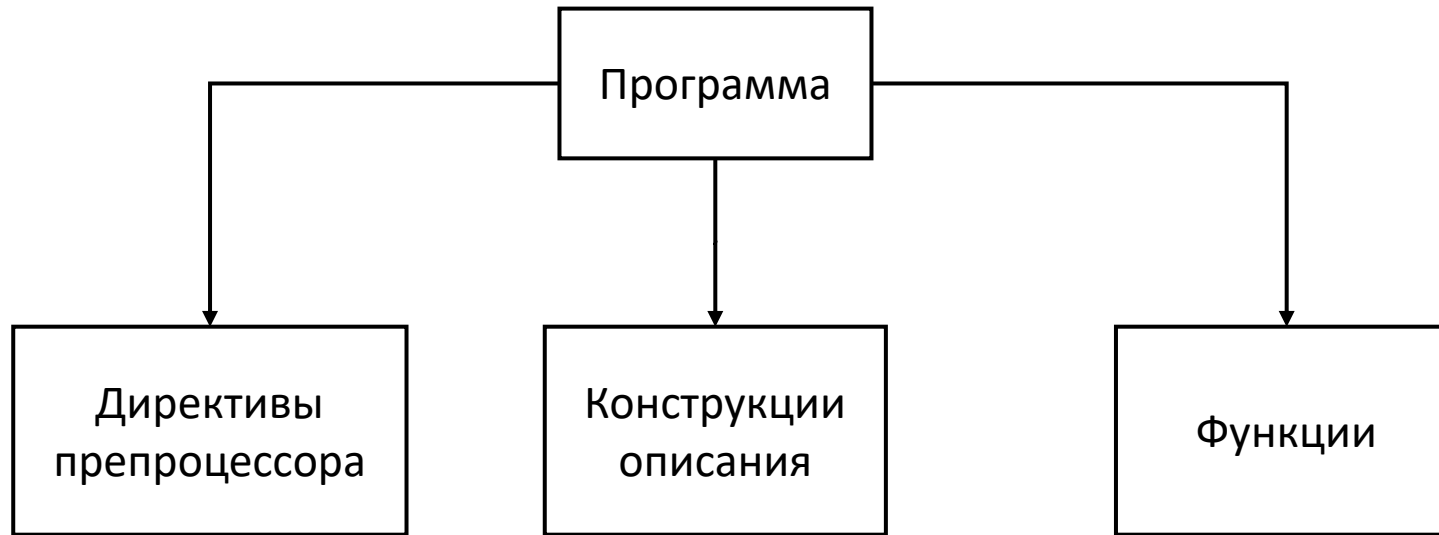


Алгоритмизация и программирование

Лекция 2

Базовая структура программы



Директивы препроцессора

include

```
#include <имя файла>
```

```
#include "имя файла"
```

На сленге include подключает библиотеку.

На самом деле – выполняется замена директивы на содержимое файла с указанным именем. Файл предварительно обрабатывается препроцессором.

Разница между <> и "" в том, с какого каталога начинается поиск файла.

Вариант с кавычками начинает поиск с текущей папки (та пака в которой лежит компилируемый сpp) и в случае, если файла нет, выполняется так же, как и вариант <>

<https://wandbox.org/permlink/xpWrQLFP5tslgNVU>

Statements

Statements

Statements - это фрагменты программы на C++, которые выполняются последовательно.
Тело любой функции - это последовательность statement-ов.

Types of statements

- 1) labeled statements (метка);
- 2) expression statements (выражение);
- 3) compound statements (блок или составной оператор);
- 4) selection statements (операторы выбора);
- 5) iteration statements (операторы цикла);
- 6) jump statements (операторы перехода);
- 7) declaration statements (операторы объявления);
- 8) try blocks;
- 9) atomic and synchronized blocks (TM TS).

Statements

```
#include <iostream>
```

```
int main(){
```

```
    int a;
```

declaration statement

```
    std::cin >> a;
```

expression statement

```
    if (a > 0) std::cout << "pos";
```

```
    else std::cout << "neg";
```

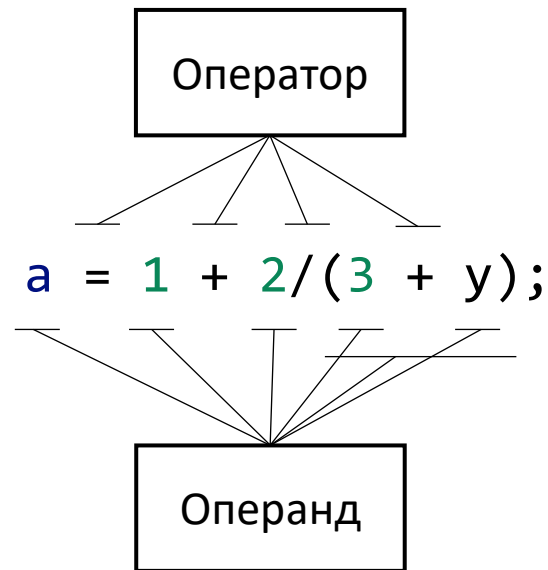
selection statement

```
}
```


Выражения

Выражения

Выражение состоит из операторов и операндов. Операнд может тоже быть выражением.



Виды выражений

- Просто символ ; - пустое выражение
- Выражение, которое не является частью другого выражения – полное выражение (заканчивается ;)

Операторы

Основные операторы

Общие операторы						
Присваивание	Инкремент Декремент	Арифметические	Логические	Сравнение	Доступа	Остальные
a = b a += b a -= b a *= b a /= b a %= b a &= b a = b a ^= b a <<= b a >>= b	++a --a a++ a--	+a -a a + b a - b a * b a / b a % b ~a a & b a b a ^ b a << b a >> b	!a a && b a b	a == b a != b a < b a > b a <= b a >= b a <=> b	a[b] *a &a a->b a.b a->*b a.*b	Вызов функции
		a (...)				
		Запятая				
		a, b				
		Тернарный				
		a ? b : c				
		Специальные операторы				
static_cast преобразование одного типа в другой dynamic_cast преобразование типов с учётом иерархий наследования const_cast добавляет или удаляет cv -квалификаторы reinterpret_cast преобразование типа без учёта их связи C-style_cast преобразование типов в смешанном режиме <code>static_cast</code> , <code>const_cast</code> , и <code>reinterpret_cast</code> new создаёт новый объект с динамическим временем жизни delete уничтожает объекты, ранее созданные оператором <code>new</code> , и освобождает полученную область памяти sizeof возвращает размер типа sizeof... возвращает размер пачки parameter pack (since C++11) typeid возвращает информацию о типе noexcept проверяет есть ли в выражении исключения (since C++11) alignof определяет выравнивание необходимое для типа (since C++11)						

Приоритет и ассоциативность

Приоритет	Оператор	Пояснение	Ассоциативность
1	::	Scope resolution	Left-to-right →
2	a++ a--	Suffix/postfix increment and decrement	
	type() type{}	Functional cast	
	a() a[] . ->	Function call Subscript Member access	
3	++a --a	Prefix increment and decrement	Right-to-left ←
	+a -a	Unary plus and minus	
	! ~	Logical NOT and bitwise NOT	
	(type)	C-style cast	
	*a	Indirection (dereference)	
	&a	Address-of	
	sizeof	Size-of ^[note 1]	
	co_await	await-expression (C++20)	
	new new[]	Dynamic memory allocation	
	delete delete[]	Dynamic memory deallocation	
4	.* ->*	Pointer-to-member	Left-to-right →
5	a*b a/b a%b	Multiplication, division, and remainder	
6	a+b a-b	Addition and subtraction	

7	<< >>	Bitwise left shift and right shift	Left-to-right →
8	<=>	Three-way comparison operator (since C++20)	
9	< <= > >=	For relational operators < and ≤ and > and ≥ respectively	
10	== !=	For equality operators = and ≠ respectively	
11	a&b	Bitwise AND	
12	^	Bitwise XOR (exclusive or)	
13		Bitwise OR (inclusive or)	Right-to-left ←
14	&&	Logical AND	
15		Logical OR	
16	a?b:c	Ternary conditional ^[note 2]	
	throw	throw operator	
	co_yield	yield-expression (C++20)	
	Direct	Direct assignment (provided by default for C++ classes)	
	=	Compound assignment by sum and difference	
	+= -=	Compound assignment by product, quotient, and remainder	
17	*= /= %=	Compound assignment by bitwise left shift and right shift	
	<<= >>=	Compound assignment by bitwise AND, XOR, and OR	
	&= ^= =		
17	,	Comma	Left-to-right →

Блок или составной оператор

Составной оператор

- Составной оператор выглядит как пара фигурных скобок `{ }`.
- Составной оператор можно использовать в любом месте кода, где ожидается один стэйтмент.
Составной оператор служит для группировки нескольких стэйтментов.
- Составной оператор создаёт локальную область видимости. Переменные созданные внутри блока автоматически уничтожаются при выходе из него.

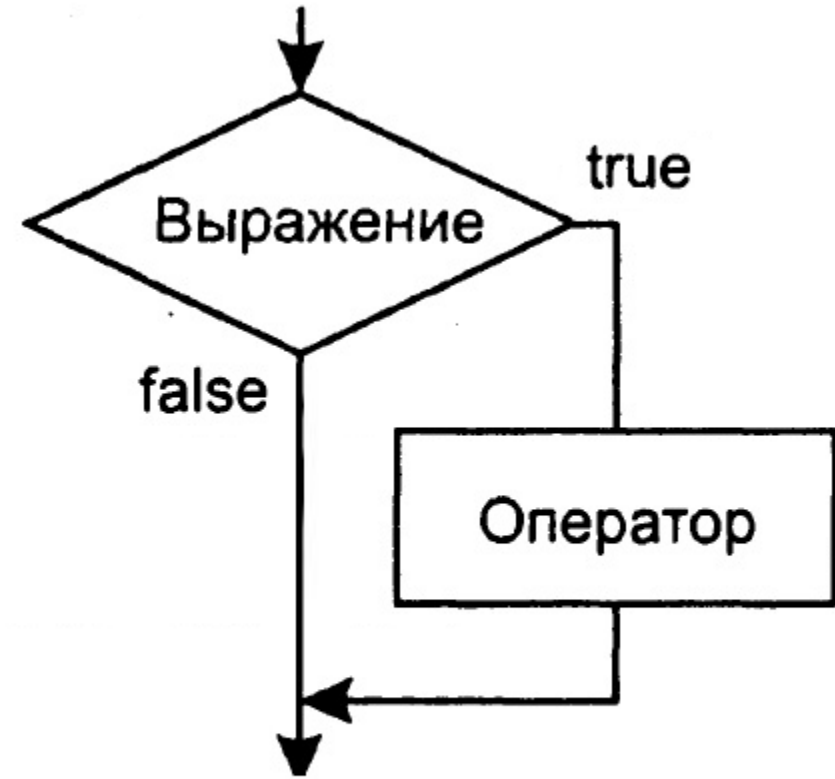
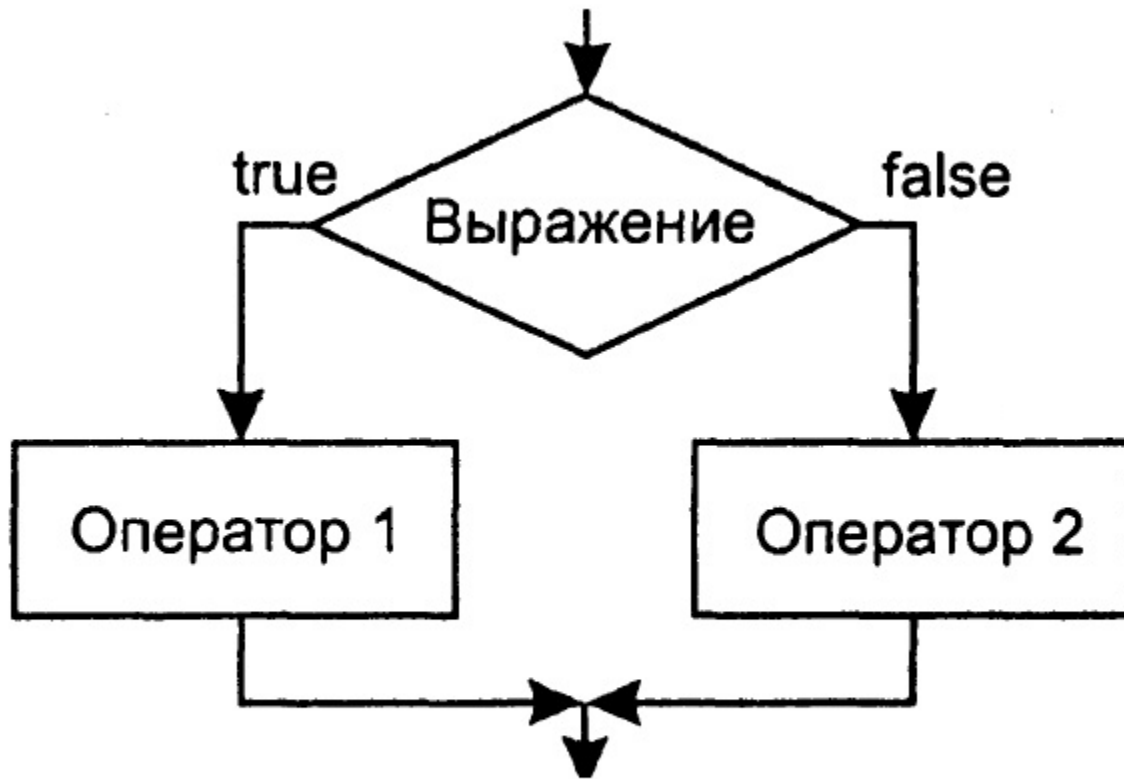
Составной оператор

```
#include <iostream>

int main(){
    int i = 1;
    {
        int i = 2, j = 3;
        std::cout << i;    // 2
    }
    std::cout << j;        // Ошибка
}
```

Операторы выбора (ветвления)

Условный оператор



Структурная схема условного оператора

if

Ключевое слово **если**

Выражение

Тело

The diagram shows the components of a C++ if statement. An arrow points from the text 'Ключевое слово если' to the word 'if'. Another arrow points from 'Выражение' to the expression '(a > b)'. A third arrow points from 'Тело' to the statement 'std::cout << "Hello";'. The code is: `if (a > b) std::cout << "Hello";`

Ключевое слово **если**

Выражение

Тело

The diagram shows the components of a C++ if statement with a multi-line body. An arrow points from the text 'Ключевое слово если' to the word 'if'. Another arrow points from 'Выражение' to the expression '(a > b)'. A third arrow points from 'Тело' to the block of code '{ std::cout << "Hello"; }'. The code is: `if (a > b){
 std::cout << "Hello";
}`

if (выражение)

Ожидается, что выражение в скобках типа `bool` или преобразуемо в него.

Если выражение НЕ типа `bool`, то будет предпринята попытка неявно [преобразовать его к `bool`](#).

Если преобразование не допустимо, то – ошибка.

```
true // bool
```

```
1 // Преобразуется в true
```

```
"world" // Преобразуется в true
```

```
std::string("world") // Нет преобразования - ошибка
```

if-else

Выражение



Ключевое слово **если** → **if** (a > b) **std::cout** << "Hello"; ← Тело **if**

Ключевое слово **иначе** → **else** **std::cout** << "Bye"; ← Тело **else**

Выражение



Ключевое слово **если** → **if** (a > b){
 std::cout << "Hello"; ← Тело **if**

Ключевое слово **иначе** → } **else** {
 std::cout << "Bye"; ← Тело **else**
}

if(инициализация; проверка)

init-statement
↓
if (int res = a > b; res) std::cout << "Hello";
else std::cout << "Bye";

if

```
if (/* условие */)
{
    /* true */
}
```

If-else

```
if (/* условие */)
{
    /* true */
}
else
{
    /* false */
}
```

If-else if

```
if (/* условие 1 */)
{
    /* true */
}
else if (/* условие 2 */)
{
    /* true */
}
else
{
    /* false */
}
```

Вложенный if

```
if (/* условие */)
{
    if (/* условие */)
    {
        /* true */
    }
    else
    {
        /* false */
    }
}
else
{
    if (/* условие */)
    {
        /* true */
    }
    else
    {
        /* false */
    }
}
```


If-else (ошибки)


```
if (a > b);
```

```
if (a > b){/* код */};  
else {/* код */}
```

```
if (a > b)  
    if (a > c) std::cout << "Hello";  
else std::cout << "Bye";
```

Тернарный оператор (?:)

```
variable = a > b ? a : b;
```



The diagram illustrates the logic of the ternary operator. It shows a horizontal line representing the expression: `/* результат */ = /* условие */ ? /* выражение 1 */ : /* выражение 2 */;`. Above the line, the word "true" is centered. A horizontal line extends from the "true" label to the right, then turns down into an arrow pointing to the first expression after the colon, "/* выражение 1 */". Below the line, the word "false" is centered. A horizontal line extends from the "false" label to the left, then turns up into an arrow pointing to the second expression after the colon, "/* выражение 2 */".

```
/* результат */ = /* условие */ ? /* выражение 1 */ : /* выражение 2 */;
```

* выражение 1 и выражение 2 должны быть одного или приводимого к одному типу

Логические операторы

Название	Как выглядит		Как использовать	
И	&&	and	a && b;	a and b;
ИЛИ		or	a b;	a or b;
НЕ	!	not	!a;	not a;

a	b	a and b	a or b	not a
false	false	false	false	true
true	false	false	true	false
false	true	false	true	true
true	true	true	true	false

Логические операторы применяются только к операндам типа `bool`, поэтому перед их применением будет попытка преобразовать операнды в `bool`. Если это не возможно, то получаем ошибку.

Операторы И и ИЛИ вычисляются по сокращённым правилам, т.к. если результат можно получить вычислив первый аргумент, второй не вычисляется:

```
false && std::cout << 1; // пусто
```

```
true && std::cout << 2; // 2
```

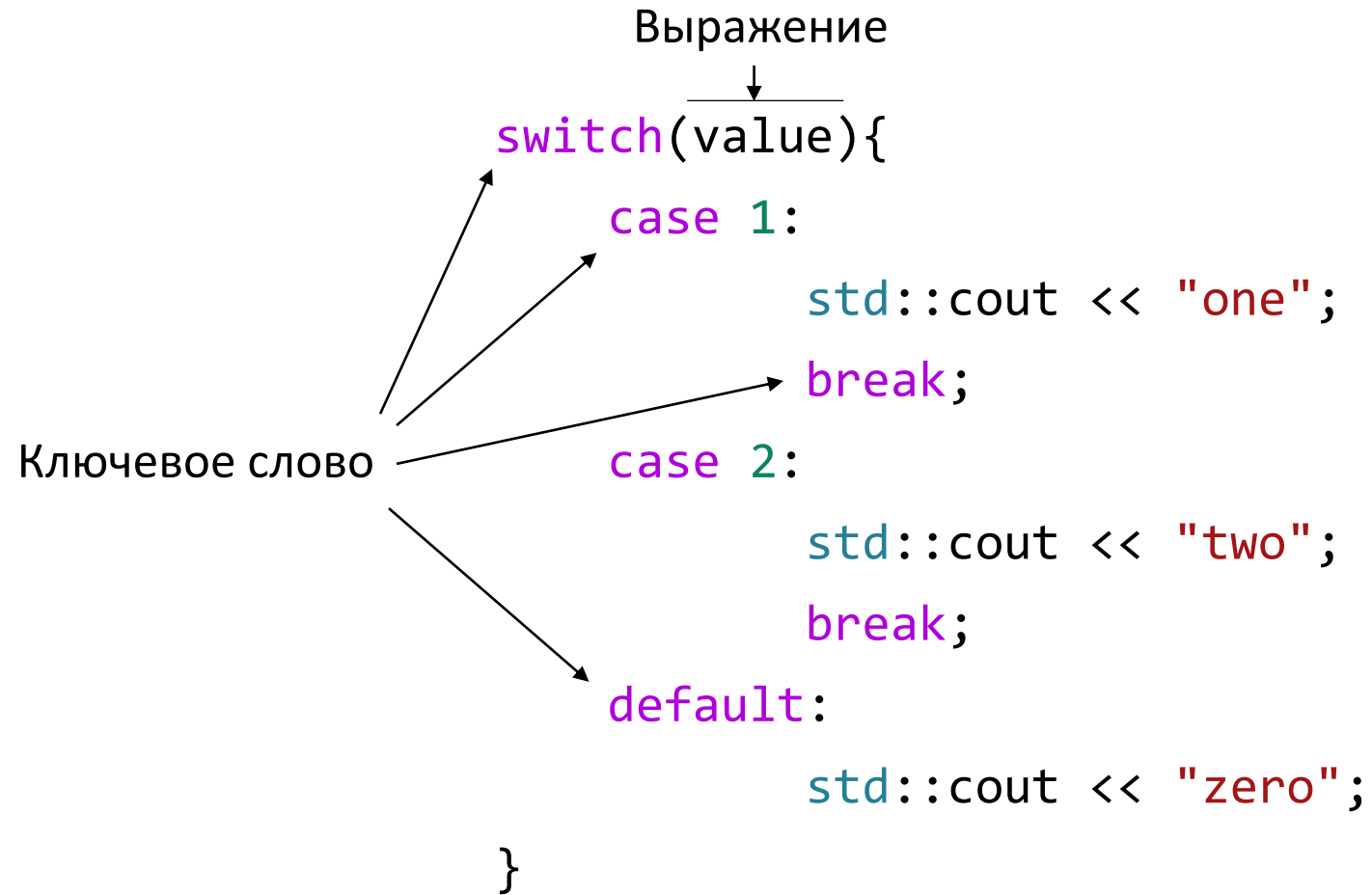
Логические операторы

```
int a = 5;  
(a > 1) and (a < 10) // true  
(a == 5) or (a == 10) // true  
not (a == 10) // true  
-1 < a < 2 // true
```

Последнее выражение вычисляется последовательно: $(-1 < a) < 2 \rightarrow \text{true} < 2$
если хотите получить результат по математическим правилам пишите: $(-1 < a) \text{ and } (a < 2)$

```
(a > 1) and (a < 3) // false  
(a == 1) or (a == 3) // false  
not (a == 5) // false
```

switch



switch (выражение)

Выражение перечислимого типа (целого), enum или что-то, что можно преобразовать в эти типы.

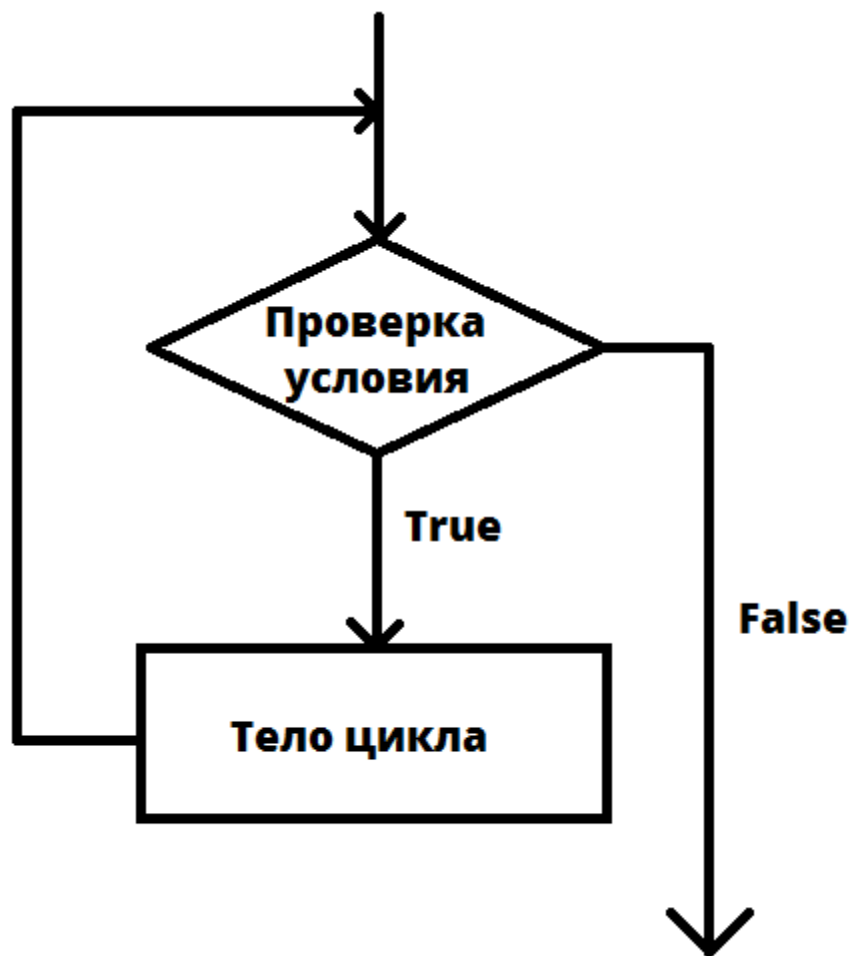
switch(инициализация; выражение)

init-statement
↓

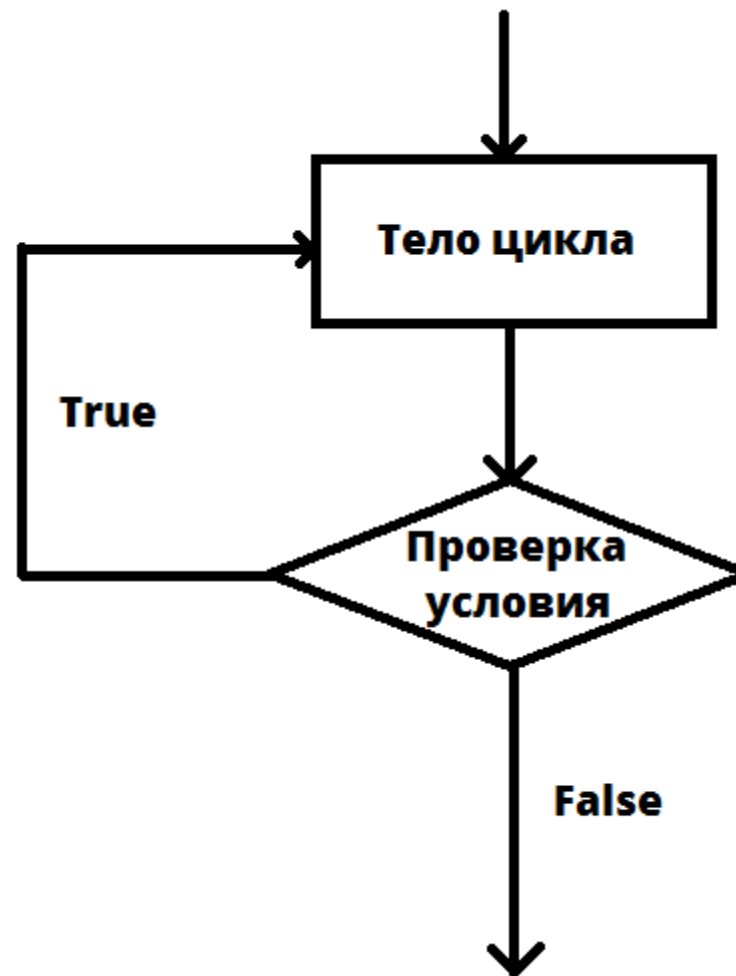
```
switch (int res = a > b; res ){  
    case true: std::cout << " :)"; break;  
    default: std::cout << " :(";  
}
```

Операторы цикла (iteration)

Оператор цикла



while



do-while

while

Ключевое слово

Выражение

Тело

```
while (a > b) std::cin >> a;
```

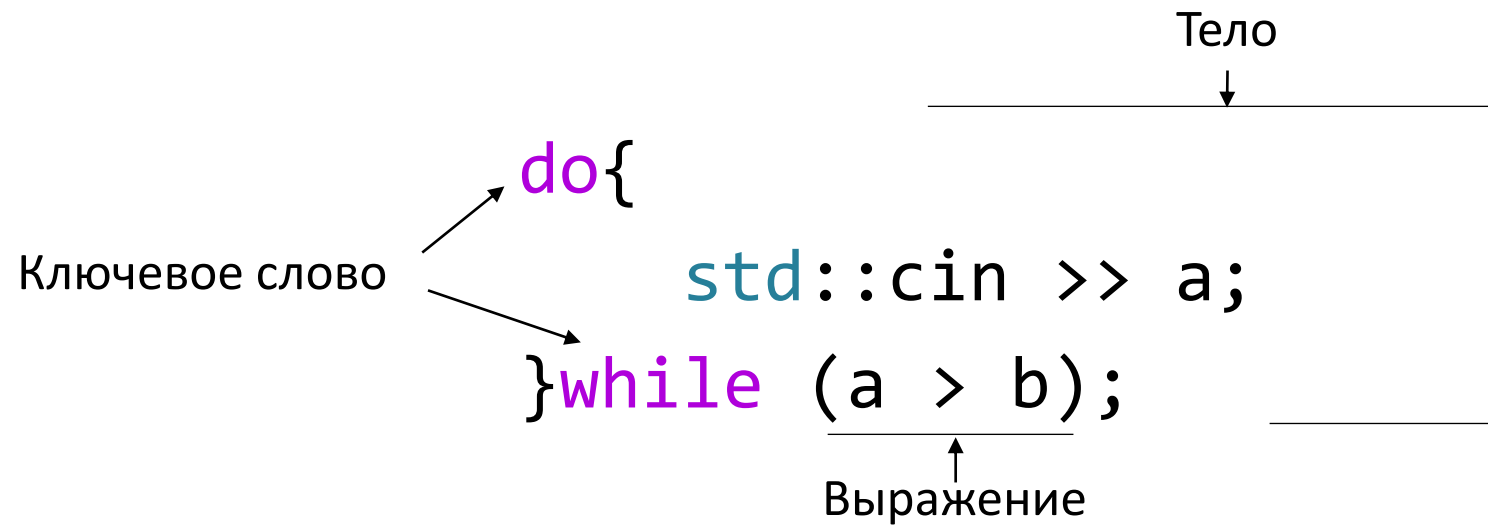
Ключевое слово

Выражение

Тело

```
while (a > b){
    std::cin >> a;
}
```

do-while

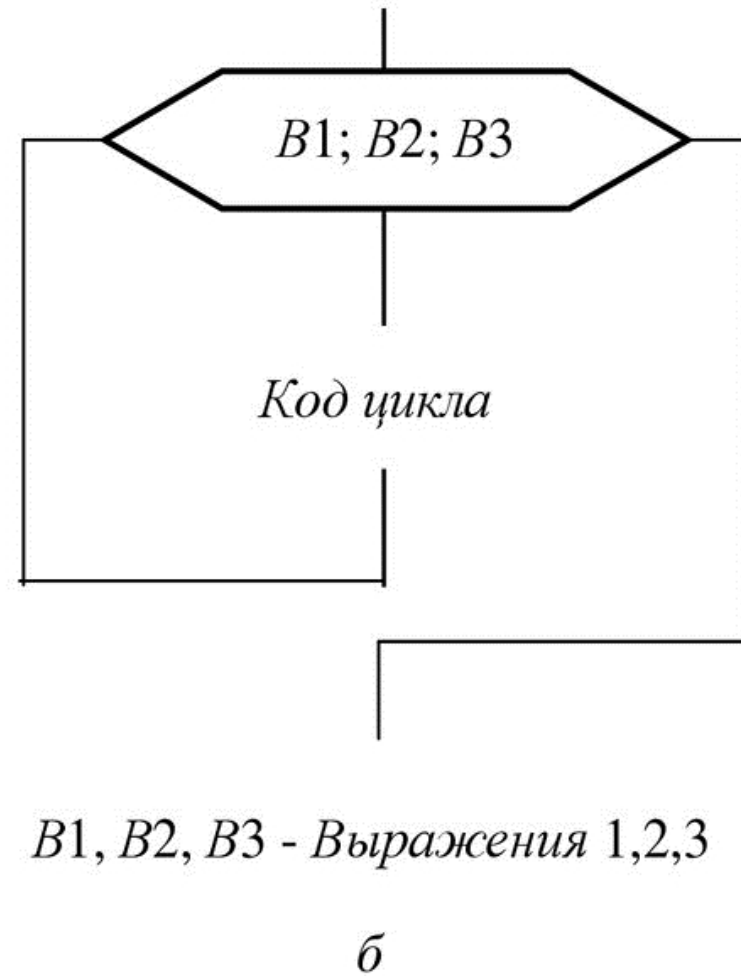
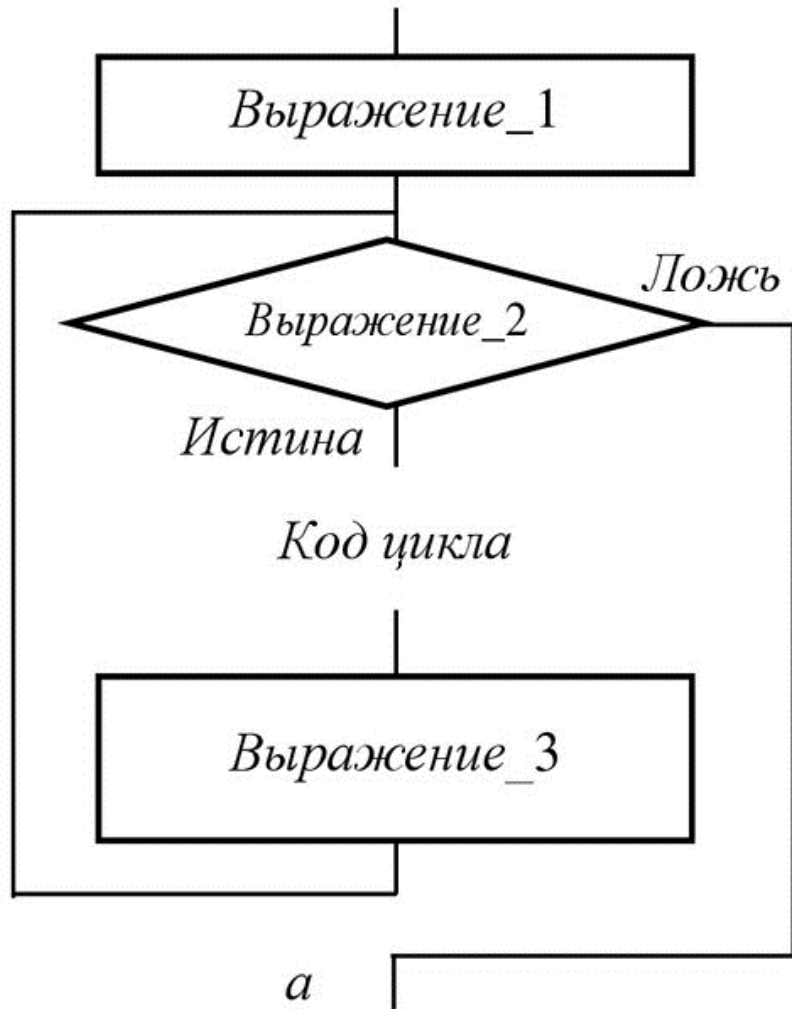


while (выражение)

Ожидается, что выражение в скобках типа `bool`, поэтому будет попытка неявно [преобразовать его к `bool`](#).

Если преобразование не допустимо, то – ошибка.

Оператор цикла for



for

Ключевое слово

Выражение1

Выражение2

Выражение3

Тело

```
for(int i=0; i < count; i++) std::cout << i;
```

Ключевое слово

Выражение1

Выражение2

Выражение3

Тело

```
for(int i=0; i < count; i++){  
    std::cout << i;  
}
```

for (выражение1; выражение2; выражение3)

Выражение1 – любое выражение или инициализация переменной. Обычно - инициализация переменной счётчика или нескольких;

Выражение2 – любое выражение или инициализация переменной. Обычно - выражение проверяющее условие работы цикла. Если выражение не указано, то считается, что оно равно true.

Выражение3 – выражение. Обычно инкремент/декремент счётчика(ов).

* каждое из выражение не обязательное (можно не писать), но точки с запятой писать нужно.

range-based for

Ключевое слово Переменная Контейнер Тело

↓ ↓ ↓

```
for(auto i : array) std::cout << i;
```

Ключевое слово Переменный Выражение2 Тело

↓ ↓ ↓ ↓

for(auto [key, value] : mymap) std::cout << i;

for (range-declaration : range-expression)

range-expression – любое выражение, представляющее последовательность элементов (либо массив, либо объект, для которого определены методы или функции `begin` и `end`) или список инициализации.

range-declaration – объявление именованной переменной, тип которой является типом элемента последовательности, представленного **range-expression**, или ссылкой на этот тип. Часто использует спецификатор `auto` для автоматического определения типа.

for(инициализация; range-declaration : range-expression)

```
for(auto list = {1,2,3}; auto i : list){  
    std::cout << i;  
}
```