#include <iostream>

#include <stdexcept>

using namespace std;

//klasa szablonu tablicą zmienną wielkości

template <class T>

class Array

{

public:

//synonimy typów

typedef T type\_value;

typedef T\* iterator;

typedef const T\* const\_iterator;

typedef T& reference;

typedef const T& const\_reference;

typedef size\_t size\_type;

//konstruktor

Array(const size\_type &n = minsize)

{

Count = 0;

if (n < minsize)

{

elems = new type\_value[minsize]; Size = minsize;

}

else { elems = new type\_value[n]; Size = n; };

for (unsigned i = 0; i<Size; i++)

elems[i] = 0;

}

//konstruktor kopiuący

Array(const Array &ref)

{

Count = ref.Count;

Size = ref.Size;

elems = new type\_value[ref.Size];

for (unsigned i = 0; i<Count; i++)

elems[i] = ref.elems[i];

}

//konstruktor przez iteratory

Array(iterator first, iterator last)

{

if (first<last)

{

size\_type n = last - first;

Count = n;

if (n < minsize)

{

elems = new type\_value[minsize]; Size = minsize;

}

else elems = new type\_value[n];

for (unsigned i = 0; i<Count; i++)

elems[i] = \*(first + i);

}

else throw invalid\_argument("Invalid argument!");

}

//operator przepicania

Array& operator = (const Array &ref)

{

if (this != &ref)

{

Count = ref.Count;

Size = ref.Size;

type\_value \*new\_elems = new type\_value[ref.Size];

for (unsigned i = 0; i<Count; i++)

new\_elems[i] = ref.elems[i];

delete[] elems;

elems = new\_elems;

}

return \*this;

}

//dostęp do elementów

reference front() { return elems[0]; } //zmienić 1 el

const\_reference front() const { return elems[0]; }//otrzymać 1 el.

reference back() { return elems[size() - 1]; }//zmienić ostatni el.

const\_reference back() const { return elems[size() - 1]; }//otrzymać ostatni el.

reference operator[] (size\_type index)

{

if (index<Count) return elems[index];

else throw out\_of\_range("Index out of range!");

}

const\_reference operator[] (size\_type index)const

{

if (index<Count) return elems[index];

else throw out\_of\_range("Index out of range!");

}

//rozmiary

bool empty() const { if (Count == 0) return true; else return false; }; //sprawdź pustkę

size\_type size() const { return Count; }; //obecny rozmiar

size\_type capacity() const { return Size; };

//zmiana rozmiaru

void resize(size\_type new\_size)

{

if (new\_size > capacity())

{

type\_value \*data = new type\_value[new\_size]; //nowa tablica

for (size\_type i = 0; i < Count; ++i)

data[i] = elems[i];

delete[] elems;//czyszczenie starej tablicy

elems = data;//przypisanie wskaznika

Size = new\_size;

}

}

friend ostream& operator << (ostream& s, const Array <T> &a)

{

for (size\_t i = 0; i<a.Count; i++)

s << a[i] << " ";

s << endl;

return s;

}

friend istream& operator >> (istream& s, Array <T> &a)

{

for (size\_t i = 0; i<a.Count; i++)

s >> a[i];

return s;

}

//iteratory w klasie

iterator begin() { return elems; };

const\_iterator begin()const { return elems; };

iterator end() { return elems + Count; };

const\_iterator end()const { return elems + Count; };

//wstaw na koniec

void push\_back(const type\_value &v)

{

if (Count == Size)

resize(2 \* Size);

elems[Count++] = v;

}

//usuń element z końca

void pop\_back()

{

if (!empty())elems[Count--] = 0;

}

void clear() { Count = 0; };

//wyczyść tablicę

Array& operator += (const Array &r)

{

Array t(capacity() + r.capacity());

unsigned i;

for (i = 0; i< Count; ++i) t.push\_back(this->elems[i]);

for (i = 0; i< r.Count; ++i) t.push\_back(r.elems[i]);

this->resize(capacity() + r.capacity());

\*this = t;

return \*this;

}

~Array() { delete[] elems; };

private:

static const size\_type minsize = 10; //minimalny rozmiar tablicy

size\_type Size; //liczba przydzielonych elementów pamięci

size\_type Count; //liczba przydzielonych elementów tablicy

type\_value \*elems;//wskaźnik do danych

};

int main()

{

Array <int> d(5);

cout << d.size() << " " << d.capacity() << endl;

for (int i = 0; i< 12; ++i)

d.push\_back(i + 1);

cout << d;

cout << d.size() << " " << d.capacity() << endl;

//iterator

Array <int>g(3);

g = d; //operator =

cout << g.size() << " " << g.capacity() << endl;

Array<int>::iterator b = g.begin(), e = g.end();

for (int j = 100; b != e - 3; ++b, j += 12)

\*b = j;

g += d;

cout << g;

double f[] = { -1.0, 7.8, -4.5, 7.9, 6.9, -8.3 };

int n = sizeof(f) / sizeof(double);

Array <double> f1(f, f + n), c;

cout << f1;

//f1 = g;

/\*

cout << (f1.front() = 12) << " " << f1.back() << endl;

c += f1;

cout << (c[2] = -100) << endl;

cout << c;

cout << c.size() << " " << c.capacity() << endl;\*/

system("pause");

return 0;

}