Задача 1

На практике часто возникают объекты, создание которых занимает значительное время, например, потоки (threads), TCP-соединения или подключения к базе данных. Поэтому, когда такой объект становится не нужен, его не разрушают, а откладывают в сторону, чтобы потом переиспользовать. В этой задаче вам предстоит реализовать такую схему — вам надо написать шаблон класса ObjectPool:

```
1  template <class T>
2  class ObjectPool {
3  public:
4  T* Allocate();
5  T* TryAllocate();
6
7  void Deallocate(T* object);
8
9  ~ObjectPool();
10
11 private:
12 ...
13 };
```

- Объект класса ObjectPool должен поддерживать два набора объектов: выделенные и освобождённые, — изначально оба набора пусты.
- Метод Allocate должен работать так:
  1. если есть хотя бы один освобождённый объект, то его надо перенести в множество выделенных и вернуть указатель на него в качестве результата функции
  2. если же освобождённых объектов нет, то создаётся новый объект, помещается в множество выделенных, и указатель на него возвращается в качестве результата функции
- Метод TryAllocate работает аналогично, однако если освобождённых объектов нет, он должен просто возвращать nullptr.
- Метод Deallocate переносит объект из множества выделенных в множество освобождённых; если переданный объект не содержится в множестве выделенных, метод Deallocate должен бросать исключение invalid_argument.
- Методы Allocate и TryAllocate должны возвращать объекты в порядке FIFO, т.е. множество освобождённых объектов должно представлять собой очередь: методы [Try]Allocate должны извлекать объекты из её начала, а метод Deallocate должен помещать освобождаемый объект в её конец.
- Деструктор объекта ObjectPool должен уничтожать все объекты пула, как выделенные, так и освобождённые.

Заготовка решения:

Файл: object_pool.cpp