Занятие 14. Списки - 3

Задание 1

Разработайте класс `List` для хранения односвязных списков целых чисел. Объект класса `List` не является узлом списка – узлы реализуйте с помощью отдельного класса (или структуры) `Node`, объявленной внутри класса `List`.

Класс `List` должен обеспечить:

- хранение указателя на начало списка (список может быть с выделенной головой или без неё);

  для хранения указателей на голову (и на следующий элемент в `Node`) должны быть использованы указатели std::unique_ptr. При использовании std::shared_ptr будут снижены баллы за задание;
- методы для работы со списком:
  - `void clear()` – удаляет все элементы;
  - `void push_back()` – добавляет элемент в конец списка;
  - `void insert_sorted(int newVal)` – добавляет элемент таким образом, что если исходный список был отсортирован, то он бы остался отсортированным после вставки элемента.
  - оператор вывода списка в поток;
  - `operator == (const List &other)` для сравнения списков.

Для тестирования кода на этом этапе используйте оператор вывода в поток.

Задание 2

Реализуйте класс итератора для списка. Для этого:

1) определите класс итератора `Iterator` внутри класса `List`. Этот класс должен:
   a) хранить обычный указатель на текущий узел списка;
   b) иметь два конструктора: один без аргументов (следует установить хранимый указатель в `nullptr`); второй получает указатель на элемент;
   c) определять операторы:
      i) `Iterator & operator ++` – переходит на следующий элемент списка;

ii) `bool operator == (const Iterator other)`, `bool operator != (const Iterator other)` – для сравнения элементов;

iii) `int& operator * ()` – возвращает ссылку на данные текущего элемента списка;

2) реализуйте в классе `List` методы `Iterator begin()` и `Iterator end()`, которые должны возвращать итераторы на начало и конец списка. В первом случае воспользуйтесь конструктором итератора, которому передаётся адрес на начало списка. Во втором – конструктором итератора по умолчанию, который установит текущий итератор в `nullptr`. В данном случае удобно использовать `nullptr` как признак конца списка, так как именно это значение получится в итераторе, если перейти за конец списка.

Проверьте, что разработанный итератор работает с алгоритмами stl.

Дополнительное задание 3

Реализуйте класс константного итератора, который должен обеспечивать доступ к элементам списка, но не давать его менять. Для этого нужно создать аналогичный разработанному при решении задания 2 класс `ConstIterator` и методы `cbegin()/cend()`, которые отличаются следующим:

- методы `cbegin()/cend()` являются константными;
- `operator *` возвращает ссылку на константу.

Оценка задания

Задание 1 с использованием `unique_ptr`	50%.
Задание 1 с использованием `shared_ptr`	25%.
Задание 2	50%.
Задание 3	10%.

Баллы за каждую задачу включают:

- правильность решения задания 60 %;
- оформление кода согласно http://prog.tversu.ru/pr3/codeStyle.pdf 10 %;
- программа компилируется без предупреждений компилятора на 4-ом уровне 10%;
- модульные тесты 20%.