Пример 03.01. Ссылки lvalue и rvalue.

```
# include <iostream>

void f(int) {}

int main()
{
	int i = 0;

	int& lref1 = i;
	int& lref2(i);
	int& lref3{ i };
	int& lref4 = { i };

	int& lv1 = i;				// Ok!
	int& lv2 = 2;				// Error!	(не lvalue)
	int& lv3 = i + 1;			// Error!	(не lvalue)
	const int& lv4 = i + 1;		// Ok!		(создается объект)
	++lv1;						// Ok!

	int&& rv1 = i;				// Error!	(не rvalue)
	int&& rv2 = 2;				// Ok!
	int&& rv3 = i + 1;			// Ok!
	const int&& rv4 = i + 1;	// Ok!
	++rv2;										// Ok!

	int&& rv5 = rv2;			// Error!	(не rvalue)
	int& lv5 = rv2;				// Ok!

	int&& rv6 = (int)i;			// Ok! ( int(i) )
	int&& rv7 = std::move(i);	// Ok!

	void(&reff)(int) = f;
	reff(1);
}
```

Пример 03.02. rvalue ссылки.

```
# include <iostream>

using namespace std;

int main()
{
	int i = 0;
	int&& rv1 = i + 0;
	int&& rv2 = move(i);
	int&& rv3 = (int)i;

	++i;

	int&& rv4 = 5;
	++rv4;

	cout << "rv1 = " << rv1 << "; rv2 = " << rv2 << "; rv3 = " << rv3 << endl;
	cout << "rv4 = " << rv4 << endl;
}
```

Пример 03.03. Перегрузка функций с параметром lvalue и rvalue.

```
# include <iostream>

using namespace std;

int func(int& ref)  { cout << "lvalue - " << ref << endl; return ++ref; }
int func(int&& ref) { cout << "rvalue - " << ref << endl; return ++ref; }

int main()
{
```

```
	int i = 0;

	func(i);
	func(move(i));
	func(i + 1);
	func(func(i));

	cout << "i = " << i << endl;
}
```

Пример 03.04. Перегрузка функций.

```
# include <iostream>

using namespace std;

void func1(int& x) { cout << "func1(int&)" << endl; }
void func1(const int& x) { cout << "func1(const int&)" << endl; }

void func2(int x) { cout << "func2(int)" << endl; }
void func2(int& x) { cout << "func2(int&)" << endl; }

void func3(const int& x) { cout << "func3(const int&)" << endl; }
void func3(int&& x) { cout << "func3(int&&)" << endl; }

void func4(int& x) { cout << "func4(int&)" << endl; }
void func4(int&& x) { cout << "func4(int&&)" << endl; }

void func5(int x) { cout << "func5(int)" << endl; }
void func5(int&& x) { cout << "func5(int&&)" << endl; }

void func6(int x) {}
void func6(const int& x) {}

int main()
{
	int i = 0;
	const int ci = 0;
	int& lv = i;
	const int& clv = ci;
	int&& rv = i + 1;

	func1(i);		// int&
	func1(ci);		// const int&
	func1(lv);		// int&
	func1(clv);		// const int&
	func1(rv);		// int&
	func1(i + 1);		// const int&
	cout << endl;

//	func2(i);		// Error!
	func2(ci);		// int
//	func2(lv);		// Error!
	func2(clv);		// int
//	func2(rv);		// Error!
	func2(i + 1);		// int
	cout << endl;

	func3(i);		// const int&
	func3(ci);		// const int&
	func3(lv);		// const int&
	func3(clv);		// const int&
	func3(rv);		// const int&
	func3(i + 1);		// int&&
	cout << endl;

	func4(i);		// int&
//	func4(ci);		// Error!
	func4(lv);		// int&
//	func4(clv);		// Error!
	func4(rv);		// int&
```

```
	func4(i + 1);		// int&&
	cout << endl;

	func5(i);		// int
	func5(ci);		// int
	func5(lv);		// int
	func5(clv);		// int
	func5(rv);		// int
//	func5(i + 1);		// Error!

//	func6(i);		// Error!
//	func6(ci);		// Error!
//	func6(lv);		// Error!
//	func6(clv);		// Error!
//	func6(rv);		// Error!
//	func6(i + 1);		// Error!
}
```

Пример 03.05. Автоматическое выведение типа.

```
# include <iostream>

int main()
{
	int i = 0;
	const int ci = 0;
	int& lv = i;
	const int& clv = ci;
	int&& rv = i + 1;

	// тип int
	{
		auto x1 = i;
		auto x2 = ci;
		auto x3 = lv;
		auto x4 = clv;
		auto x5 = rv;
		auto x6 = i + 1;
	}

	// тип int&
	{
		auto& refx1 = i;
		auto& refx3 = lv;
		auto& refx5 = rv;
	}

	// тип const int&
	{
		auto& crefx2 = ci;
		auto& crefx4 = clv;
	}

	// тип const int&
	{
		const auto& crefx1 = i;
		const auto& crefx2 = ci;
		const auto& crefx3 = lv;
		const auto& crefx4 = clv;
		const auto& crefx5 = rv;
		const auto& crefx6 = i + 1;
	}

	// тип int&
	{
		auto&& refx1 = i;
		auto&& refx3 = lv;
		auto&& refx5 = rv;
	}

	// тип const int&
```

```
        {
                auto&& crefx2 = ci;
                auto&& crefx4 = clv;
        }

        // тип int&&
        {
                auto&& refx6 = i + 1;
        }
}
```