



C++

Разбор домашней работы



Шаблоны

Теория



15 минут



Что общего между этими функциями?

```
int sum(int a, int b) {  
    return a + b;  
}
```

```
double sum(double a, double b) {  
    return a + b;  
}
```

```
unsigned int sum(unsigned int a, unsigned int b) {  
    return a + b;  
}
```

Что общего между этими функциями?

```
int sum(int a, int b) {  
    return a + b;  
}
```

Все три выполняют одну задачу - сложение двух переменных

```
double sum(double a, double b) {  
    return a + b;  
}
```

```
unsigned int sum(unsigned int a, unsigned int b) {  
    return a + b;  
}
```

Какие различия между этими функциями?

```
int sum(int a, int b) {  
    return a + b;  
}
```

Все три выполняют одну задачу - сложение двух переменных

```
double sum(double a, double b) {  
    return a + b;  
}
```

```
unsigned int sum(unsigned int a, unsigned int b) {  
    return a + b;  
}
```

Какие различия между этими функциями?

```
int sum(int a, int b) {  
    return a + b;  
}
```

Все три выполняют одну задачу - сложение двух переменных

```
double sum(double a, double b) {  
    return a + b;  
}
```

Они работают с разными типами данных

```
unsigned int sum(unsigned int a, unsigned int b) {  
    return a + b;  
}
```


Можно ли вместо трёх функций написать одну, чтобы она работала сразу со всеми типами данных?

Обобщенные функции. Шаблоны

Шаблоны - средство языка C++, предназначенное для кодирования обобщённых алгоритмов, без привязки к некоторым параметрам.

Основная идея - создание функций без указания точного типа некоторых или всех переменных.

Обобщенные аргументы

`int`
`unsigned`
`Fraction`
`char`
`double`
`float`
`string`

`int`



`<typename` `T>`
`<class` `T>`

Обобщенные аргументы могут преобразовываться в нужный тип при компиляции.

typename, class - обозначение обобщенного аргумента

Обобщенная функция

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

Обобщенная функция

```
template <typename T>
```

```
T sum(T a, T b) {
```

```
    return a + b;
```

```
}
```

`template` - ключевое слово, обозначающее, что дальше будет шаблонная функция

Обобщенная функция

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

`template` - ключевое слово, обозначающее, что дальше будет шаблонная функция

`<typename T>` - обозначение обобщенного аргумента

Обобщенная функция

```
template <typename T>
```

```
T sum(T a, T b) {
```

```
    return a + b;
```

```
}
```

template - ключевое слово, обозначающее, что дальше будет шаблонная функция

<typename T> - обозначение обобщенного аргумента

T используется как обозначение типа данных для параметров и (возможно) типа возвращаемого значения

Синтаксис

```
template <список обобщенных аргументов>  
тип возвращаемого значения имя(аргументы) {  
    //Тело функции  
}
```

```
template<class T>  
T sum(T a, T b) {  
    return a + b;  
}
```

```
template<class A, class B>  
void show(A v1, B v2) {  
    cout << v1 << " " << v2;  
}
```


Функция вывода

```
template <class T>
void show(T a) {
    cout << a;
}
```



Функция вывода

```
template <class T>
void show(T a) {
    cout << a;
}
```

Будет ли эта функция работать с классом Fraction?

Функция вывода

```
template <class T>
void show(T a) {
    cout << a;
}
```

Будет ли эта функция работать с классом Fraction?

Только если для этого класса перегружена операция вывода

Функция вывода

```
template <class T>
void show(T a) {
    cout << a;
}
```

Будет ли эта функция работать с
vector - вывести все элементы
массива?

Явная перегрузка обобщенной функции

```
template <class T>
void show(T a) {
    cout << a;
}
```

```
template <>
void show(vector<int> a) {
    for (int i = 0; i < a.size(); i++)
        cout << a[i] << " ";
}
```

Явная перегрузка обобщенной функции

```
template <class T>
void show(T a) {
    cout << a;
}
```

Можно ли сделать обычную
перегрузку функции, без
template<>?

```
template <>
void show(vector<int> a) {
    for (int i = 0; i < a.size(); i++)
        cout << a[i] << " ";
}
```

Явная перегрузка обобщенной функции

```
template <class T>
void show(T a) {
    cout << a;
}
```

Также можно явно указывать, для какого типа данных перегружать функцию, но не обязательно.

```
template <>
void show<vector<int>>>(vector<int> a) {
    for (int i = 0; i < a.size(); i++)
        cout << a[i] << " ";
}
```

Практика



20 минут



Теория



15 минут



Можно ли обобщить классы?

Обобщенные (шаблонные) классы

```
template <class T>
class Array {
    vector<T> data;
public:
    Array();
    void read();

    T max();
};
```

Как и функции, классы можно обобщить, используя ключевое слово `template` и обобщенные аргументы.

```
void print();
```

Обобщенные (шаблонные) классы

```
template <class T>
class Array {
    vector<T> data;
public:
    Array();
    void read();

    T max();
};
```

Как и функции, классы можно обобщить, используя ключевое слово `template` и обобщенные аргументы.

```
void print();
```

Методы также могут использовать как тип данных `T`.

Обобщенные (шаблонные) классы

```
template <class T>
class Array {
    vector<T> data;
public:
    Array();
    void read();

    T max();
};
```

//Где-то в программе

```
Array<int> a1;
Array<string> a2;
```

Как и функции, классы можно обобщить, используя ключевое слово `template` и обобщенные аргументы.

```
void print();
```

Методы также могут использовать как тип данных `T`.

При создании объектов необходимо явно указывать тип данных, с которым должен работать этот объект.

**Можно ли явно перегрузить
поведение класса под
конкретный тип данных?**

**Можно. Делается аналогично
? перегрузкам функций.**

Практика



25 минут



Итоги урока

- 1) Как объединить одинаковые функции в одну
- 2) Шаблонные функции
- 3) Обобщенные аргументы
- 4) Явная перегрузка шаблонных функций
- 5) Шаблонные классы