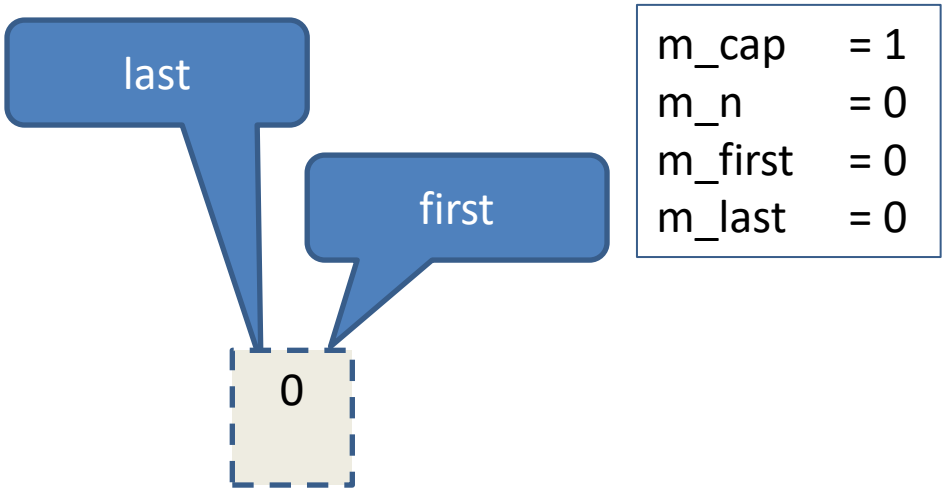
Работа с круговым буфером

Вспомогательные материалы
для лабораторный работы
по курсу С220

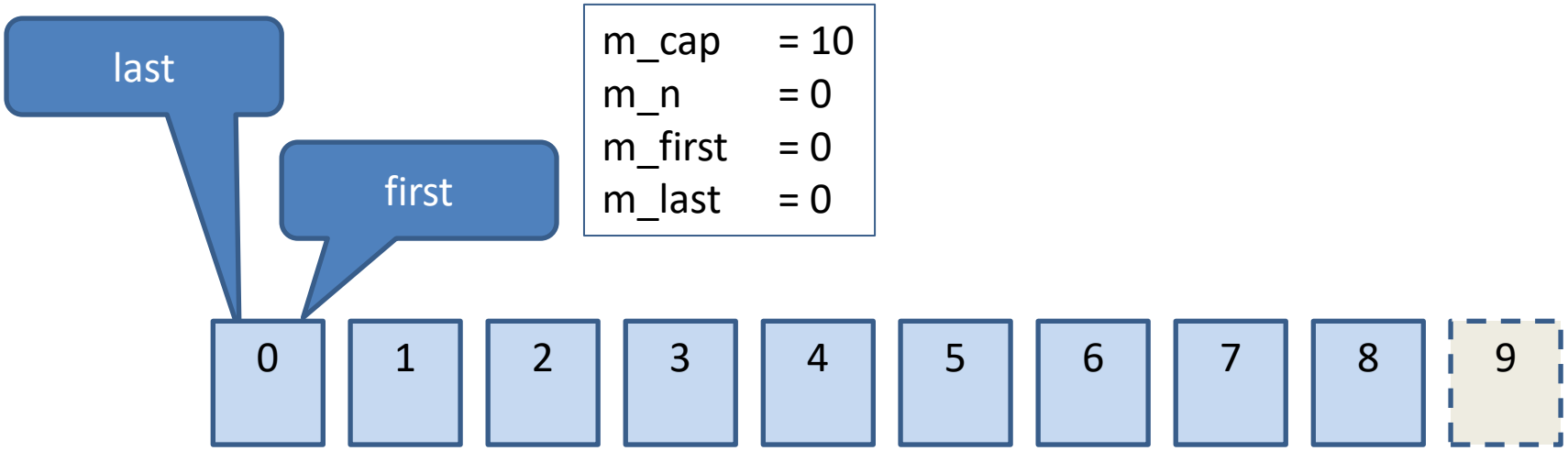
Создание кругового буфера



Для корректного прохода по буферу с помощью range-based for ***first*** и ***last*** должны отличаться друг от друга как минимум на один элемент. Буфер будет полностью заполнен при ***m_n == m_cap – 1***.

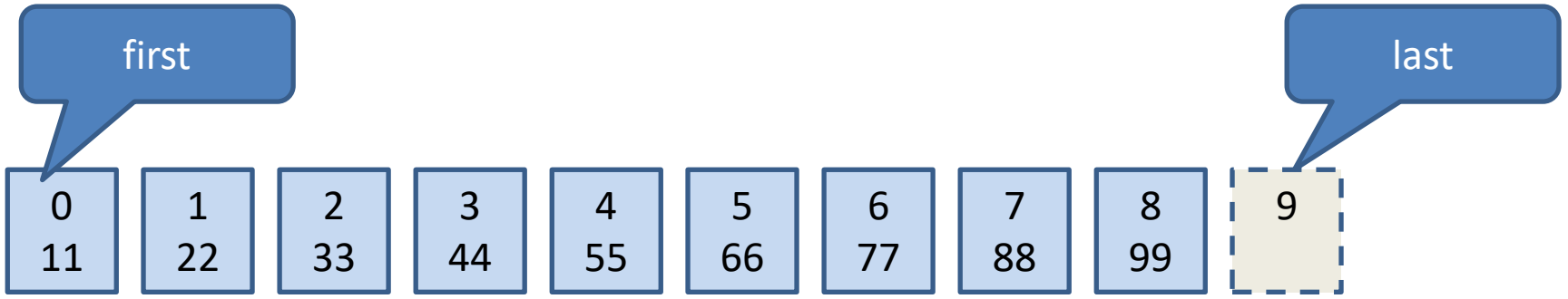
Пока память можно не выделять – этот элемент не хранит данные.

Создание кругового буфера



А можно сразу выделить память, например, под **10** элементов.
Последний элемент не будет хранить данные.

Заполнение кругового буфера

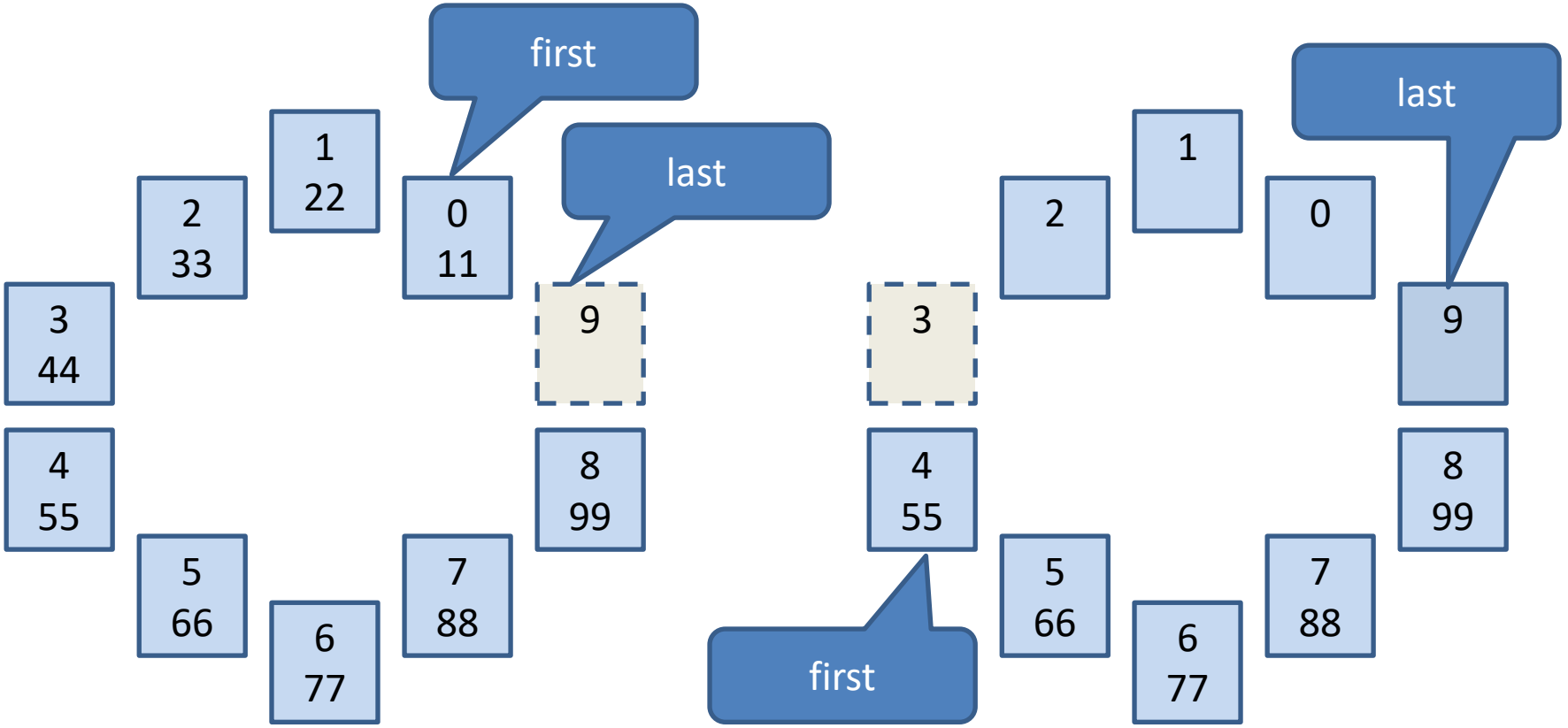


m_cap = 10
m_n = 9
m_first = 0
m_last = 9

m_last = m_first + m_n

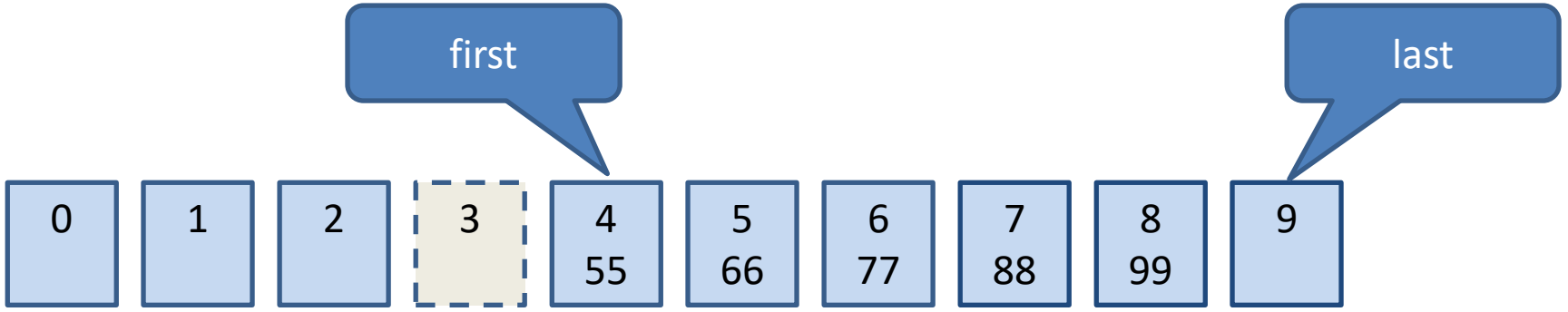
Буфер полон - ***m_n == m_cap - 1***. Чтобы в буфер можно было заносить новые значения, нужно увеличить объем выделенной памяти. Но извлекать значения из буфера можно.

Чтение кругового буфера



Изменение состояния буфера после извлечения четырех элементов.

Чтение кругового буфера

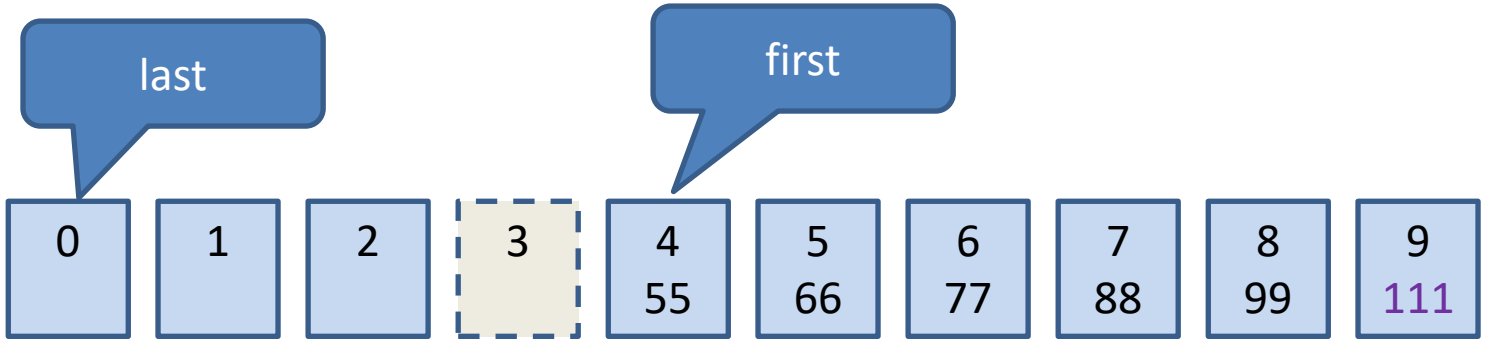


m_cap =10
m_n = 5
m_first = 4
m_last = 4+5= 9

m_last = m_first + m_n

Прочитанные значения удаляются из буфера, освобождая соответствующие им элементы. Данные в динамическом массиве пока не закольцованы.

Заполнение кругового буфера (переход на начало)



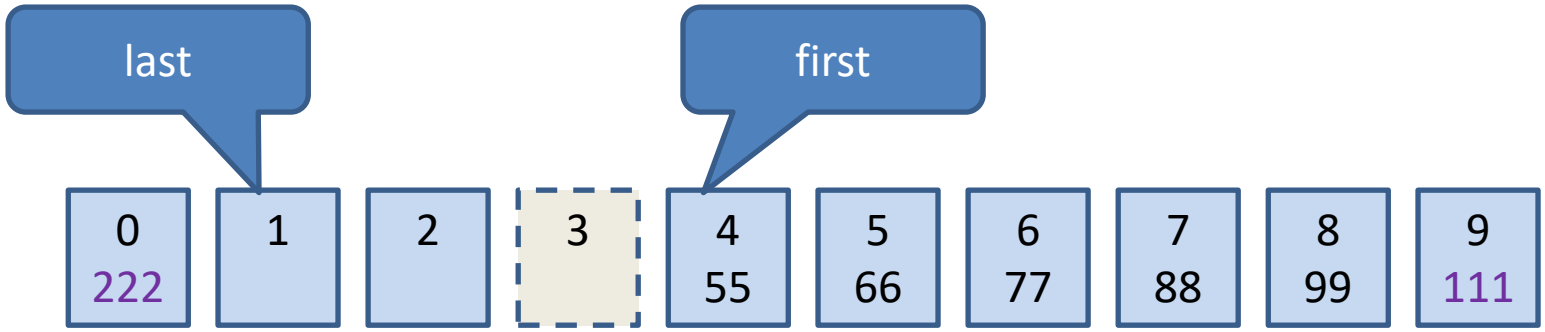
m_cap = 10
m_n = 6
m_first = 4
m_last = (4+6)%10 = 0

~~m_last = m_first + m_n~~

m_last = (m_first + m_n) % m_cap

Индекс следующего свободного элемента оказывается в начале динамического массива.

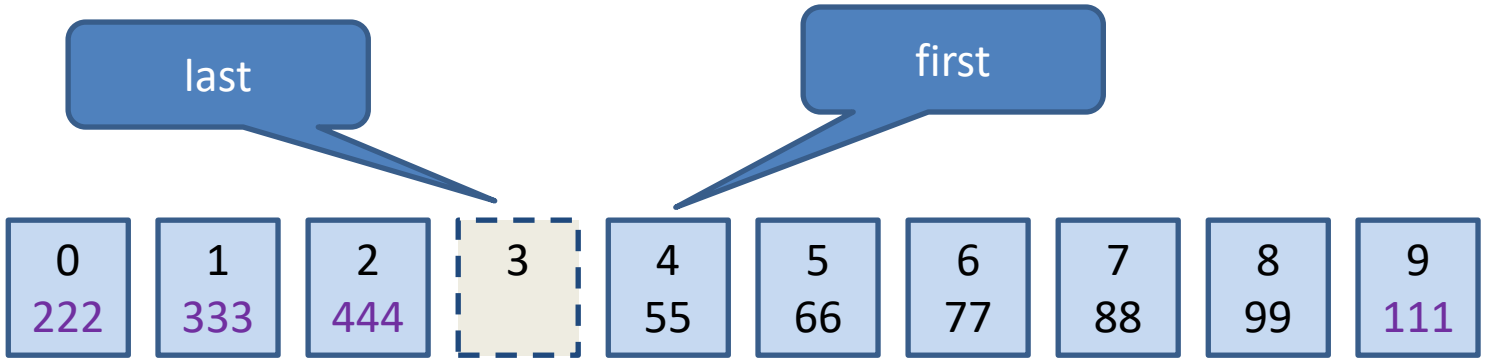
Заполнение кругового буфера (переход на начало)



m_cap = 10
m_n = 7
m_first = 4
m_last = (4+7)%10 = 1

m_last = (m_first + m_n) % m_cap

Заполнение кругового буфера (переход на начало)

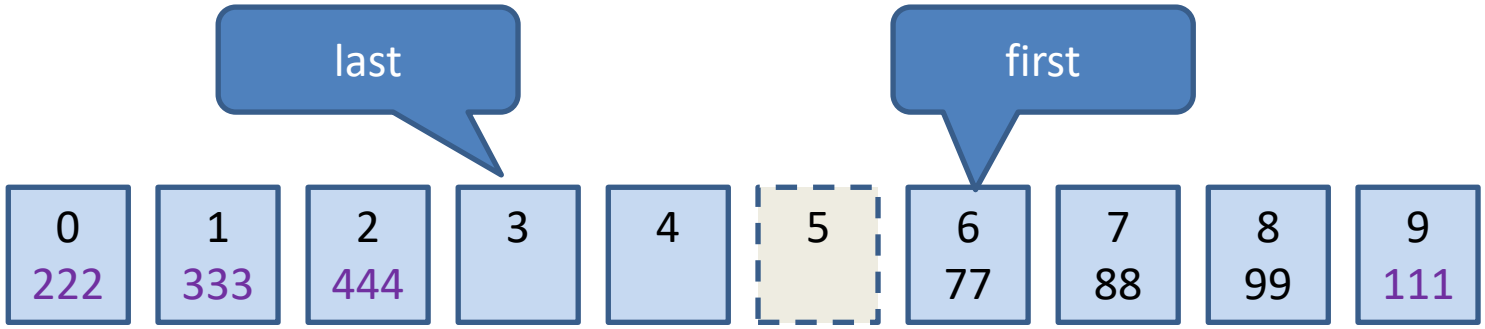


m_cap = 10
m_n = 9
m_first = 4
m_last = (4+9)%10 = 3

m_last = (m_first + m_n) % m_cap

Буфер полон - ***m_n == m_cap – 1.*** Для занесения новых значений нужно перераспределить память (увеличить размер динамического массива). Но извлекать значения из буфера можно.

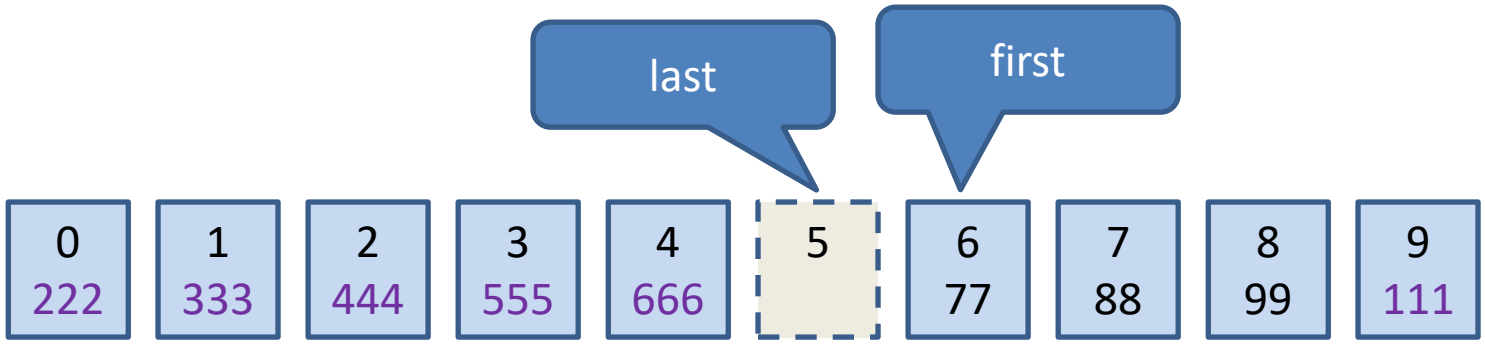
Чтение кругового буфера (переход на начало)



m_cap = 10
m_n = 7
m_first = 6
m_last = (6+7)%10 = 3

m_last = (m_first + m_n) % m_cap

Перебор элементов кругового буфера



m_cap = 10
m_n = 9
m_first = 6
m_last = (6+9)%10 = 5

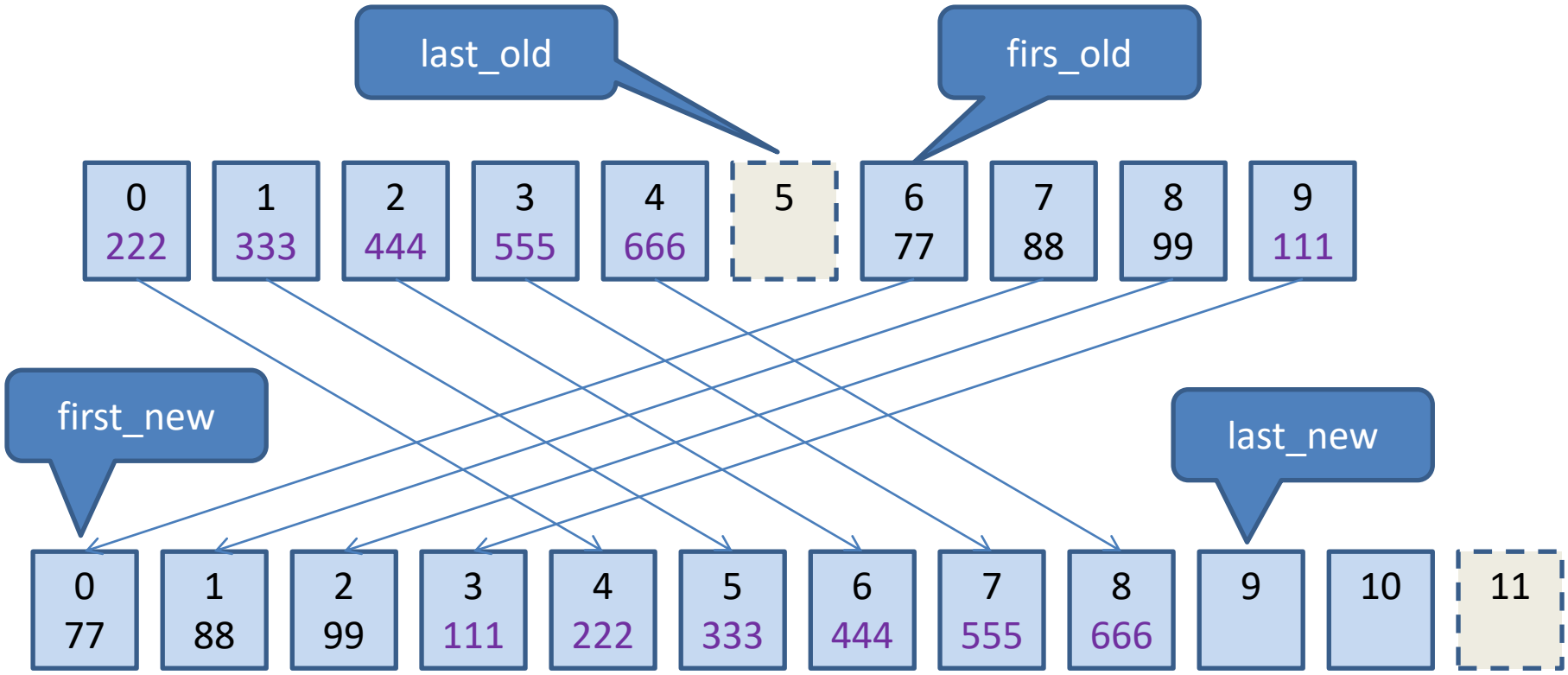
m_last = (m_first + m_n) % m_cap

Перебор элементов:

```
for (int i = 0;  i < m_n; i++)  {

   …  buf [ ( m_first + i ) % m_cap ];  // доступ к i-му элементу кругового буфера
}
```

Увеличение размера кругового буфера



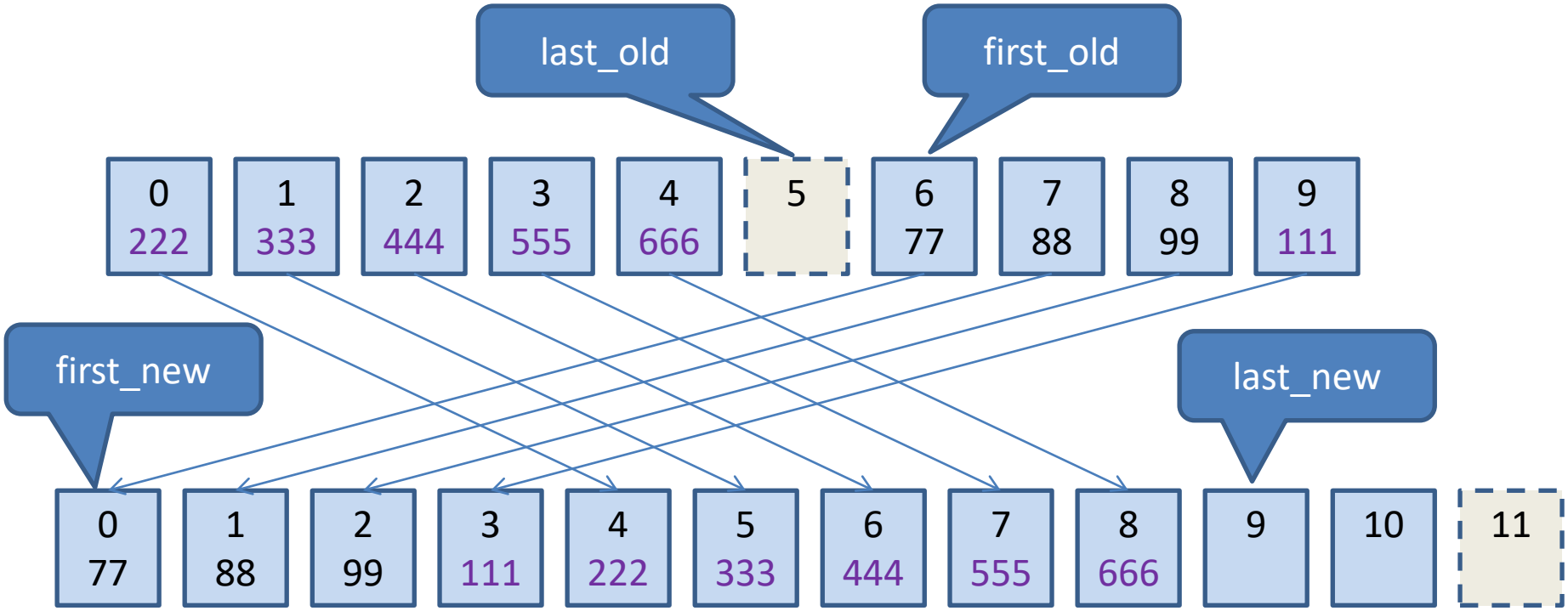
m_cap_old = 10
m_n _old = 9
m_first_old = 6
m_last _old = 5

m_n_old == m_cap_old – 1 (буфер полон)

- увеличиваем размер дин. массива,
- «разворачиваем» данные.

m_cap_new = 12
m_n_new = 9
m_first_new = 0
m_last_new = 9

Перезапись кругового буфера (копирование)




```
for (int i = 0;  i < m_n_old; i++)
{
      buf_new[ i ] = buf_old[ ( m_first_old + i ) % m_cap_old ];
}
```

При перезаписи данных из старого динамического массива в новый данные «разворачиваем», копируем от ***first_old*** до ***last_old***.

Реализация очереди с кольцевым буфером

Замечания к представляемому решению:

1. В примере очередь по умолчанию создается без резерва (память не резервируется, емкость равна 0 или 1). Но! Никто не мешает заготовить некоторое количество резервных элементов (заданное, например, параметром конструктора).

2. Чтобы в дальнейшем (согласно заданию) можно было использовать такую очередь в диапазонном for, требуется ввести итератор и предоставить методы для получения итераторов на начало (begin()) и на конец (end()) очереди. Чтобы итераторы на начало и на конец очереди можно было различать, в массиве всегда должен быть хотя бы один резервный элемент! То есть емкость очереди должна быть больше размера, в противном случае требуется перераспределение памяти.

Класс очереди

```
template<typename T> class MyQueue
{
        T* m_p{};                   // указатель на начало динамического массива
        size_t m_n{};               // актуальное количество элементов в очереди
        size_t m_cap{1};            // емкость (сколько выделено памяти)
        size_t m_first{};           // индекс первого элемента в очереди (это тот элемент, который
                                    // можно извлечь из очереди с помощью pop())
        size_t m_last{};            // индекс первого свободного элемента в очереди (это  тот элемент, в
                                    // который можно присвоить новое значение с помощью push())
        const size_t delta{1};      // на сколько увеличиваем емкость при перераспределении памяти

public:
        class iterator {
            // данные и методы, реализующие функциональность итератора для кольцевой очереди
              . . .
         };

        iterator begin ( ) const  { . . . }    // итератор на начало очереди
        iterator end ( ) const    { . . . }    // итератор на конец очереди
```

Класс очереди (продолжение)

```
// так как класс сложный, реализуем «джентльменский» набор:
MyQueue ( );
~MyQueue ( );

MyQueue ( const MyQueue& );
MyQueue ( MyQueue&& );

MyQueue& operator= ( const MyQueue& );
MyQueue& operator= ( MyQueue&& );
        ...

 // методы, реализующие функциональность очереди
 void push (...);
 T pop ( );
        ...

// а также, методы, необходимые для выполнения задания
        ...
};
```

Класс итератора (возможная реализация)

```
template<typename T> class MyQueue
{           ...
            class iterator
            {
                        const MyQueueIt* m_pQ;  // итерируемая очередь
                        int m_i;                // индекс текущего элемента
            public:
                        iterator(…) { … }

                        // реализуем методы, которые использует range-based for
                        iterator& operator++ ( ) { … }      // оператор префиксного инкремента
                         T& operator* ( ) { … }             // оператор разыменования
                        bool operator!= ( const iterator& ) const { … }  // оператор сравнения

                        // может быть потребуется что-то еще
                         ...
            };
            ...
};
```

Проверка разработанного класса кольцевой очереди

```
// Следующий фрагмент должен работать не только КОРРЕКТНО, но и ЭФФЕКТИВНО:

MyQueue<MyString>  q1 { MyString("AAA"), MyString ("qwerty"),
                        < другие_инициализаторы>  };   // конструктор по списку инициализации

for (const auto& el : q1)  {  std::cout << el << ' '; }   // range-based for

MyString s ( "abc" );
q1.push ( s );
q1.push ( MyString ("123") );  // rvalue push
MyString s1 = q1.pop();
q1.push ( "qqq" );
MyQueue < MyString >  q2 = q1;
MyQueue < MyString >  q22 = std::move(q1);

MyQueue < MyString >  q3 { 10, MyString ("!") };  // очередь должна содержать 10 элементов
                                                  // со строкой «!»
q1 = q3;
q2 = MyQueue < MyString >  ( 5, MyString ("?") );
q1 = { MyString ( "bbb" ), MyString ("ssss") };      // оператор присваивания по списку инициализации
```

Замечание:
если вы не разрабатывали ранее класс MyString можно использовать класс std::string