**Построение Простого Контейнера на примере вектора**

template <class T>

class vector

{public:

typedef T\* iterator;

explicit vector(int n=100); //create a size n array

vector(const vector<T>& v); //copy vector

vector(const T a[], int n); //copy an array

~vector() { delete []p; }

iterator begin(){ return p;}

iterator end(){ return p + size;}

T& operator[](int i); //range checked element

vector<T>& operator=(const vector<T>& v);

private:

T\* p; //base pointer

int size; //number of elements

};

template <class T>

vector<T>::vector(int n = 100): size(n)

{ assert(n > 0); p = new T[size]; assert(p != 0);}

template <class T>

vector<T>::vector(const T a[], int n)

{ assert(n > 0); size = n; p = new T[size]; assert(p != 0);

for (int i = 0; i < size; ++i) p[i] = a[i];

}

template <class T>

vector<T>::vector(const vector<T>& v)

{ size = v.size; p = new T[size]; assert(p != 0);

for (int i = 0; i < size; ++i) p[i] = v.p[i];

}

template <class T>

T& vector<T>::operator[](int i)

{ assert (i >= 0 && i < size ); return (p[i]);}

template <class T>

vector<T>& vector<T>::operator=  
(const vector<T>& v)

{ assert(v.size == size);

for (int i = 0; i < size; ++i) p[i] = v.p[i];

return \*this;

}

int main()

{

vector<double> v(5);

vector<double>::iterator p ;

int i = 0;

for (p = v.begin() ; p != v.end(); ++p) \*p = 1.5 + i++;

do { --p; cout << \*p << " , ";} while (p != v.begin());

cout << endl;

}

**Упражнения.**

Расширить интерфейс класса, добавляя функции: **push\_back, pop\_back, insert, erase,** **resize,** **operator int()** (находит количество элементов).

При добавлении нового элемента и переполнении хранилища расширить его размер в два раза.

ВОПРОСЫ ПО ПОСЛЕДОВАТЕЛЬНЫМ КОНТЕЙНЕРАМ

Студент начинает с вопроса, номер которого совпадает с номером студента в списке подгруппы, и далее по списку вопросов добавляя к номеру 6, пока не ответит на шесть вопросов. Если номер вопроса превысил число 34, берём остаток от деления на 34. Например, 10+16+22+28+34+ 6.

VECTOR

1. Какие реализации конструкторов используются для векторов?
2. Как задать размер контейнера?
3. Как определить потенциальный размер контейнера?
4. Как получить доступ к элементу контейнера?
5. Как присвоить значение элементу контейнера?
6. Опишите функциональность методов begin и end
7. Опишите использование метода back и front
8. Опишите использование метода insert
9. Как и куда можно вставить новые элементы в контейнер?
10. Опишите функциональность методов capacity и max\_size
11. Зачем нужны методы reserve и resize?
12. Как вставит/удалить элемент в начало контейнера?
13. Как вставит/удалить элемент в конец контейнера

DEQUE

1. Какие виды двусторонних очередей позволяют создать конструкторы?
2. Как получить доступ к элементу контейнера?
3. Как присвоить значение элементу контейнера?
4. Опишите функциональность методов front и back
5. Как вставит/удалить элемент в начало контейнера?
6. Как вставит/удалить элемент в конец контейнера

LIST

1. Опишите использование метода merge
2. Опишите использование метода remove\_if
3. Опишите использование метода splice
4. Опишите использование метода unique

STACK

1. Опишите функциональность методов push и top и как она реализуется в зависимости от выбора базового контейнера?
2. Почему стек реализован как адаптер контейнера? Каким образом?
3. Каким образом изменить умолчание в выборе базового контейнера?

QUEUE

1. Опишите функциональность методов push и top и как она реализуется в зависимости от выбора базового контейнера?
2. Почему очередь реализован как адаптер контейнера? Каким образом?
3. Каким образом изменить умолчание в выборе базового контейнера?

ОБЩИЕ ВОПРОСЫ

1. У каких контейнеров допустим произвольный доступ к элементам?
2. Для каких контейнеров сохраняются значения указателей, итераторов после вставки/удаления?
3. Какие последовательные контейнеры поддерживают упорядоченность элементов автоматически?  
   Какие средства можно использовать для сортировки элементов контейнера?
4. Когда, в каких случаях нужно отдать предпочтение выбору одного из контейнеров?
5. Какие средства можно использовать и что необходимо реализовать для сравнения элементов контейнеров, если они содержат объекты пользовательских классов?

template <class T, class Container = deque<T> >

class stack {

public:

typedef typename Container::value\_type value\_type;

typedef typename Container::size\_type size\_type;

typedef Container container\_type;

protected:

Container c; *// container*

public:

explicit stack(const Container& = Container());

bool empty() const { return c.empty(); }

size\_type size() const { return c.size(); }

void push (const value\_type& x) { c.push\_back(x); }

void pop() { c.pop\_back(); }

value\_type& top() { return c.back(); }

const value\_type& top() const { return c.back(); }

};

template <class T, class Container>

bool operator==(const stack<T, Container>&,

const stack<T, Container>&);

template <class T, class Container>

bool operator< (const stack<T, Container>&,

const stack<T, Container>&);

...*// (other comparison operators)*

}

queue< int, list< int > > q1, q2, q3;

// А можно было и так: queue< int, deque< int > > q1, q2, q3;