#include <iostream>

using namespace std;

//ситуация: есть два числа a и b, допустим, сначала они были целочисленные,

//и их нужно сравнить. Для этого прописываем ф-ию с целочисленным типом.

//затем нам может понадобиться домножить числа a и b на константы (например 3.14 и 6.48)

//и вновь сравнить числа

//для этого придется поменять тип данных у функции

int min(int a, int b)

{

if (a < b)

{

return a;

}

return b;

}

double min(double a, double b)

{

if (a < b)

{

return a;

}

return b;

}

#include <iostream>

using namespace std;

//ситуация: есть два числа a и b, допустим, сначала они были целочисленные,

//и их нужно сравнить. Для этого прописываем ф-ию с целочисленным типом.

//затем нам может понадобиться домножить числа a и b на константы (например 3.14 и 6.48)

//и вновь сравнить числа

//для этого придется поменять тип данных у функции

template<typename Type>

Type \_min(Type a, Type b)

{

if (a < b)

{

return a;

}

return b;

}

int main(int argc, char\*\* argv)

{

cout << \_min(1, 2) << endl;

cout << \_min(3.1, 1.2) << endl;

return 0;

}

1. Проблема перегрузки функций

#include <iostream>

using namespace std;

// Проблема перегрузки функций - в некоторых ситуациях

// нужно прописывать слишком много различных вариантов

// перегруженных функций

int sum(int x, int y) {

cout << "Int\n";

return x + y;

}

int sum(double x, double y) {

cout << "Double\n";

return x + y;

}

int sum(double x, int y) {

cout << "Double + Int\n";

return x + y;

}

int sum(int x, double y) {

cout << "Int + Double\n";

return x + y;

}

int main()

{

cout << sum(1, 2) << endl;

cout << sum(1.1, 2) << endl;

cout << sum(2, 3.5) << endl;

cout << sum(4.5, 6.7) << endl;

}

1. Простейший пример шаблонной функции

#include <iostream>

using namespace std;

template<typename T>

T sum(T x, T y) {

cout << typeid(x).name() << endl;

return x + y;

}

int main()

{

cout << sum(1.3, 2.3) << endl;

cout << sum(1, 2) << endl;

}

1. Пример использования нескольких типов:

#include <iostream>

using namespace std;

template<typename T, typename T2, typename T3>

T3 sum(T x, T2 y) {

cout << typeid(x).name() << endl;

cout << typeid(y).name() << endl;

return x + y;

}

int main()

{

cout << sum<int, double, int>(1.5, 2.5) << endl;

// Конкретные типы данных прописываются в том же порядке,

// в каком они объявлены после ключевого слова template

}

1. Пример простейшего шаблонного класса

#include <iostream>

using namespace std;

template<class T>

class Pos {

public:

T x;

T y;

Pos() = delete; // Удаляем конструктор по умолчанию

Pos(T&& x,T&& y) { // Конструктор для r-value

this->x = x; // выражений

this->y = y;

}

Pos(T& x, T& y) { // Конструктор для l-value

this->x = x; // выражений

this->y = y;

}

void print() {

cout << "x = " << x << " y = " << y << endl;

}

};

int main()

{

Pos<int> point(5, 6);

point.print();

}

1. Для шаблонного класса можно определить различное поведение в зависимости от типа данных

template<>

class Pos<string> { // Можем уточнить поведение шаблонного

public: // класса для определённого типа данных

string x;

string y;

Pos(string x, string y) : x(x), y(y) {} // Конструктор

// через список инициализации

void print() {

cout << x << " " << y;

}

};

1. Наследование шаблонных классов

#include <iostream>

using namespace std;

template<class T>

class Pos {

public:

T x;

T y;

Pos() = delete;

Pos(T x, T y) {

this->x = x;

this->y = y;

}

virtual void print() {

cout << "x = " << x << " y = " << y << endl;

}

};

template<class T>

class Pos3D : public Pos<T> {

public:

T z;

Pos3D(T x, T y, T z) : Pos<T>(x, y), z(z) {}

void print() override {

cout << "x = " << this->x << " y = " << this->y << " z = " << z;

}

};

int main()

{

Pos3D<int> point3d(1, 2, 3);

point3d.print();

}

1. Класс, наследуемый от шаблонного класса не обязательно должен быть шаблоном

class Pos3D : public Pos<int> {

public:

int z;

Pos3D(int x, int y, int z) : Pos<int>(x, y), z(z) {}

void print() override {

cout << "x = " << this->x << " y = " << this->y << " z = " << z;

}

};

1. Преобразование типов для шаблонных классов

template<class T>

class Pos3D : public Pos<T> {

public:

T z;

Pos3D(T x, T y, T z) : Pos<T>(x, y), z(z) {}

void print() override {

((Pos<T>)(\*this)).print();

cout << "z = " << z;

}

};