# Шаблоны

```
max<int>()
```

- Шаблоны – механизм введения *обобщенного* кода
- Шаблоны позволяют на основе *заготовки* создавать разные сущности (функции или классы), различающиеся определенными параметрами (типами аргументов, полей и т.п.)
- В C++ можно создавать шаблоны для функций и для классов
- Шаблоны в C++ не являются законченным программным кодом, пригодным для компиляции. Шаблоны – это заготовка для создания кода.
- Шаблонные функции и классы должны компилироваться совместно с кодом обращения к шаблонным функциям и классам. Поэтому шаблоны распространяются в виде исходного кода (как правило, в заголовочных файлах)

На развитие в C++ средств обобщенного программирования огромное влияние оказал Александр Степанов, один из создателей библиотеки STL

# Макросы в Си

- *Макросы* в C/C++ являются средством препроцессорной обработки
- Позволяют вводить элементы, которые имитируют функции, но по сути являются подстановочными конструкциями
- Вместо «вызова» макроса подставляется заданное выражение; вместо «аргументов» подставляются значения, определенные в «вызове»
- Макросы не требуют указания типов, поэтому являются средством обобщенного кода

```
#define f(a, b) a + b
#define max(a, b) a > b ? a : b

int main()
{
    int v = f(2, 3);
    double m = max(17.34, 11.987);
    cout << m << endl;
    return 0;
}
```

# Опасность макросов

- Из-за «подстановочной» природы макросы могут приводить к другим результатам по сравнению с функциями

```
#define max(a, b) a > b ? a : b
#define f(a, b) a * b
int main()
{
    cout << f(2 + 2, 2) << endl;
    cout << max(3.14, 5.17) << endl;
}
```

# Шаблоны функций

- Определение шаблонной функции предваряется ключевым словом **template** и параметрами шаблона
- Ключевое слово **class** или **typename** указывают, что параметром шаблона является тип элемента, который обозначается как T.
- Компилятор сформирует программный код функции или нескольких функций на основе заданного шаблона, если встретит вызов шаблонной функции с информацией о параметрах шаблона (*инстанциация шаблона*)

```
template<class T>
T findMax(T values[], int size)
{
    T res = values[0];
    for(int i=1; i<size; i++)
        if(res < values[i])
            res = values[i];
    return res;
}
```

```
double d = findMax<double>(ar, 5);
int res = findMax<int>(m, 5);
```

# Шаблоны функций

- *Инстанциация* для одинаковых параметров шаблона выполняется один раз (порождается одна функция)

- В некоторых случаях вызов шаблонной функции может осуществляться без явного указания параметров шаблона (компилятор автоматически выводит параметры шаблона)

```
findMax<int>(m, 5);

findMax<int>(m, 5);

int res = findMax(m, 5);
```

# Особенности шаблонов в C++

- Шаблонная функция не компилируется до *инстанцирования*
- Не любое значение параметра шаблона приводит к успешной компиляции
- В теле шаблонной функции допускаются любые манипуляции с параметрами

```
template<typename T>
T Sum(T a, T b)
{
    return a + b;
}
```

```
template<class T>
int f(T v)
{
    return v.getValue();
}
```

# Явная специализация шаблона

- Общий шаблон может не подходить для всех значений параметров (для всех типов)
- В C++ можно определить частный случай шаблонной функции (или шаблонного класса) для определенных значений параметров шаблона (сформировать исключение из общего правила шаблона)

```
template<>
const char findMax(const char* values,
                                  int size)
{
    char res = toupper(*values);
    char curr;
    for(int i=1; i<size; i++)
    {
        curr = toupper(values[i]);
        if(res < curr)
            res = curr;
    }
    return res;
}
```

```
int res = findMax(m, 5);
char z = findMax("Hello world", 11);
```

# Шаблон класса

- Если методы класса работают с элементами этого же класса, то необходимо указывать имя класса с параметром шаблона

```
template<class T>
class Matrix {
    T* data;
    int size;
public:
    Matrix(): data(nullptr), size(0);
    Matrix(T* items, int size);
    T operator[](int idx);
    Matrix<T> operator+(Matrix<T> other);
    ~Matrix();
    int size() { return size; }
};
```

- При определении методов класса необходимо указывать как шаблонную функцию

```
template<class T> Matrix<T>::~Matrix()
{
    delete[] data;
}
```

# Частичная специализация для классов

- Шаблонные классы поддерживают явную специализацию с указанием значений всех параметров шаблона
- Также поддерживается частичная специализация с указанием конкретных значений некоторых параметров шаблона
- Можно вводить разные шаблоны для случаев работы с указателем, ссылкой и собственно экземпляром типа

```
template<class A, class B>
class Tuple
{
    A first;
    B second;
   /* .. */
};
```

```
template<class A>
class Tuple<class A, int>
{
    /* .. */
};
```

```
template<class A, class B>
class Tuple<A*, B*> {
    /* .. */
};
```

# Параметры шаблона в виде констант

- В качестве параметров шаблона могут выступать константные выражения

```
template<class T, int N>
class Stack
{
    T st[N];
    short top;
public:
    Stack(): top(-1) {}
    bool push(T item);
    T& pop();
};
```

```
Stack<int, 20> stack;
stack.push(1);
stack.push(2);
stack.push(3);
```

# STL = standard template library

- Стандартная библиотека шаблонов была разработана А. Степановым и Менг Ли, сотрудниками Hewlett-Packard, и с 1998 г. входит в стандарт языка C++.
- STL – стандартная библиотека для хранения и обработки данных.
- Основные компоненты STL:
  - *Контейнеры* – структура для хранения данных; построены в виде шаблонных классов и могут содержать данные любого типа.
    - *Последовательные* – векторы (vector), списки (list), очереди с двусторонним доступом (deque); Производные типы: stack, queue, priority queue.
    - *Ассоциативные* – множества (set), мультимножества (multiset), отображения (map) и мультиотображения (multimap);
  - *Алгоритмы* – процедуры обработки данных, унифицированные для работы с разными структурами данных, включая обычные массивы
  - *Итераторы* – интеллектуальные указатели, которые обеспечивают доступ к элементам контейнера, скрывая особенности строения