Template and Linked List Sorting

Skeleton Project

- A solution file containing
  - The Linked List class mentioned in lecture
    - simpleLinkedListTemplate.h
    - simpleLinkedListTemplate.cpp
  - A main file to use the Linked List:
    - main.cpp
  - And the files for class Food:
    - food.h
    - food.cpp

Files to be Modified

- You will only submit the modifications in these two files ONLY:
  - simpleLinkedListTemplate.h
  - simpleLinkedListTemplate.cpp
- However, you can modify other files
  - But in our grading, we will assume all the other files are the original given ones
  - More precisely, you are only allowed to ADD more helping members or methods and you should NOT change the existing implemented declarations and implementations

In Fact

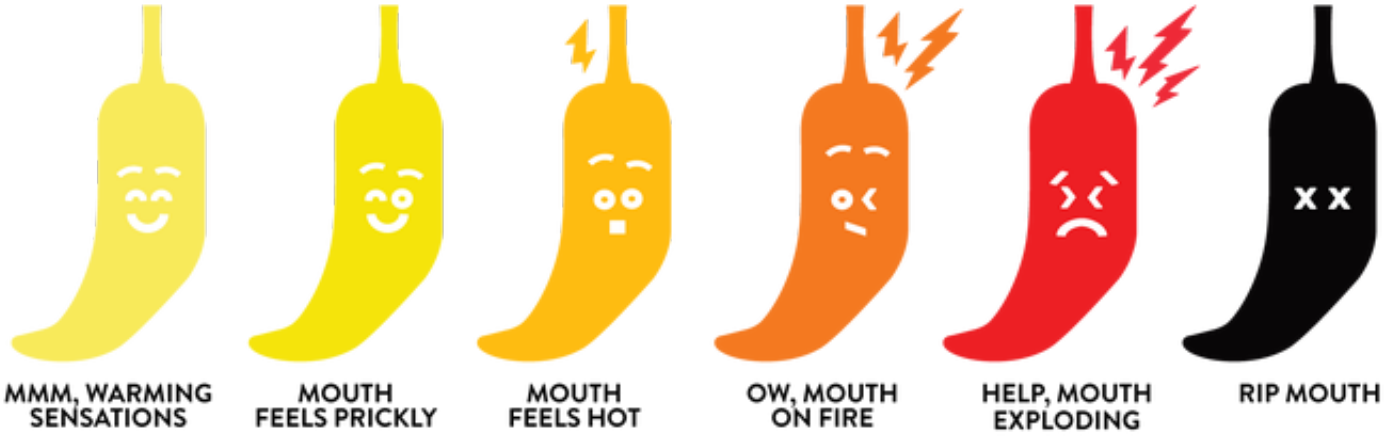
- You should modify the following functions in simpleLinkedListTemplate.cpp ONLY:
  - exist()
  - extractMax()
  - reverseOp()
- And, add (but not modify or subtract) the class definition in simpleLinkedListTemplate.h

main.cpp

```
int main()
{
  testIntLL();
  testFoodExist();
  testFoodOpGreaterThan();
  testFoodAddition();
  testFoodSort();
  testReverseOp();
  return 0;
}
```

Your Tasks

Ranging from



Task 1

- The member function exist() is missing. Your job is to implement it in the same way you did in the previous assignment
- Sample output:

```
Testing List<int>
This is the linked list we have:
20 1 9 11 123

Testing the function exist()
The number 10 is not in the array
The number 20 is in the array
The number 30 is not in the array
The number 40 is not in the array
The number 50 is not in the array
```

Task2

- Task 2 is to implement the operation “+” for food
- In the function `testFoodAddition()`

```
Food food1("Salad", 100);
Food food2("Chicken", 200);
Food food3("Curry", 40);
Food food23 = food2 + food3;
```

- so the name of **food23** will be Chicken Curry with 240 calories


```
Testing operator "+" for class Food
Curry Chicken Ice Cream Ice Cream with 840 calories
Curry Salad with 140 calories
Chicken Salad with 300 calories
Chicken Curry with 240 calories
Ice Cream with 300 calories
Curry with 40 calories
Chicken with 200 calories
Salad with 100 calories
```

Task 2

- Remember we implement the operator “>” for the class Food?

```
class Food {
private:
  string _name;
  int _cal;
public:
  Food() { _name = ""; _cal = 0; };
  Food(string, int);
  Food operator+(const Food&);
  bool operator>(const Food&);
  bool operator==(const Food&);
  string name() { return _name; };
  int cal() { return _cal; };

  friend ostream &operator<<(ostream&, const Food&);
};
```

Task 2

- Implementation of “>” for food

```
bool Food:: operator>(const Food& f)
{
   return _cal > f._cal;
}
```

- So, Task 2 is to the same thing for the operation “+” for food

Implemented functions

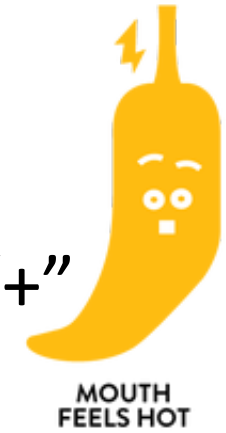
- All these should work because the function exist() and operator “>” are implemented

```
Testing exist for List<Food>
The food "Fish" exists in the list
The food "Banana" does not exist in the list
The food "Fish Soup" does not exist in the list

Testing operator ">" for class Food
Among Salad and French Fries...
The food with more calories is French Fries with 10000 calories
```

- Details in the two functions:
  - `testFoodExist();`
  - `testFoodOpGreaterThan();`

Task2



- Task 2 is to the same thing for the operation “+” for food
- In the function `testFoodAddition()`

```
Food food1("Salad", 100);
Food food2("Chicken", 200);
Food food3("Curry", 40);
Food food23 = food2 + food3;
```

- so the name of **food23** will be Chicken Curry with 240 calories

```
Testing operator "+" for class Food
Curry Chicken Ice Cream Ice Cream with 840 calories
Curry Salad with 140 calories
Chicken Salad with 300 calories
Chicken Curry with 240 calories
Ice Cream with 300 calories
Curry with 40 calories
Chicken with 200 calories
Salad with 100 calories
```

Task 3

- To implement the function `extractMax()` in List<T>

```
template <class T>
T List<T>::extractMax()
{

   // if there are duplicates maximas in the
   list, return the leftmost one (the one
   closest to the _head)

   return T();
}
```

Task 3

- To implement the function `extractMax()` in `List<T>`
- Function `extractMax()` will
  - Find the maximum item in the list
  - delete the node in the list and return the maximum item
- Sample Output:

```
Testing Extract Maximum for List<int>
This is the linked list we have:
20 1 9 11 123

After one extractMax()
20 1 9 11

After another one extractMax()
1 9 11
```

Task 3

- If you have done it right
  - You can uncomment the code in the function `testIntLL()`

```
while (!l.empty())
   cout << l.extractMax() << " " ;
cout << endl;
```

- And it will do a sorting for you!

Task 3

- And it should work for the class Food also!

```
Here is the list of food stored, according to the list order from head to tail:
Soup with 50 calories
Veggies with 100 calories
Fish with 100 calories
Salad with 50 calories
Chicken Chop with 100 calories
Pork Chop with 150 calories
Chocolate with 200 calories
Rice with 500 calories
Beef with 300 calories


The sorted list of food in decending order is:
Rice with 500 calories
Beef with 300 calories
Chocolate with 200 calories
Pork Chop with 150 calories
Veggies with 100 calories
Fish with 100 calories
Chicken Chop with 100 calories
Soup with 50 calories
Salad with 50 calories
```

- If you implement the “>” for food

Task 4

- Write down your thoughts
  - Not in code this time. Phew…
- What are the disadvantages of the “LinkedList extractMax Sorting”?
  - List at least two disadvantages in coursemology
- We suggest you to do this **after** you finish all the coding.
  - Use this lab session to code more and ask questions

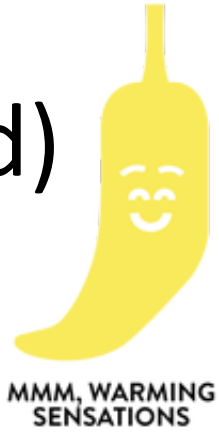
Task 5

- Implement the function `reverseOp()`
  - It will reverse the order of the items in the list
- Sample output from the function `testReverseOp()`:

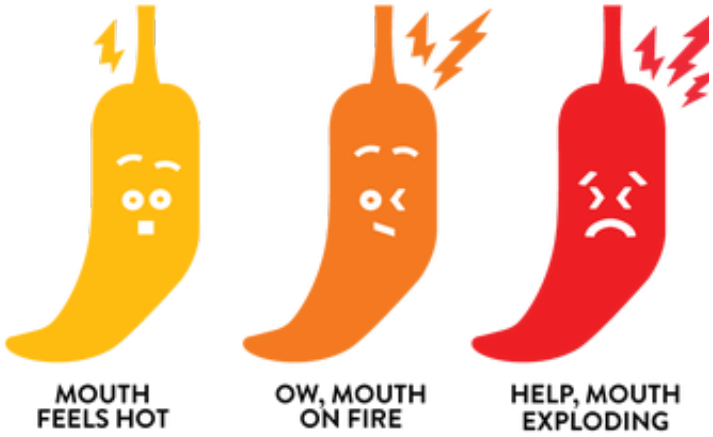
```
Testing reverseOp()
This is the linked list we have:
20 1 9 11 123
This is the linked list after reverseOp():
123 11 9 1 20
This is the linked list after reverseOp() again:
20 1 9 11 123
This is the linked list after reverseOp() again and again
123 11 9 1 20
This is the linked list after reverseOp() again and again and again
20 1 9 11 123
```

- Each time you call `l.reverseOp()` to modify "itself"

Extra Tasks You can do (Not Graded)



- Implement the “==“ operators for the class Food
- Google how did we do “`cout << food;`”
- Implement another class to use it with the LinkedList Template



For example

- create a class called “Hero”

```
class Hero {
private:
   string _name;
   int _str, _dex, _wis;
public:
   Hero() ;
   Hero(string name,int str,int dex,int wis) ;
   bool operator>(const Hero& hero) ;
};
```

The hero’s name

The parameters of the hero, namely strength, dexterity and wisdom

Implement “>” for hero by comparing the sum of their parameters

```
void testHeroSort()
{
  cout << "Test Hero So

  List<Hero> l_hero;
  l_hero.insertHead(Hero("Iron Man", 20, 20, 20));
  l_hero.insertHead(Hero("Thor", 40, 30, 10));
  l_hero.insertHead(Hero("Black Widow", 10, 40, 40));
  l_hero.insertHead(Hero("Green Hulk", 50, 30, 10));
  l_hero.insertHead(Hero("Captain America", 15, 30, 35));

  cout << "The original list of heros" << endl;
  l_hero.print();

  cout << endl << "The sorted list of heros" << endl;

  while (!l_hero.empty())
    cout << l_hero.extractMax() << endl;
}
```

```
Test Hero Sorting
The original list of heros
Captain America Green Hulk Black Widow Thor Iron Man

The sorted list of heros
Green Hulk
Black Widow
Captain America
Thor
Iron Man
```