5.6. Реализуйте класс std::insert_iterator и функцию std::inserter. Объясните, как работает ваша реализация. Приведите пример использования этого класса и этой функции.

Представим, что нам нужно скопировать один вектор в конец другого. Для этого, как известно, существует удобная функция `std::copy(first, last, other_container_first)`, объявленная в <algorithm>. Мы могли бы написать что-то вроде:

```
std::vector<int> v1 = {1, 2, 3};
std::vector<int> v2 = {0};
std::copy(v1.begin(), v1.end(), v2.end());
```

Однако это не даст желаемого результата, т.к. под v2.end и далее лежит (точнее может лежать, если вектор не аллоцировал больше памяти) чужая память и запись в неё есть UB.

Решить проблему помогает класс, конструщийся от контейнера (и итератора — начальной позиции) и являющийся “обёрткой” над итератором — `std::insert_iterator`.

```
template<class Container>
class insert_iterator;
```

Он перехватывает инкремент и разыменовывание (они, кстати говоря, ничего не делают), а вот присваивание вызывает `insert` у контейнера, от которого `insert_iterator` сконструировался, и инкрементирует обёрнутый итератор. Таким образом, следующий код сделает то что нужно:

```
std::copy(v1.begin(), v1.end(), std::insert_iterator<std::vector<int>>(v2,
    v2.end()));
//v2 = 0, 1, 2, 3;
```

Можно использовать более удобную функцию `std::inserter` (принимает контейнер и указатель на место вставки), которая за нас выводит шаблонный параметр и возвращает `insert_iterator`. Т.е. код можно переписать:

```
std::copy(v1.begin(), v1.end(), std::inserter(v2, v2.end()));
```

Реализации std::insert_iterator и std::inserter

```
#include <iterator>

template <typename Container>
class insert_iterator {
 public:
  // необходимые для работы с std::iterator_traits typedef-ы (помеченные как void считаются
    невалидными)
  // (подробности см. в реализации reverse_iterator)
  typedef void value_type;
  typedef void reference;
  typedef void difference_type;
  typedef void pointer;
  typedef std::output_iterator_tag iterator_category;

  explicit insert_iterator(Container& container, typename Container::
    iterator it) : container(&container), iter(it) {} //
    (1)
  // инкремент ничего не делает
  insert_iterator& operator++()    noexcept { return *this; };
  insert_iterator& operator++(int) noexcept { return *this; };
```

```
  // ничего не делает, но возвращает insert_iterator чтобы можно было писать *it = value
  insert_iterator& operator*()      noexcept { return *this; };

  // не операторы копирования/перемещения, а операторы присваивания элементам контейнера!
  // у контейнера должен быть определён typedef: value_type - тип элементов контейнера
  insert_iterator& operator=(const typename Container::value_type& value) {
    // iter необходимо обновлять результатом операции, т.к. он может поменяться после
   реаллокации
    iter = container->insert(iter, value);
    ++iter;
    return *this;
  }
  // вторая версия для вставки rvalue элементов
  insert_iterator& operator=(typename Container::value_type&& value) {
    iter = container->insert(iter, std::move(value));
    ++iter;
    return *this;
  }

  // наследники могут хотеть иметь доступ к контейнеру и итератору, поэтому не private
 protected:
  // проще хранить указатель, чем возиться с копиями и ссылками
  Container* container;
  typename Container::iterator iter;
};

template <typename Container>
insert_iterator<Container> inserter(Container& container, typename Container
   ::iterator it) {
  return insert_iterator<Container>(container, it);
}

/* Примечания:
(1) Итераторы принято передавать по значению, тк.. они достаточно малы, и ссылки не
   дают преимущества перед передачейпо
 значению. */
```

5.7. Реализуйте классы std::istream_iterator и std::ostream_iterator. Объясните, как работает ваша реализация. Приведите пример использования этих классов.

Задача: Считать из файла input.txt массив целых чисел, разделенных пробельными символами. Отсортировать их и записать в файл output.txt

Решение:

```
#include <vector>
#include <algorithm>
#include <fstream>

int main(){
  // открываем input.txt для чтения
  std::ifstream fin("input.txt");
  // открываем output.txt для записи
  std::ofstream fout("output.txt");
  // объявление и инициализация пустого целочисленного вектора
  std::vector<int> v;

```

```
  // сложная магия, благодаря которой из потока чтения вставляются элементы в конец вектора
  std::copy(std::istream_iterator<int>(fin), std::istream_iterator<int>(),
   std::inserter(v, v.end()));
  // алгоритм сортировки
  std::sort(v.begin(), v.end());
  // сложная магия, благодаря которой элементы из вектора копируются в поток записи
  std::copy(v.begin(), v.end(), std::ostream_iterator<int>(fout, " "));
  return 0;
}
```

- Одной из основ библиотеки являются итераторы, а также полуинтервалы, ими определяемые. По семантике (читай — по поведению) они совпадают с указателями. То есть, опреатор разыменования * вернет вам элемент, на который ссылается итератор, ++ переведет итератор на следующий элемент. В частности, любой контейнер представляется его концевыми итераторами [begin, end), где begin указывает на первый элемент, end — за последний;

- Алгоритмы, работающие с контейнерами, в качестве параметров принимают начало и конец контейнера (или его части);

- Алгоритм копирования copy просто переписывает элементы из одного полуинтервала в другой. Если в целевом контейнере не выделена память, то поведение непредсказуемо [copy];

- Функция `inserter` вставляет значение в контейнер перед итератором [`inserter`]

- `istream_iterator` и `ostream_iterator` предоставляют доступ к потокам в стиле контейнеров [`istream_iterator, ostream_iterator`]

Ещё один пример использования:

```
#include <iostream>
#include <sstream>
#include <iterator>
#include <numeric>
#include <algorithm>

int main()
{
    std::istringstream str("0.1 0.2 0.3 0.4");
    std::partial_sum(std::istream_iterator<double>(str),
                     std::istream_iterator<double>(),
                     std::ostream_iterator<double>(std::cout, " "));

    std::istringstream str2("1 3 5 7 8 9 10");
    std::cout << "\nThe first even number is " <<
        *std::find_if(std::istream_iterator<int>(str2),
                      std::istream_iterator<int>(),
                      [](int i){return i%2 == 0;})
        << ".\n";
    //"9 10"left in the stream
    // Вывод:
    // 0.1 0.3 0.6 1
    // The first even number is 8.
}
```

Реализация:

```
    template <typename T>
    class istream_iterator {
        std::istream& in;
        T value;
    public:
        istream_iterator(std::istream& in): in(in) {
```

```
            in >> value;
        }
        istream_iterator<T>& operator++() {
            in >> value;
        }
        T& operator*(){
            return value;
        }
    };
    // children:
    // std::ifstream in("input.txt");
    // std::istringstream iss(s);
```