Постановка задачи

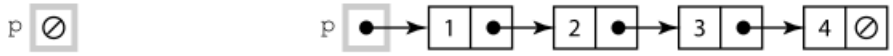
Требуется реализовать односвязный список. В списке должны быть следующие методы:

1. `void push_back(T value)` – добавляет элемент в конец списка;
2. `void push_front(T value)` – добавляет элемент в начало списка;
3. `void insert(size_t idx, T value)` – добавляет элемент по индексу;
4. `void pop_back()` – удаляет последний элемент в списке;
5. `void pop_front()` – удаляет первый элемент в списке;
6. `void remove_at(size_t index)` – удаляет элемент в списке по указанному индексу;
7. `T& operator[](const size_t index)` – для записи элемента по указанному индексу;
8. `T const& operator[](const size_t index) const` – для чтения по индексу;
9. `size_t size() const` – возвращает количество элементов в списке;
10. `bool empty() const` – отвечает на вопрос пустой ли список;
11. `void clear()` – очищает список, удаляет все узлы;
12. `T front() const` – возвращает первый элемент списка;
13. `T back() const` – возвращает последний элемент списка.
14. Для списка реализовать класс `ListIterator` и добавить к списку поддержку итераторов `begin()` и `end()`. Проверить работу соответствующим кодом.

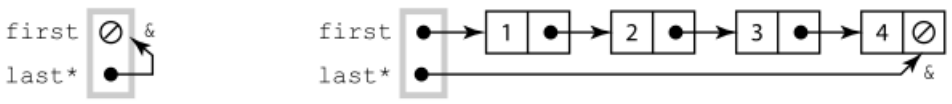
```
LList<int> l = { 3, 5, 2, 7 };
for (auto &i : l) i += 2;
auto lambda = [&l](int a, int b) { return a + b*10; };
std::cout << std::accumulate(l.begin(), l.end(), 0, lambda) << "\n";//250
```

Внутренне представление списка должно соответствовать одному из представленных вариантов. Необходимо выполнить только один вариант списка. Ваш вариант указан в таблице с посещаемостью группы. Обратите внимание, что структура узла во всех вариантах идентична. На рисунках представлен пустой (слева) и заполненный список (справа). Варианты представлены ниже.

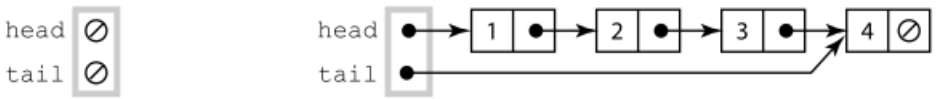
1. Указатель на первый узел, в последнем узле указатель на `nullptr`.



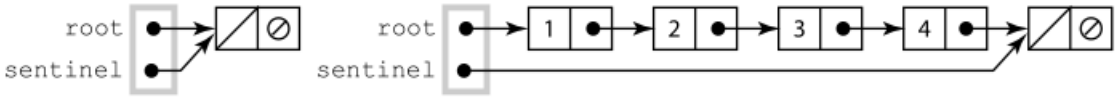
2. Указатель на первый узел. Указатель `last` содержит адрес последнего узла, как и указатель `next` последнего узла.



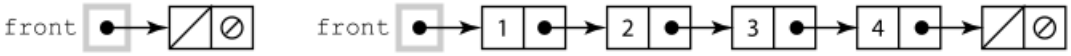
3. Указатель на первый и последний узел.



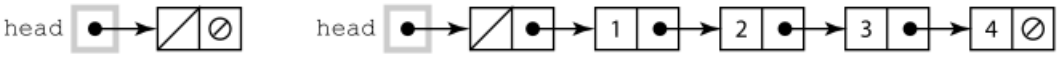
4. Два указателя, дополнительный служебный узел в конце списка.



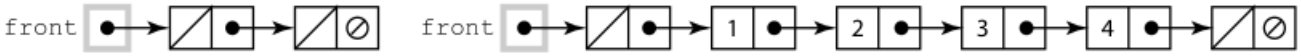
5. Дополнительный служебный узел в конце списка.



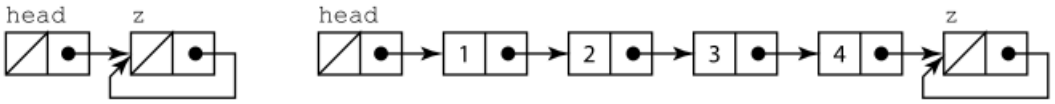
6. Дополнительный служебный узел в начале списка.



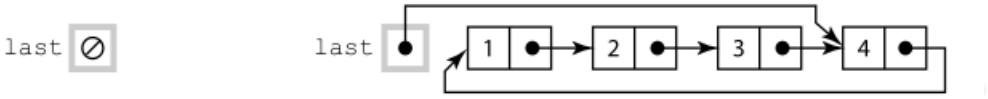
7. Два дополнительных служебных узла.



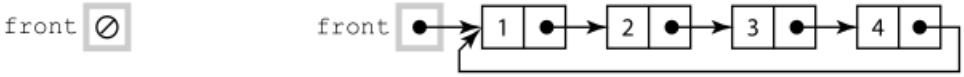
8. Дополнительный служебный узел указывает сам на себя.



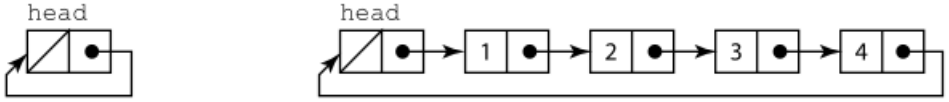
9. Циклический список, указатель на последний узел.



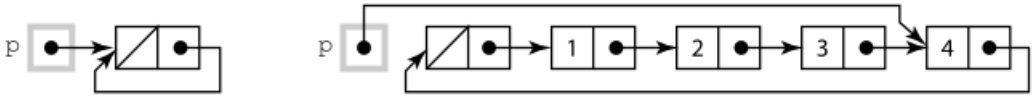
10. Циклический список, указатель на первый узел.



11. Циклический список с дополнительным служебным узлом.



12. Циклический список со служебным узлом, указатель на последний узел.



```cpp
void print_lst(const LList<char> &l) {
  // вывод списка на экран с помощью l[i]
}
int main() {
  LList<char> lst; // ваш список
  std::cout << std::boolalpha << lst.empty() << std::endl;

  for(int i = 0; i < 5; i++)
    lst.push_back(char('a' + i));

  print_lst(lst);

  for(int i = 0; i < 5; i++)
    lst.insert(0, char('z' - i));

  print_lst(lst);

  for(size_t i = 0; i != lst.size(); i++)
    lst[i] = char('a' + i);

  print_lst(lst);

  lst.pop_back();
  lst.pop_front();

  print_lst(lst);

  lst.remove_at(5);
  lst.insert(3, 'o');

  print_lst(lst);

  lst.clear();

  lst.push_back('q');
  lst.push_back('w');

  std::cout << lst.size() << ' ' << std::boolalpha << lst.empty() << std::endl;
} // деструктор освободит оставшиеся узлы
```

Результат работы

Пример с использованием всех реализованных методов должен работать без ошибок. Вывод должен соответствовать примеру ниже.

Вывод приложения

```
true
a -> b -> c -> d -> e
v -> w -> x -> y -> z -> a -> b -> c -> d -> e
a -> b -> c -> d -> e -> f -> g -> h -> i -> j
b -> c -> d -> e -> f -> g -> h -> i
b -> c -> d -> o -> e -> f -> h -> i
2 false
```

Файл с исходным кодом добавьте себе в репозиторий.