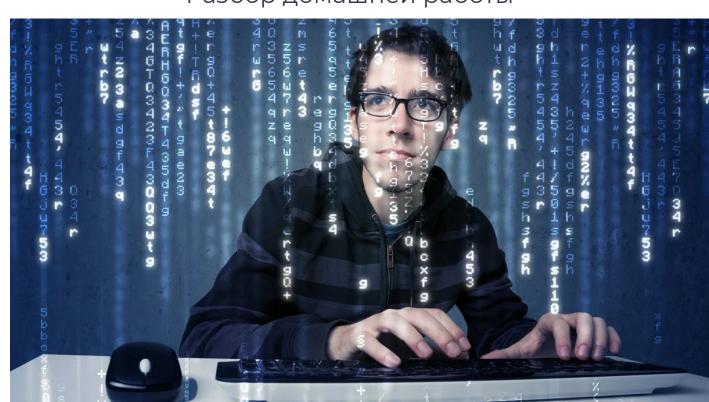




Разбор домашней работы



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) {
                           Все три выполняют
                                                  одну
   return a + b;
                           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) {
                           Все три выполняют
                                                  одну
   return a + b;
                           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) {
                           Все три выполняют
                                                 одну
   return a + b;
                           задачу - сложение
                                                 ДВУХ
                           переменных
double sum (double a, double b) {
                                   Они работают с разными
  return a + b;
                                   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?

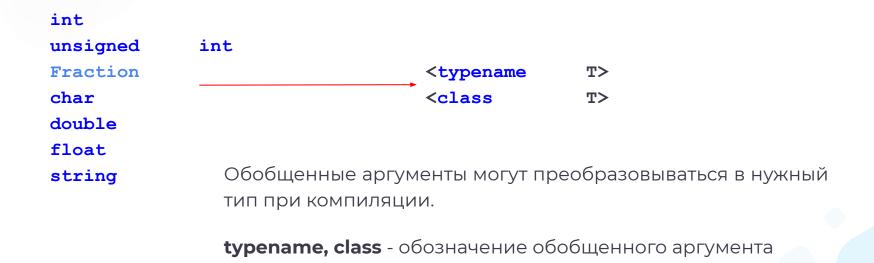


Обобщенные функции. Шаблоны

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



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



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

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

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

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

<typename Т> - обозначение обобщенного аргумента

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

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

<typename Т> - обозначение обобщенного аргумента

т 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>
                      template<class A, class B>
T \text{ sum}(T \text{ a, } T \text{ b})  { void show(A v1, B v2) {
                        cout << v1 << "" << v2;
  return a + b;
```

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



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

Будет ли эта функция работать с классом Fraction?

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

Будет ли эта функция работать с классом Fraction?
Только если для этого класса перегружена операция вывода



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

Будет ли эта функция работать с vector - выводить 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++)</pre>
        cout << a[i] << " ";
```

Явная перегрузка обобщенной функции

функции,

без

```
Можно ли сделать обычную
template <class T>
void show(T a) {
                                перегрузку
                                 template<>?
   cout << a;
template <>
void show(vector<int> a) {
    for (int i = 0; i < a.size(); i++)</pre>
        cout << a[i] << " ";
```

Явная перегрузка обобщенной функции

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

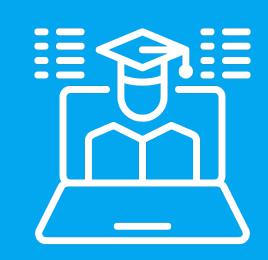
Также можно явно указывать, для какого типа данных перегружать функцию, но не обязательно.



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 \mathbf{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 ${f T}$.

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



Можно ли явно перегрузить поведение класса под конкретный тип данных?

31 ====

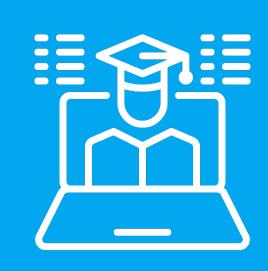
Можно. Делается аналогично ? перегрузкам функций.



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