Очень краткое описание контейнеров STL Vector и List (4 семестр)

w!!Контейнер вектор!!

- *vector* – имя контейнера,
- *T* – тип элементов контейнера (*value_type*),
- *A* – распределитель памяти (*allocator_type*) – необязательный параметр.

```
template < class T , class A = allocator < T > > class vector {
    // ...
public:
```

Доступ к элементам

```
    reference       operator [] (size_type n);  // доступ без проверки диапазона
    const_reference operator [] (size_type n) const;
    reference       at          (size_type n);  // доступ с проверкой диапазона (если индекс
                                                // выходит за пределы диапазона, возбуждается исключение out_of_range)
    const_reference at          (size_type n) const;
    reference       front       ();             // первый элемент вектора
    const_reference front       () const;
    reference       back        ();             // последний элемент вектора
    const_reference back        () const;
```

Конструкторы, которые могут вызываться с одним параметром, во избежание случайного преобразования объявлены *explicit*, что означает, что конструктор может вызываться только явно (*vector<int> v* = 10 – ошибка, попытка неявного преобразования 10 в *vector<int>*).

```
    explicit        vector      (const A&=A()); // создается вектор нулевой длины
    explicit        vector      (size_type n; const T& value = T(); const A& = A());
```

Создается вектор из *n* элементов со значением *value* (или с «нулями» типа *T*, если второй параметр отсутствует; в этом случае конструктор умолчания в классе *T* обязателен).

```
    // инициализация вектора копированием элементов из [first, last), I - итератор для чтения
template <class I>  vector      (I first, I last, const A& = A());
                    vector      (const vector  < T, A > & obj );   // конструктор копирования
    vector&         operator =  (const vector  < T, A > & obj );
                   ~vector      ();
```

Некоторые функции-члены класса *vector*.

```
    iterator    erase  ( iterator i ); // удаляет элемент, на который указывает данный
                                       // итератор. Возвращает итератор элемента, следующего за удаленным.
    iterator    erase  ( iterator st, iterator fin );       // удалению подлежат все элементы
```

```
                                                        // между st и fin, но fin не удаляется. Возвращает fin.
    Iterator    insert ( iterator i , const T& value = T()); // вставка некоторого
                                                        // значения value перед i. Возвращает итератор вставленного элемента).
    void        insert (iterator i , size_type n, const T&value); // вставка n копий
                                                          // элементов со значением value перед i.
    void        push_back ( const T&value ) ; // добавляет элемент в конец вектора
    void        pop_back () ;                 // удаляет последний элемент (не возвращает значение!)
    size_type   size() const;                 // выдает количество элементов вектора
    bool        empty () const;               // возвращает истину, если вызывающий вектор пуст
    void        clear();                      // удаляет все элементы вектора
    //...
}
```

Пример

Печать элементов вектора в прямом порядке.

```
#include < vector >;
using namespace std;

int main () {
    vector < int > V ( 100,5 );
    vector < int > :: const_iterator  p = V.begin ();
    while (p != V.end ()) {
        cout << *p << ' ';
        ++p;
    }
    cout << endl;
}
```

Или в обратном порядке.

```
vector < int > :: const_reverse_iterator  q = V.rbegin ( );
while ( q != V.rend ()) {
    cout << *q << ' ';  // печать в обратном порядке
    ++q;
}
cout << endl;
```

Контейнер список

- *list* – имя контейнера,
- T – тип элементов, которые будут храниться в списке,
- A – распределитель памяти.

```
template < class T , class A = allocator < T > > class list {
    // ...
public:
```

Доступ к элементам.

```
reference          front      ();       // первый элемент списка
const_reference    front      () const;
reference          back       ();       // последний элемент списка
const_reference    back       () const;
```

Конструкторы и т.п.

```
explicit           list       (const A&=A()); //создается список нулевой длины
explicit           list       (size_type n; const T& value = T(); const A& = A());
```

Создается список из n элементов со значением value (или с «нулями» типа T, если второй параметр отсутствует.

```
// инициализация списка копированием элементов из [first, last), I --- итератор для чтения
template <class I> list       (I first, I last, const A& = A());
                   list       (const list  < T, A > & obj );     // конструктор копирования
list&              operator = (const list  < T, A > & obj );
                  ~list       ();
// ... ...
```

Некоторые функции-члены класса *list*.

```
iterator    erase  ( iterator i ); // удаляет элемент, на который указывает данный
                                   // итератор. Возвращает итератор элемента, следующего за удаленным.
iterator    erase  ( iterator st, iterator fin );         // удалению подлежат все элементы
                                                          // между st и fin, но fin не удаляется. Возвращает fin.
Iterator    insert ( iterator i, const T& value = T() );  // вставка некоторого
                                                          // значения value перед i. Возвращает итератор вставленного элемента).
void        insert ( iterator i, size_type n, const T&value ); // вставка n копий
                                                          // элементов со значением value перед i.
void        push_back ( const T&value ) ;  // добавляет элемент в конец списка
void        push_front ( const T&value ) ; // добавляет элемент в начало списка
void        pop_back () ;                  // удаляет последний элемент (не возвращает значение!)
void        pop_front () ;                 // удаляет первый элемент списка
size_type   size() const;                  // выдает количество элементов списка
bool        empty () const;                // возвращает истину, если вызывающий список пуст
void        clear();                       // удаляет все элементы списка
//...
}
```