linked_list_2.append(3.3)
linked_list_2.append(4)
```

If you have successfully created such an instance, then invoke `print_list_inc` on `linked_list_2` after the final call to `append` by `linked_list_2`. Otherwise, comment out the above calls to `append` by `linkedList2` and add a comment explaining why such an instance cannot be created.

Exercise 15 (5 pts)

Create a `LinkedList` called `linked_list_3` so that the following `append` method invocations run without error.

```
linked_list_3.append(0)
linked_list_3.append("zero")
linked_list_3.append(1)
```

If you have successfully created such an instance, invoke `reverse` from `linked_list_3` after the final call to `append` by `linked_list_3`. Otherwise, comment out the above calls to `append` by `linked_list_3` and add a comment explaining why such an instance cannot be created.

Exercise 16 (5 pts)

Create a `LinkedList` called `linked_list_4` so that the following `append` method invocations run without error.

```
linked_list_4.append(0)
linked_list_4.append('z')
linked_list_4.append(1)
```

If you have succesfully created such an instance, `print_list_inc` on `linked_list_4` after the final call to `append` by `linked_list_4`. Otherwise, comment out the above calls to `append` by `linked_list_4` and add a comment explaining why such an instance cannot be created.

Running Python

Assuming you have Python installed already, you can run your completed `assignment_4.py` on the command line as follows:

```
python assignment_4.py
```