Задание 9

Темы: классы, контейнеры, итераторы, стандартная библиотека шаблонов (STL)

STL (Standard Template Library) – Стандартная библиотека шаблонов – это набор согласованных обобщенных алгоритмов, контейнеров и средств доступа к их содержимому, а также вспомогательных функций.

Обобщенность алгоритмов и контейнеров достигается за счет использования шаблонов.

Важной частью STL являются обобщенные контейнеры.

Контейнер (контейнерный класс) – класс, предназначенный для хранения набора элементов произвольного типа.

В общем случае, доступ к элементам контейнера осуществляется посредством итераторов.

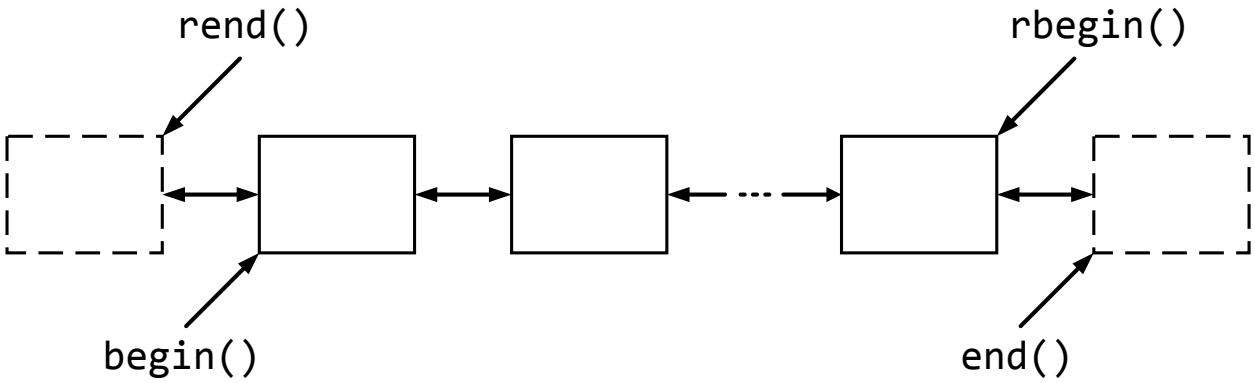
Итератор – средство доступа к элементам контейнера и навигации по контейнеру.

Итератор играет роль индекса в контейнере для операций (вставки, удаления элемента и т.д.), требующих указания положения в контейнере.

Итераторы бывают прямые и обратные. Прямые итераторы предназначены для прохода контейнера от начала к концу. обратные – для прохода от конца к началу.

При поддержке прямых итераторов, в контейнере присутствуют методы **begin**() и **end**(). Метод **begin**() возвращает итератор на первый элемент контейнера, метод **end**() – на мнимый элемент, находящийся за последним элементом контейнера.

При поддержке обратных итераторов, в контейнере присутствуют методы **rbegin**() и **rend**(). Метод **rbegin**() возвращает итератор на последний элемент контейнера, метод **rend**() – на мнимый элемент, находящийся перед первым элементом контейнера.



Для объявления прямого итератора необходимо сначала указать класс контейнера, которому принадлежит итератор, а затем тип iterator:

```
std::vector<int>::iterator it = ...;
```

При объявлении обратного итератора - класс контейнера, которому принадлежит итератор и тип reverse_iterator:

```
std::vector<int>::reverse_iterator rit = ...;
```

При написании обобщенного кода, который может работать с итераторами различных типов контейнеров:

```
typename T::iterator it = ...;
```

Итератор является обобщением указателя на элемент для контейнерных классов.

Операции с итератором напоминают операции с указателем. Для перехода к следующему элементу контейнера итератор необходимо инкрементировать, к предыдущему – декрементировать (при условии поддержки этих операций итератором).

Для получения или изменения значения элемента, на который указывает итератор, его необходимо разыменовать.

Также посредством шаблонов можно реализовать код, способный работать с различными типами контейнеров. Например, реализуем следующую шаблонную функцию:

```
template <typename T>
void DoSomething(T& Container)
{
    for (int i = 0; i < Container.size(); i++)
    {
        Container[i] += Container[0];
    }
}
```

Функция **DoSomething** способна обрабатывать контейнеры (классы), в которых определены метод **size** и оператор **[]**. Кроме того, для элементов контейнера должна быть определена операция **+=**.

При использовании в шаблонном коде для обобщенного контейнера зависимого типа (например, итератора) перед его именем необходимо использовать ключевое слово **typename**. Например, функция, выводящая на экран элементы контейнера с использованием итератора:

```
template <typename T>
void Print(T& Container)
{
    for (typename T::iterator it = Container.begin();
         it != Container.end(); it++)
    {
        std::cout << *it << " ";
    }
}
```

Данная функция способна работать с контейнерами, для которых определен прямой итератор, а элементы контейнера могут быть записаны в поток вывода посредством оператора **<<**.

Задание

Пункт 1

Создать экземпляр класса std::array<int>, std::vector<int> и std::deque<int> и заполнить их целыми числами с использованием списков инициализации.

Реализовать шаблонные функции **PrintIdx**, **PrintAt** и **PrintIterator**, которые выводят на экран элементы контейнера с использованием оператора **[]**, метода **at** и итераторов соответственно. Функции должны иметь только один шаблонный аргумент типа T (контейнер, элементы которого необходимо вывести на экран) и принимать его **по ссылке**. Функции **PrintIdx**, **PrintAt** и **PrintIterator** должны быть реализованы в обобщенном виде, то есть работать с любым из контейнеров std::array<int>, std::vector<int> и std::deque<int>.

Отсортировать элементы в контейнере с использованием алгоритма std::**sort** в порядке возрастания и убывания. После каждой сортировки необходимо выводить элементы контейнера на экран.

Элементы контейнера выводятся на экран после его инициализации и после сортировок различными функциями. Например:

```
std::vector<int> v = { ... };
PrintIdx(v);
std::sort(...);
PrintAt(v);
std::sort(...);
PrintIterator(v);
```

Пункт 2

Создать экземпляр класса std::list<int> и инициализировать его с использованием списка инициализации. Для вывода элементов списка на экран использовать одну из реализованных ранее функций **PrintIdx**, **PrintAt** и **PrintIterator**. **Модифицировать или объявлять другие функции для вывода элементов списка нельзя!** Отсортировать элементы списка в порядке возрастания и убывания

с использованием метода **sort**. Элементы контейнера выводятся на экран после его инициализации и после сортировок.

Пункт 3

Создать экземпляры классов std::queue<int> и std::stack<int> и добавить туда некоторые значения. Вывести все элементы, содержащиеся в контейнере на экран.

Пункт 4

Создать экземпляр класса std::map<std::string, std::string> и добавить туда некоторые значения.

Реализовать шаблонную функцию **Print**, осуществляющую вывод на экран всех элементов, содержащихся в контейнере. Функция принимает единственный аргумент – ссылку на контейнер std::map<T1, T2>.

Осуществить вставку элемента со значением ключа, которого еще нет в контейнере, с использованием оператора **[]** и метода **insert**.

Попытаться с использованием метода **insert** осуществить вставку элемента с ключом, который уже присутствует в контейнере.

Обратиться к элементу с несуществующем в контейнере ключом с использованием оператора **[]**. Например:

```
std::string res = myMap[“No such key”];
```

После каждого действия выводить на экран элементы контейнера. Объяснить наблюдаемые эффекты.

Объявление используемых в работе шаблонных функций должно содержаться в заголовочном файле Fun.h, реализация – **в заголовочном файле** Fun.hpp. К файлу Fun.hpp должен быть подключен заголовочный файл Fun.h:

```
//Fun.hpp
#include “Fun.h”
```

К файлу исходного кода, содержащему функцию **main**, необходимо подключить файл Fun.hpp.

Вопросы для самопроверки:

1. Что такое контейнер (контейнерный класс)?
2. Перечислите известные вам последовательные контейнеры.
3. В чем отличие статического (встроенного) массива от контейнера std::array?
4. В чем отличие контейнера std::array от std::vector
5. В чем отличие контейнера std::array от std::deque
6. В чем отличие контейнера std::vector от std::deque
7. Какие последовательные контейнеры оптимизированы для вставки элементов в конец контейнера?
8. Какие последовательные контейнеры оптимизированы для вставки элементов в начало контейнера?
9. Можно ли осуществить вставку элемента в начало контейнера? Если да, то каким образом?
10. Что такое адаптеры?
11. Что такое итератор?
12. Какие бывают итераторы?
13. Какие последовательные контейнеры имеют итератор с произвольным доступом?
14. Какие последовательные контейнеры не имеют итератора с произвольным доступом?
15. Какие последовательные контейнеры не имеют обратного итератора?
16. Что такое ассоциативные контейнеры?
17. Какие ассоциативные контейнеры вы знаете?
18. Чем std::map отличается от std::multimap
19. Какие методы, существующие для std::map, не определены для std::multimap?
20. Чем std::map отличается от std::set?
21. Какие методы, существующие для std::map, не определены для std::set?
22. Чем std::set отличается от std::multiset?
23. Почему алгоритм std::**sort** не может быть применен для контейнеров std::list и std::forward_list?
24. Какая структура данных лежит в основе контейнеров std::map и std::set?