#2: Dynamic Array

Одной из важнейших задач при разработке программ является задача организации хранения и предоставления доступа к данным. Эта задача решается при помощи различных контейнеров данных, обеспечивающих управление памятью, необходимой для хранения данных и предоставляющих интерфейсы для доступа к хранимым данным.

Задание

В вашу задачу входит реализация АТД динамического массива:

```
template<typename T>
class Array final
```

должен поддерживать следующие операции:

- `Array();`
  `Array(int capacity);`
  Конструктор распределяет память, необходимую для хранения некоторого количества элементов. Конструктор без параметров использует значение емкости по умолчанию (например, 8 или 16), конструктор с параметром capacity – явно переданное значение.
- `~Array();`
  Деструктор освобождает память, выделенную под хранение элементов. При необходимости, при освобождении памяти вызываются деструкторы хранимых элементов.
- `int insert(const T& value);`
  `int insert(int index, const T& value);`
  Вставляет переданное значение в конец массива, или в указанную позицию, увеличивая размер массива на 1 и, при необходимости, сдвигая существующие элементы вправо. Возвращает индекс, в который был вставлен элемент. Если памяти для добавления элемента недостаточно, то перераспределяет память, копируя уже существующие элементы в новую область. Затем старый блок освобождается. Для копирования и сдвигания элементов необходимо использовать move-семантику. (Если тип T ее

поддерживает. В противном случае необходимо использовать копирование с последующим вызовом деструктора). Выделение памяти происходит каждый раз с увеличением в 1.6..2 раза относительно текущего размера.

- `void remove(int index);`
  Удаляет элемент из указанной позиции массива, сдвигая остальные элементы влево (сдвиги выполняются аналогично `insert()`, но память при этом не освобождается).

- `const T& operator[](int index) const;`
  `T& operator[](int index);`
  Оператор индексирования позволяет обратиться к элементу массива по индексу для чтения и для записи.

- `int size() const;`
  Возвращает текущий размер (количество реально существующих в массиве) элементов.

- `Iterator iterator();`
  `ConstIterator iterator() const;`
  Возвращает итератор, указывающий на первый элемент массива. При помощи такого итератора массив можно обойти в прямом порядке: от первого элемента до последнего.

- `Iterator reverseIterator();`
  `ConstIterator reverseIterator() const;`
  Возвращает итератор, указывающий на последний элемент массива. При помощи такого итератора массив можно обойти в обратном порядке: от последнего элемента к первому.

Обратите внимание, что по правилам C++ такой массив может копироваться и присваиваться, однако поведение копирования и присваивания по умолчанию приведет к проблемам, связанным с тем, что оба экземпляра будут использовать один и тот же блок памяти, и пытаться освобождать его в деструкторе. Поэтому вы должны предоставить правильный механизм копирования и перемещения.

Функции работы с индексом не должны выполнять проверку границ. Но в отладочной версии такие проверки можно добавить используя механизм утверждений.

Управление памятью должно быть реализовано через функции `malloc()` и `free()`.

Итератор должен быть реализован как внутренний для Array<T> класс, и предоставлять следующий интерфейс:

- `const T& get() const;`
  Получает значение массива в текущей позиции итератора.
- `void set(const T& value);`
  Устанавливает значение в текущей позиции итератора.
- `void next();`
  Перемещает текущую позицию итератора на следующий элемент.
- `bool hasNext() const;`
  Возвращает `true`, если итератор может перейти к следующему элементу, или `false` в противном случае.

Пример использования: поместим в массив числа от 1 до 10, умножим каждое на 2, затем выведем их на экран.

```
Array<int> a;
for (int i = 0; i < 10; ++i)
  a.insert(i + 1);

for (int i = 0; i < a.size(); ++i)
  a[i] *= 2;

for (auto it = a.iterator(); it.hasNext(); it.next())
  std::cout << it.get() << std::endl;
```

Для разработанных классов реализовать модульные тесты на базе Google Test или CppUnit.

(*) Дополнительно для итератора можно перегрузить оператор разыменовывания и оператор автоинкремента (в префиксной и постфиксной формах).

(*) Дополнительно можно реализовать функции `begin()`/`cbegin()` и `end()`/`cend()` для использования такого массива в цикле `range-for`.