



Лекция 29
26 ноября 2019 года

**Реализация очереди с
использованием ООП**

Что такое очередь?

Очередью называется структура данных, содержащая последовательность однотипных элементов и позволяющая эффективно выполнять следующие операции:

- **вставка** нового элемента в **хвост** очереди;
- **удаление** из очереди элемента, находящегося в **голове**;
- **просмотр** элемента, находящегося в **голове** очереди;
- **определение количества** элементов в очереди (или определение того, пуста ли очередь)

Нестандартная очередь

Над очередью иногда можно выполнить дополнительные нестандартные для очереди операции :

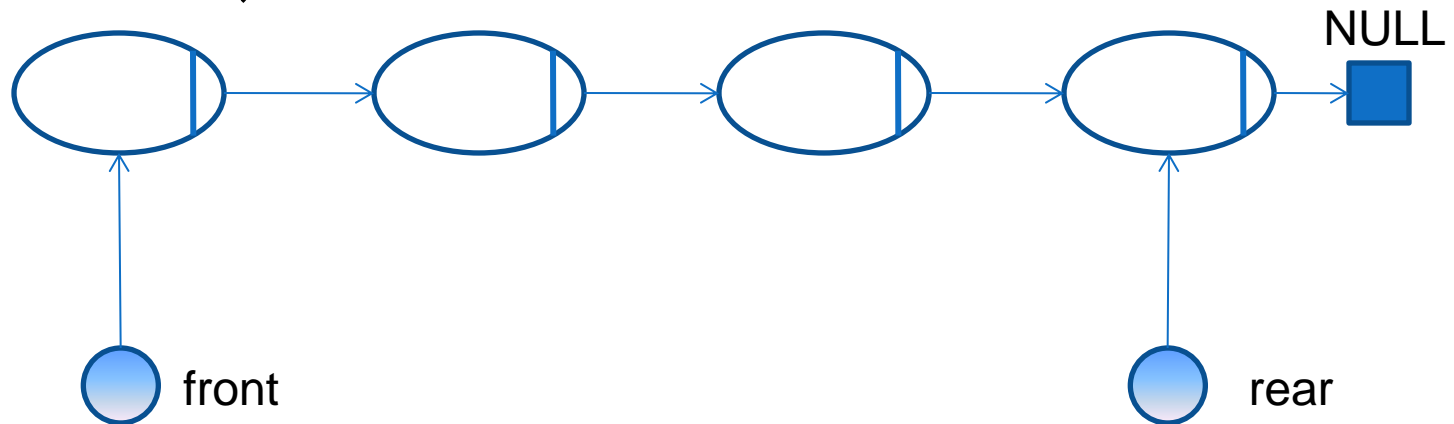
- **доступ к элементу очереди по его номеру** (элемент, находящийся в голове очереди, имеет номер 0);
- **выполнение однотипных действий** над всеми элементами очереди.

Реализация очереди

Существует несколько способов реализации очереди. Основные способы:

- на массивах (рассматривали ранее циклическую очередь);
- на списках

Рассмотрим второй способ реализации очереди (линейной).



Описание очереди (файл queue.h)

Часть 1 (предварительные описания)

```
#ifndef __LQueue_defined__
#define __LQueue_defined__

#include <iostream>
using namespace std;

typedef int InfoType;

class LQueue {
    ...
};

#endif
```

Описание очереди (файл queue.h)

Часть 2 (защищённые поля и методы)

```
private:  
    struct QItem {  
        InfoType info;  
        QItem* next;  
        QItem(InfoType Ainfo): info(Ainfo), next(NULL) {}  
    };  
    QItem *front, *rear;  
    unsigned size;  
  
    void Erase();  
    void Clone(const LQueue &);
```


Описание очереди (файл queue.h)

Часть 3 (публичные методы)

public:

```
LQueue(): front(NULL), rear(NULL), size(0) {};
```

```
LQueue(const LQueue&);
```

```
~LQueue();
```

```
LQueue& operator = (const LQueue&);
```

```
void Push(InfoType AInfo);
```

```
bool Pop();
```

```
InfoType GetFirst() const;
```

```
bool IsEmpty() const;
```

```
unsigned GetSize() const;
```

```
InfoType operator [] (unsigned) const;
```

```
void Browse(void ItemWork(InfoType)) const;
```

```
void Browse(void ItemWork(InfoType&));
```

Реализация очереди (файл queue.cpp)

Часть 1 (защищённые методы)

```
void LQueue::Erase() {  
    while (Pop());  
    size = 0;  
}  
  
void LQueue::Clone(const LQueue& Q) {  
    //for (unsigned i=0; i<Q.size; i++)  
    //    Push(Q[i]);  
    QItem *tmp = Q.front;  
    for (unsigned i=0; i<Q.size; i++) {  
        Push(tmp->info);  
        tmp = tmp->next;  
    }  
}
```


Реализация очереди (файл queue.cpp)

Часть 2 (конструктор копирования, деструктор, оператор присваивания)

```
LQueue::LQueue(const LQueue& Q) {  
    size = 0; Clone(Q);  
}  
  
LQueue::~~LQueue() {  
    Erase();  
}  
  
LQueue& LQueue::operator = (const LQueue& Q) {  
    if (this != &Q) {  
        Erase();  
        Clone(Q);  
    }  
    return *this;  
}
```

Реализация очереди (файл queue.cpp)

Часть 3 (метод Push)

```
void LQueue::Push(InfoType Ainfo) {  
    QItem* tmp = new QItem(Ainfo);  
    if (size>0)  
        rear->next = tmp;  
    else  
        front = tmp;  
    rear = tmp;  
    size++;  
}
```

Реализация очереди (файл queue.cpp)

Часть 4 (метод Pop)

```
bool LQueue::Pop() {  
    if (size==0)  
        return false;  
    QItem *tmp = front;  
    front = front->next;  
    delete tmp;  
    size--;  
    if (size==0)  
        rear = NULL;  
    return true;  
}
```

Реализация очереди (файл queue.cpp)

Часть 4 (методы GetFirst и IsEmpty)

```
InfoType LQueue::GetFirst() const {  
    if (size==0)  
        throw exception("Impossible to execute  
                        GetFirst: queue is empty");  
    return front->info;  
}  
  
bool LQueue::IsEmpty() const {  
    return (size==0);  
}
```

Реализация очереди (файл queue.cpp)

Часть 4 (методы GetSize и operator [])

```
unsigned LQueue::GetSize() const {  
    return size;  
}  
  
InfoType LQueue::operator [] (unsigned k) const {  
    if ((k<0) || (k>=size))  
        throw exception("Impossible to execute  
operator[]: invalid index");  
    QItem *tmp = front;  
    for (unsigned i=0; i<k; i++)  
        tmp = tmp->next;  
    return tmp->info;  
}
```

Реализация очереди (файл queue.cpp)

Часть 4 (другой вариант перегрузки [])

```
const InfoType& LQueue::GetByIndex (unsigned k) const
{
    if ((k<0) || (k>=size))
        throw exception("Impossible to execute
                        operator[]: invalid index");
    QItem *tmp = front;
    for (unsigned i=0; i<k; i++)
        tmp = tmp->next;
    return tmp->info;
}

InfoType& LQueue::operator [] (unsigned k)
{
    return (InfoType&) GetByIndex(k);
}
```


Реализация очереди (файл queue.cpp)

Часть 5 (перегруженный метод Browse)

```
void LQueue::Browse(void ItemWork(InfoType)) const {  
    QItem *tmp = front;  
    for (unsigned i=0; i<size; i++) {  
        ItemWork(tmp->info);  
        tmp = tmp->next;  
    }  
}
```

```
void LQueue::Browse(void ItemWork(InfoType&)) {  
    QItem *tmp = front;  
    for (unsigned i=0; i<size; i++) {  
        ItemWork(tmp->info);  
        tmp = tmp->next;  
    }  
}
```

КСР: Реализовать класс НЕСТАНДАРТНЫЙ СТЕК

- **вставка** нового элемента в стек;
- **просмотр** элемента, находящегося на вершине стека;
- **удаление** из стека элемента;
- **определение** количества элементов в стеке (или определение того, пуст ли стек)

нестандартные для стека операции :

- **доступ к элементу стека по его номеру** (элемент, находящийся на вершине, имеет номер 0);
- **выполнение однотипных действий** над всеми элементами стека.