Шаблоны

```
template <список параметров шаблона>
  объявление
```

```
template
 <список параметров шаблона функции>
 тип имя (список параметров)
 {...}
```

```
template <typename TYPE> TYPE abs(TYPE x)
 { return x >= 0 ? x : -x; }

void main()
 { int    i = 567;
   double d = -123.45;

   printf("%7d\n",    abs<int>(i));
   printf("%7.2lf\n", abs<double>(d));
   printf("%7d\n",    abs(i));
   printf("%7.2lf\n", abs(d));
  }
```

```
template <typename TYPE> TYPE max(TYPE a, TYPE b)
 { return a > b ? a : b; }

void main()
 { int    i = 567;
   float  f = 7.5;
   double d = -123.45;

   printf("%7d\n",    max(i, 0));
   printf("%7.2lf\n", max(d, 0));
   printf("%7.2lf\n", max(d, 0.0));
   printf("%7.2lf\n", max(d, f));
   printf("%7.2lf\n", max(d, (double)f));
   printf("%7.2lf\n", max<double>(d, f));
  }
```

```
template <typename TYPE> TYPE max(TYPE a, TYPE b)
 { return a > b ? a : b; }

template <typename TYPE> TYPE max(TYPE a, TYPE b,
                                  TYPE c)
 { TYPE d;

   d = a;
   if (b > d) d = b;
   if (c > d) d = c;
   return d;
  }
```

```
template
 <список параметров шаблона класса>
 class имя_шаблона_класса
 { ... };


имя_шаблона_класса <список фактических
                    параметров
шаблона>
```

```
template <class TYPE> class Vector
 { private:
    int  size;
    TYPE *v;
   public:
    Vector(int n = 0);
    ~Vector();
   ...
  };

template <class TYPE, int NMAX = 100> class Vector
 { private:
    int  size;
    TYPE v[NMAX];
   public:
    Vector();
    ~Vector();
   ...
  };

template <class CLASS, void (*err_fun)()> class List
 { ... };
```

```
template <class TYPE, int NMAX> class Vector
 { private:
    int  size;
    TYPE v[NMAX];
   public:
    Vector();
    Vector(TYPE x);
  };
```

```
template <class TYPE, int NMAX>
      Vector<TYPE, NMAX>::Vector()
 { size = NMAX;
   for (int i = 0; i < size; i++)
    v[i] = 0;
  }

template <class TYPE, int NMAX>
      Vector<TYPE, NMAX>::Vector(TYPE x)
 { size = NMAX;
   for (int i = 0; i < size; i++)
    v[i] = x;
  }
```

```
template <class T> class X
{ public:
    friend void  f1();
    friend X<T>* f2();
    friend int   f3(X<T> *p);
    friend void  f4(X *p);
  };

void f1() { … }

template <class T> X<T>* f2 () { … }

template <class T> int f3(X<T> *p) { … }
```

```
template <class TYPE = int, int NMAX = 10>
class Vector
 { private:
    int  size;
    TYPE v[NMAX];
   ...
  };

Vector<> v;
```

```
template <> char abs<char>(char x)
 { return x; }
```

```
template <> class Vector<void*>
 { ... };

Vector<void*> vpv;
```

```
template <class T> class Vector<T*>
 { ... };

Vector<Shape*> spv;
// <T*> – это <Shape*>, поэтому T – это Shape

Vector<int**>  ippv;
// <T*> – это <int**>, поэтому T – это int*
```

```
template <class T> class Vector;

template <class T> class Vector<T*>;

template <> class Vector<void*>;
```

```
template <class TYPE, int NMAX> class Vector
 { private:
    int  size;
    TYPE v[NMAX];
   public:
    Vector();
    Vector(TYPE x);
    ~Vector() { }
    int  GetSize() const { return size; }
    TYPE&       operator [] (int n);
    const TYPE& operator [] (int n) const;
    int         operator == (const Vector& vector2);
    Vector      operator +  (const Vector& vector2);
    Vector      operator += (const Vector& vector2);
    Vector      operator -  (const Vector& vector2);
    Vector      operator -= (const Vector& vector2);
    TYPE        operator *  (const Vector& vector2);
  };

template <class TYPE, int NMAX> ostream& operator<< (ostream &f,
                                    const Vector<TYPE, NMAX> &v);

template <class TYPE, int NMAX> istream& operator>> (istream &f,
                                      Vector<TYPE, NMAX> &v);
```

```
#include <iostream>
using namespace std;
#include "Vector.h"

template <class TYPE, int NMAX> Vector<TYPE, NMAX>::Vector()
 { size = NMAX;
   for (int i = 0; i < size; i++)
    v[i] = 0;
  }

template <class TYPE, int NMAX> Vector<TYPE, NMAX>::Vector(TYPE x)
 { size = NMAX;
   for (int i = 0; i < size; i++)
    v[i] = x;
  }
```

```
template <class TYPE, int NMAX>
inline TYPE& Vector<TYPE, NMAX>::operator [] (int n)
 { if (n < 0) n = 0;
   else if (n >= size) n = size - 1;
   return v[n];
  }

template <class TYPE, int NMAX>
inline const TYPE& Vector<TYPE, NMAX>::operator [] (int n) const
 { if (n < 0) n = 0;
   else if (n >= size) n = size - 1;
   return v[n];
  }

template <class TYPE, int NMAX>
int Vector<TYPE, NMAX>::operator == (const Vector& vector2)
 { for (int i = 0; i < size; i++)
    if (v[i] != vector2.v[i])
     return 0;
   return 1;
  }
```

```
template <class TYPE, int NMAX> Vector<TYPE, NMAX>
Vector<TYPE, NMAX>::operator +  (const Vector& vector2)
 { Vector res;

   for (int i = 0; i < size; i++)
    res.v[i] = v[i] + vector2.v[i];
   return res;
  }

template <class TYPE, int NMAX> Vector<TYPE, NMAX>
Vector<TYPE, NMAX>::operator += (const Vector& vector2)
 { for (int i = 0; i < size; i++)
    v[i] += vector2.v[i];
   return *this;
  }
```

```
template <class TYPE, int NMAX> Vector<TYPE, NMAX>
Vector<TYPE, NMAX>::operator -  (const Vector& vector2)
 { Vector res;

   for (int i = 0; i < size; i++)
    res.v[i] = v[i] - vector2.v[i];
   return res;
  }

template <class TYPE, int NMAX> Vector<TYPE, NMAX>
Vector<TYPE, NMAX>::operator -= (const Vector& vector2)
 { for (int i = 0; i < size; i++)
    v[i] -= vector2.v[i];
   return *this;
  }

template <class TYPE, int NMAX>
TYPE Vector<TYPE, NMAX>::operator *  (const Vector& vector2)
 { TYPE res = 0;

   for (int i = 0; i < size; i++)
    res += v[i] * vector2.v[i];
   return res;
  }
```

```
template <class TYPE, int NMAX>
ostream& operator<< (ostream &f, const Vector<TYPE, NMAX> &v)
 { streamsize s = f.width();

   for (int i = 0; i < v.GetSize(); i++)
    f << setw(0) << " " << setw(s) << v[i];
   f << endl;
   return f;
  }

template <class TYPE, int NMAX>
istream& operator>> (istream &f, Vector<TYPE, NMAX> &v)
 { for (int i = 0; i < v.GetSize(); i++)
    f >> v[i];
   return f;
  }
```

```
class Complex
 { private:
    double r, m;
   public:
    Complex(double nr = 0, double nm = 0) : r(nr), m(nm) {};
    Complex operator ++();
    Complex operator ++(int);
    Complex operator --();
    Complex operator --(int);
    Complex operator +(const Complex& c) const;
    Complex operator -(const Complex& c) const;
    Complex operator +=(const Complex& c);
    Complex operator -=(const Complex& c);
    bool    operator ==(const Complex& c) const;
    bool    operator !=(const Complex& c) const;
    friend ostream& operator<< (ostream &f, const Complex &c);
    friend istream& operator>> (istream &f, Complex &c);
  };
```

```
#include <iostream>
#include <iomanip>
#include "Vector.cpp"
#include "Complex.h"
using namespace std;

void main()
 {
   // Вещественный вектор
   Vector<double, 10> v1, v2, v3;
   const Vector<double, 10> v5;

   for (int i = 0; i < v1.GetSize(); i++)
    { v1[i] = i; v2[i] = i + 50; }
   cout << fixed << setprecision(1);
   cout << "v1:      " << setw(5) << v1;
   cout << "v2:      " << setw(5) << v2;
   v3 = v1 + v2;
   cout << "v1 + v2: " << setw(5) << v3;
   v3 = v1 - v2;
   cout << "v1 - v2: " << setw(5) << v3;
   double r = v1 * v2;
   cout << "r = " << r << endl;
   cout << "v5:      " << setw(5) << v5;
```

```
// Целочисленная матрица
  Vector<Vector<int, 3>, 3> m1, m2(7), m3;

  cout << "m1" << endl << setw(2) << m1;
  cout << "m2" << endl << setw(2) << m2;
  for (int i = 0; i < m1.GetSize(); i++)
   for (int j = 0; j < m1[i].GetSize(); j++)
    { m1[i][j] = (i + 1) * (j + 1); m2[i][j] = (i + 1) * j + 5; }
  cout << "m1" << endl << setw(2) << m1;
  cout << "m2" << endl << setw(2) << m2;
  m3 = m1 + m2;
  cout << "m3" << endl << setw(2) << m3;
  m3 = m1 - m2;
  cout << "m3" << endl << m3;
  Vector<int, 3> v = m1 * m2;
  cout << "v = " << v << endl;
```

```
// Вектор комплексных чисел
  Vector<Complex, 3> c1, c2, c3;

  for (int i = 0; i < c1.GetSize(); i++)
   c1[i] = Complex(i, -i);
  for (int i = 0; i < c2.GetSize(); i++)
   c2[i] = Complex(i * 2, -i * 3);
  cout << "c1 = " << setw(4) << c1;
  cout << "c2 = " << setw(4) << c2;
  c3 = c1 + c2;
  cout << "c3 = " << setw(4) << c3;
  c3 = c1 - c2;
  cout << "c3 = " << setw(4) << c3;
  Complex x = c1 * c2;
 }
```