Операторы ветвления. Логические операции

Михаил Марков
C++ - Разработчик

Михаил Марков

О спикере:
C++-разработчик, фрилансер

- Разработка алгоритма для релевантной выдачи объявлений.
- Разработка эмуляторов оборудования

Вспоминаем прошлое занятие

Вопрос: что такое программа?

Вспоминаем прошлое занятие

Вопрос: что такое программа?

Ответ:

- Это один из способов автоматизации деятельности человека
- Набор инструкций, выполняемый компьютером, для достижения определённой цели

Вспоминаем прошлое занятие

Вопрос: что такое переменная?

Вспоминаем прошлое занятие

Вопрос: что такое переменная?

Ответ: это контейнер, в котором будет находиться какое-нибудь значение

Вспоминаем прошлое занятие

Вопрос: что такое приведение типа?

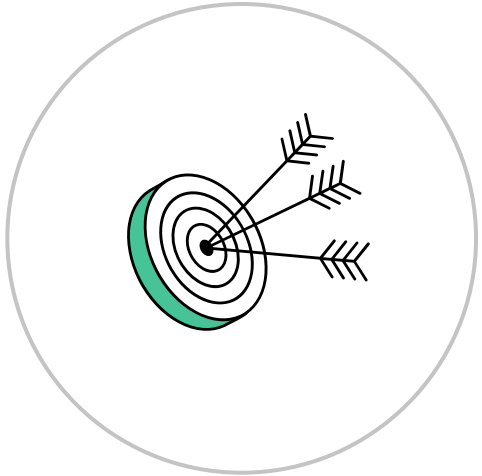
Вспоминаем прошлое занятие

Вопрос: что такое приведение типа?

Ответ: преобразование значения одного типа в значение другого типа

Цели занятия

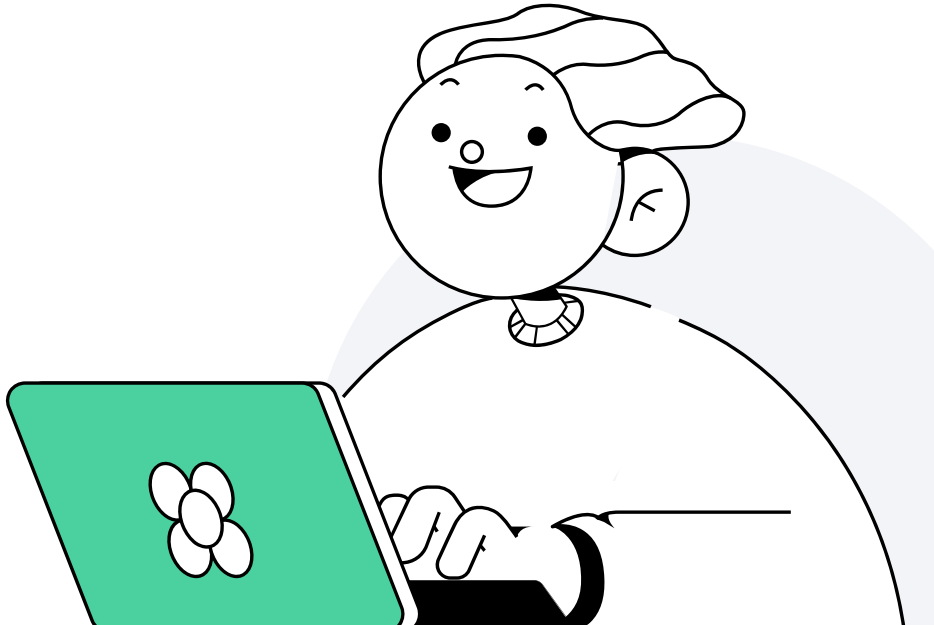
- Узнаем, что такое логический тип данных
- Познакомимся с операторами ветвления
- Разберём логические операции
- Разберём вложенные и тернарные операторы
- Научимся комбинировать операторы в условные выражения



План занятия

1. Логический тип данных
2. Операторы сравнения
3. Операторы ветвления
4. Логические операторы
5. Вложенные операторы
6. Оператор switch
7. Тернарный оператор
8. Итоги
9. Домашнее задание

*Нажмите на раздел для перехода



Логический тип данных

1

Логический тип данных

Логический тип данных может принимать одно из двух значений:

- true (1)
- false (0)

```
bool t = true, f = false;
std::cout << t << std::endl;
std::cout << f << std::endl;
```

bool — важный тип данных, т. к. все сравнения в языке приводятся к этому типу

Операторы сравнения

2

Операторы сравнения

==	**Равенство** возвращает **true**, если операнды равны
!=	**Неравенство** возвращает **true**, если операнды не равны
<	**Меньше чем** возвращает **true**, если левый операнд меньше правого
>	**Больше чем** возвращает **true**, если левый операнд больше правого
<=	**Меньше или равно** возвращает **true**, если левый операнд меньше или равен правому
>=	**Больше или равно** возвращает **true**, если левый операнд больше или равен правому

Операнд представляет собой некоторую величину, обрабатываемую в программе

Оператор равенства

```
bool t = true, f = false;
bool res;
res = (t == f);                 // false
res = (t == t);                 // true
res = (f == f);                 // true
res = (1 == 1);                 // true
res = (1 == 2);                 // false
res = (f == (1 == 2));          // ???
```

Напишите в чат, чему будет равно значение res в этом случае?

Оператор равенства

```
int i1, i2;
bool res;
i1 = 1 + 2 + 3;
i2 = 6;
res = (i1 == i2);     // true
char i1, i2;
i1 = 'a';
i2 = 'A';
res = (i1 == i2);     // false
```

Оператор неравенства

```
bool t = true, f = false;
bool res;
res = (t != f);           // true
res = (1 != 2);           // true
res = (t != (1 != 2));    // ???

i1 = 8 - 3;
i2 = 6;
res = (i1 != i2);         // true
```

Напишите в чат, чему будет равно значение res в этом случае?

Операторы сравнения

Оператор «меньше чем»:

```
res = ( 2 < 3);       // true
res = ( 2 < 2);       // false
```

Оператор «меньше или равно»:

```
res = ( 2 <= 3);                    // true
res = ( 2 <= 2);                    // true

res = (2 + 4 + 9 >= 8 + 5 + 1); // ???
```

Напишите в чат, чему будет равно значение res в этом случае?

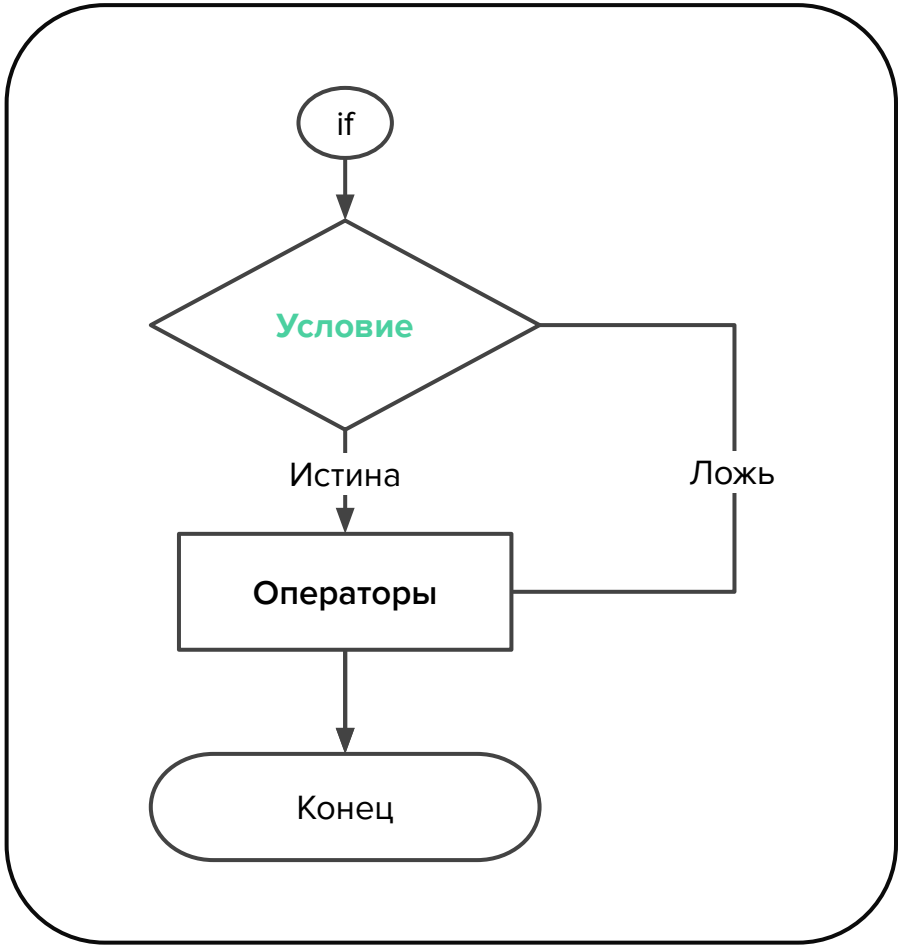
Операторы ветвления

3

Оператор if

Если условие истинно, то будут выполнены операторы в блоке if, иначе программа продолжит работу без выполнения операторов

```
if (условие) {
    операторы
}
```



Оператор if

Оператор **if** определяет, какой оператор будет выполняться при выполнении условия, заданного условным выражением.

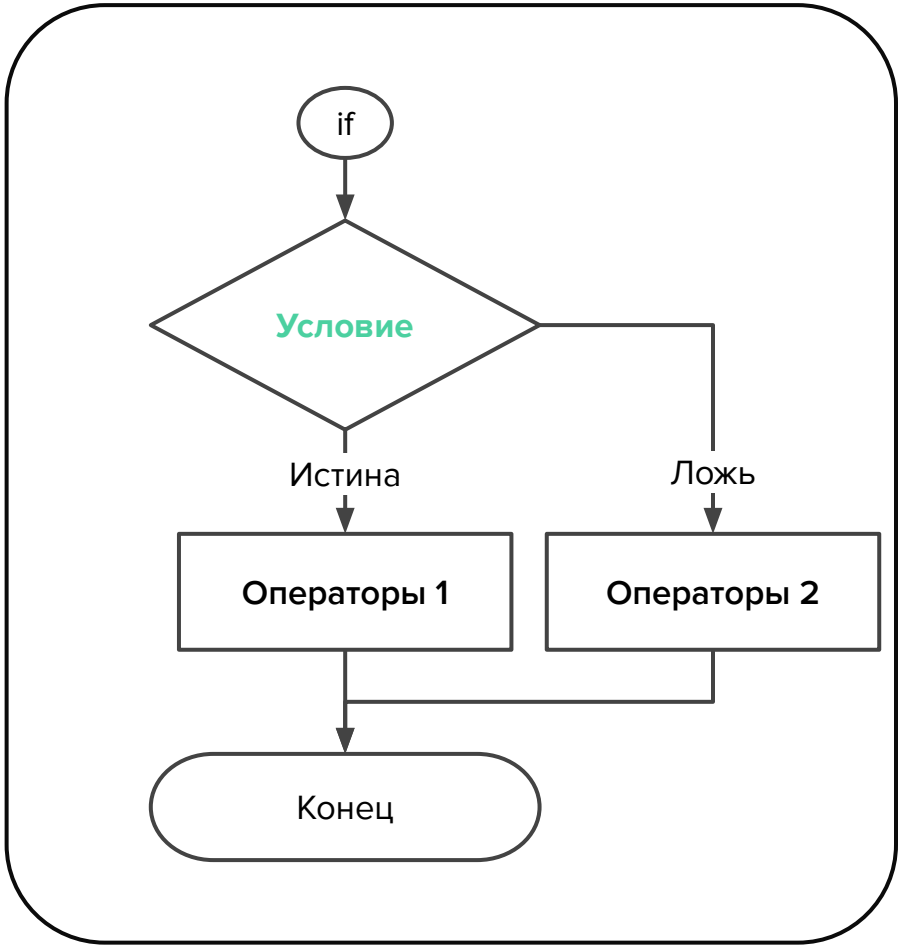
```
bool a = true;
if (a) {
    std::cout << "a is true" << std::endl;
}
```

Если условное выражение имеет значение false, то управление передаётся следующему оператору после оператора **if**

Оператор if-else

Если условие истинно, то будут выполнены операторы в блоке if, если ложно, то будут выполнены операторы в блоке **else**

```
if (условие) {
    операторы 1;
} else {
    операторы 2;
}
```



Оператор if-else

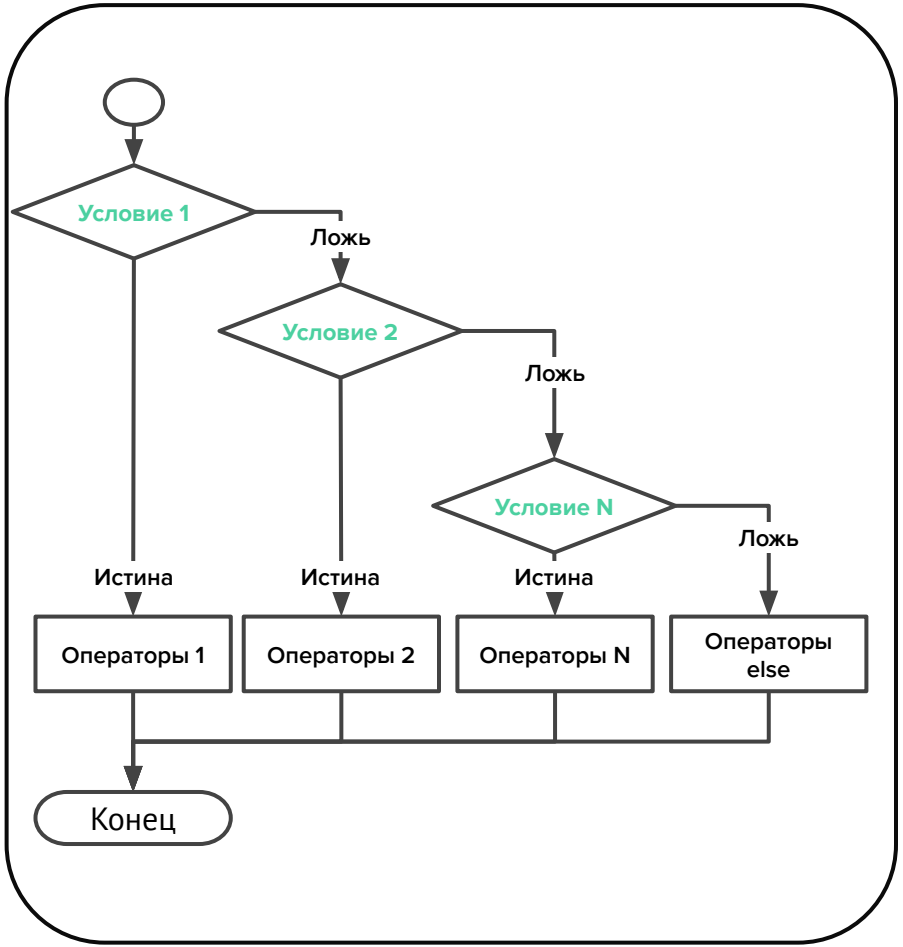
В операторе **if-else**, если условное выражение имеет значение **true**, выполняется первый блок. Если условное выражение имеет значение **false**, выполняется второй блок. Так как условное выражение не может одновременно иметь значения **true** и **false**, блоки **if-else** не могут выполняться одновременно

```
bool a = true;
if (a) {
    std::cout << "a is true" << std::endl;
} else {
    std::cout << "a is false" << std::endl;
}
```

Оператор if-else-if-else

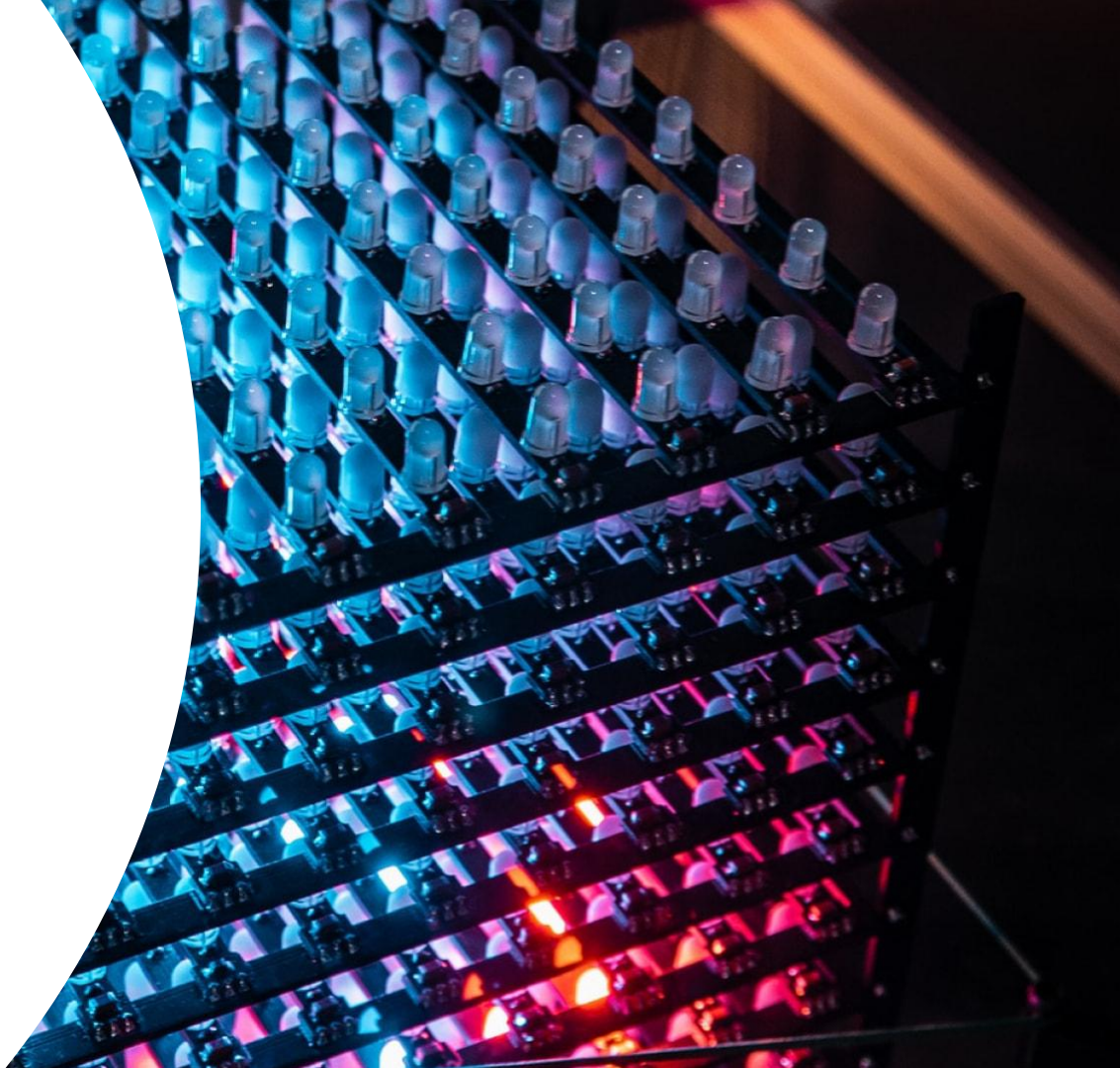
Если условие истинно, то будут выполнены операторы в блоке if, если ложно, то управление будет передано следующему блоку else-if для проверки следующего условия. Если все условия ложны, то будут выполнены операторы в блоке else

```
if (условие 1) {
  операторы 1;
} else if (условие 2) {
    операторы 2;
...
} else if (условие N) {
    операторы N;
} else {
    операторы else;}
```



Разберём пример

if-else-if-else



Оператор if-else-if-else

Оператор **if-else-if-else** позволяет проверить несколько взаимоисключающих условий

```
int a = 0;
if (a > 0) {
    std::cout << "a is positive" << std::endl;
} else if (a < 0) {
    std::cout << "a is negative" << std::endl;
} else {
    std::cout << "a is zero" << std::endl;
}
```

Ссылка на готовый код в Repl

Логические операторы

4

Логические операторы

Логические операторы определены для типа **Boolean** и позволяют вычислять простые или составные логические выражения

!	Оператор **отрицание** возвращает **true**, если операнд **false**
&&	Оператор **И** возвращает **true**, если оба операнда **true**
||	Оператор **ИЛИ** возвращает **true**, если хотя бы один из двух операндов **true**

Оператор логического отрицания

Унарный префиксный оператор **!** выполняет логическое отрицание операнда, возвращая **false**, если операнд имеет значение **true**. И **true**, если операнд имеет значение **false**

```
bool a = true;
bool b = !a;
```

a	!a
true	**false**
false	**true**

Оператор И

Условный оператор **&&** вычисляет логическое **И** для своих операндов. Результат операции **x && y** принимает значение **true**, если оба оператора **x** и **y** имеют значение **true**. В противном случае результат будет **false**.

```
bool a = true;

bool b = false;

bool c = a && b;
```

Особенность: оператор **&&** вычисляет правый операнд, только если левый операнд имеет значение **true**

Таблица истинности оператора И

a	b	a && b
true	true	**true**
true	false	**false**
false	true	**false**
false	false	**false**

Оператор ИЛИ

Условный оператор логического ИЛИ || (оператор короткого замыкания) вычисляет логическое ИЛИ для своих операндов. Результат операции **x** || **y** принимает значение **true**, если хотя бы один из операторов **x** или **y** имеет значение **true**. В противном случае результат будет **false**.

```
bool a = true;
bool b = false;
bool c = a || b;
```

Особенность: оператор || вычисляет правый операнд, только если левый операнд имеет значение **false**

Таблица истинности оператора ИЛИ

a	b	**a || b**
true	true	**true**
true	false	**true**
false	true	**true**
false	false	**false**

Приоритеты логических операторов

Логические операторы в порядке убывания приоритета:

!	Оператор отрицания
&&	Оператор И
||	Оператор ИЛИ

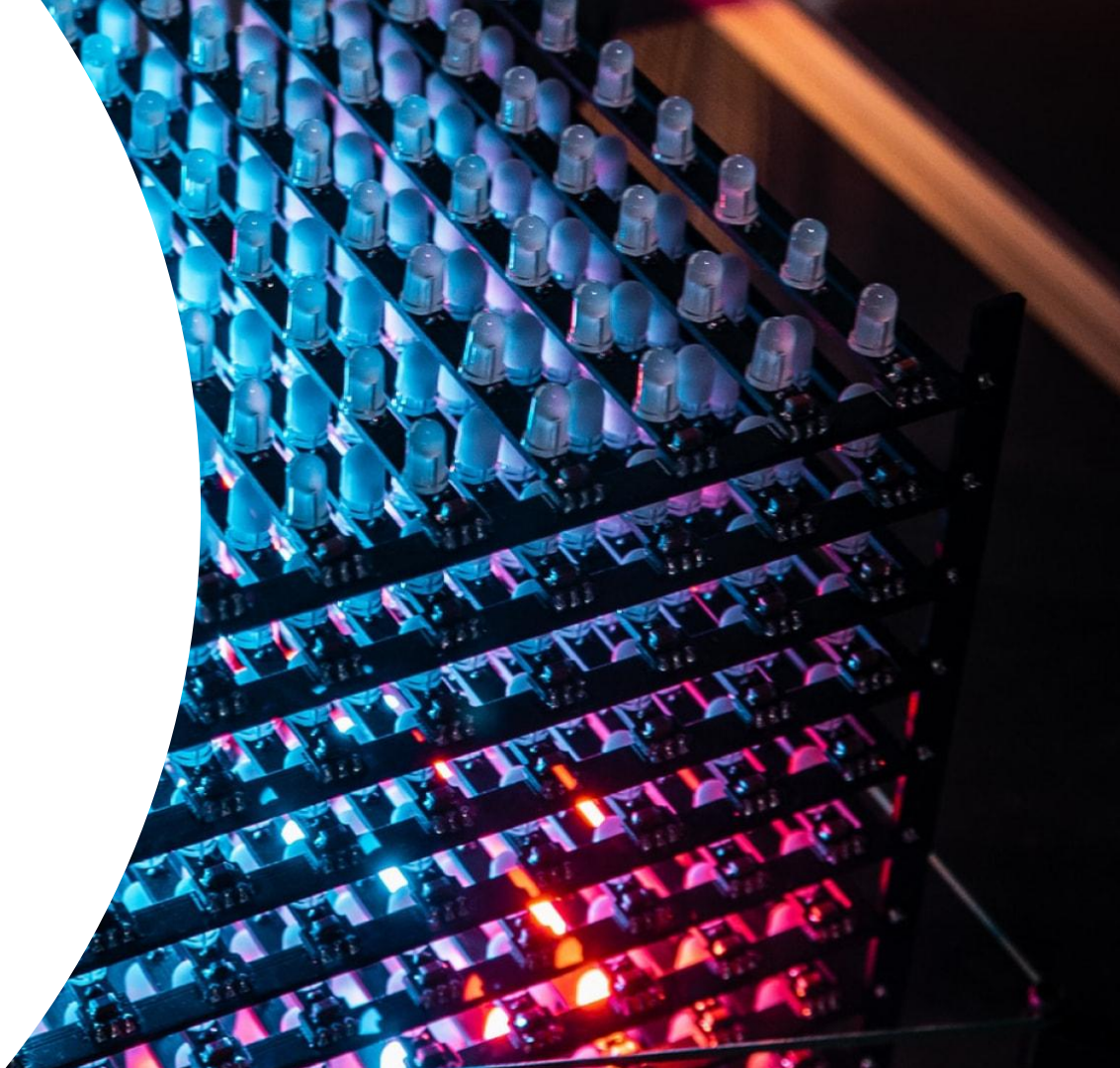
Порядок вычисления, определяемый приоритетом операторов, можно изменить с помощью скобок

Вложенные операторы

5

Разберём пример

вложенного if



Вложенный оператор if

Оператор **if** может иметь произвольное количество вложений.

Рассмотрим на примере поиска максимума из трёх чисел:

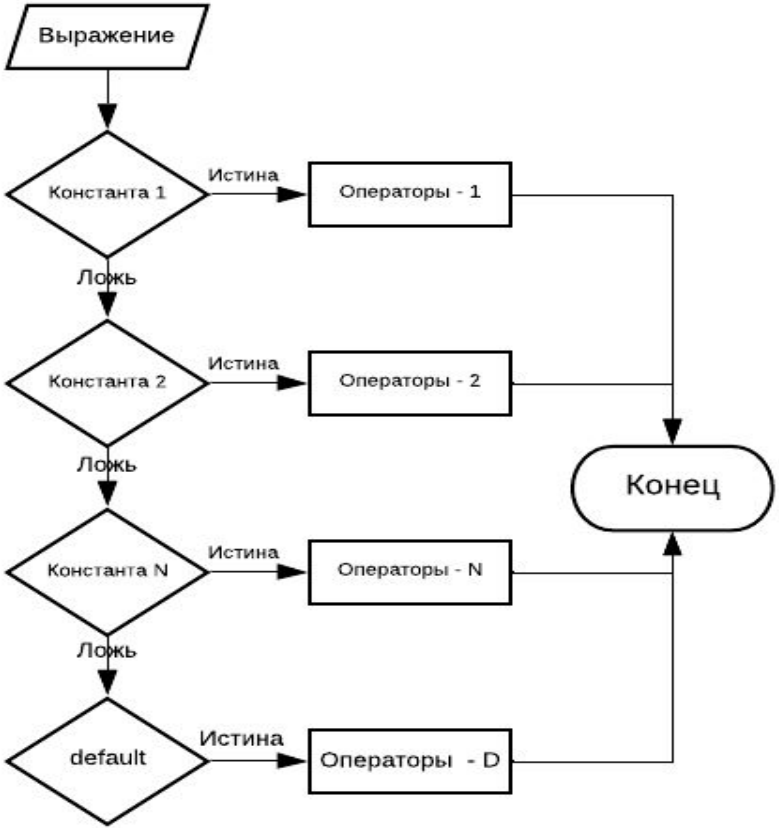
```
int a = 1, b = 2, c = 3, result;
if(a > b) {
    if(a > c) { result = a; }
    else      { result = c; }
} else {
    if(b > c) { result = b; }
    else      { result = c; }
}
std::cout << result << std::endl;
```

Ссылка на готовый код в Repl

Оператор switch

6

Оператор switch



Оператор switch последовательно проверяет результат выражения на соответствие одному из значений.

При первом совпадении с константой будут выполнены операторы, соответствующие совпадающему значению, после чего управление будет передано дальше.

Если ни одна из констант не совпала со значением выражения, то будут выполнены операторы из блока default.

Оператор switch

Оператор switch выбирает один из блоков для выполнения.

Оператор последовательно сравнивает операнд с вариантами значений и при совпадении значения передает управление оператору блока следующего за меткой case с совпавшим значением.

Если нет ни одного подходящего блока case, передает управление оператору блока default.

```
int num = 3;
switch(num) {
        case 1:
                std::cout << “One” << std::endl;
                break;
        case 2:
                std::cout << “Two” << std::endl;
                break;
        case 3:
                std::cout << “Three” << std::endl;
                break;
        default:
                std::cout << “Other number” << std::endl;
        break;
}
```

Ссылка на готовый код в Repl

Правила switch

В case обязательно использовать интегральный тип данных (т.е. char, short, int, long, long long или enum). Неинтегральный тип или тип с плавающей точкой использоваться не могут.

Константы не могут повторяться, то есть в case всегда определены исключающие друг друга значения.

Из блока case следует выйти используя break или return. В противном случае начнёт выполняться следующий блок.

```
int num = 3;
switch(num) {
        case 1:
                std::cout << “One” << std::endl;
                break;
        case 2:
                std::cout << “Two” << std::endl;
                break;
        case 3:
                std::cout << “Three” << std::endl;
                break;
        default:
                std::cout << “Other number” << std::endl;
        break;
}
```

Несколько меток switch

Каждый раздел операторов `switch` может содержать одну или несколько меток `case`.

```
int num = 3;
switch(num) {
        case 1:
        case 3:
                std::cout << “Odd” << std::endl;
                break;
        case 2:
        case 4:
                std::cout << “Even” << std::endl;
                break;
        default:
                std::cout << “Parity error” << std::endl;
                break;
}
```

Сравнение switch и if-else

```
int num = 3;
switch(num) {
        case 1:
                std::cout << “One” << std::endl;
                break;
        case 2:
                std::cout << “Two” << std::endl;
                break;
        case 3:
                std::cout << “Three” << std::endl;
                break;
        default:
                std::cout << “Unknown” << std::endl;
                break;
}
```

```
int num = 3;
if (num == 1) {
        std::cout << “One” << std::endl;
} else if (num == 2) {
        std::cout << “Two” << std::endl;
} else if (num == 3) {
        std::cout << “Three” << std::endl;
} else {
        std::cout << “Unknown” << std::endl;
}
```

Вложенный оператор switch

Оператор switch может иметь произвольное количество вложений.

В некоторых случаях подобный подход позволяет увеличить производительность при поиске последовательности символов в строке.

[Ссылка на готовый код в Repl]

```
std::string extension = "jpg";// Jpg, jpeg, JPEG
bool result = false;
switch (extension[0]) {
    case 'j':
    case 'J':
        switch (extension[1]) {
            case 'p':
            case 'P':
                switch (extension[2]) {
                    case 'g':
                    case 'G':
                        result = true;
                        break;
                    case 'e':
                    case 'E':
                        switch (extension[3]) {
                            case 'g':
                            case 'G':
                                result = true;
                                break;
                            } break;
                } break;
        } break;
}
std::cout << result << std::endl;
```

Что будет выведено на экран?

```
int a = 6;
if(a > -10 && a < 10) {
    switch(a % 2) {
        case 0:
            std::cout << (a < 0 ? "negative " : "") <<  "even" << std::endl;
            break;
        default:
            std::cout << (a < 0 ? "negative " : "") <<  "odd" << std::endl;
            break;
    }
}
```

Ссылка на готовый код в Repl

Тернарный оператор

7

Тернарный оператор

Оператор **?:** выполняет Выражение-1 если условие Истинно, иначе выполняет Выражение-2.

В общем виде записывается следующим образом:

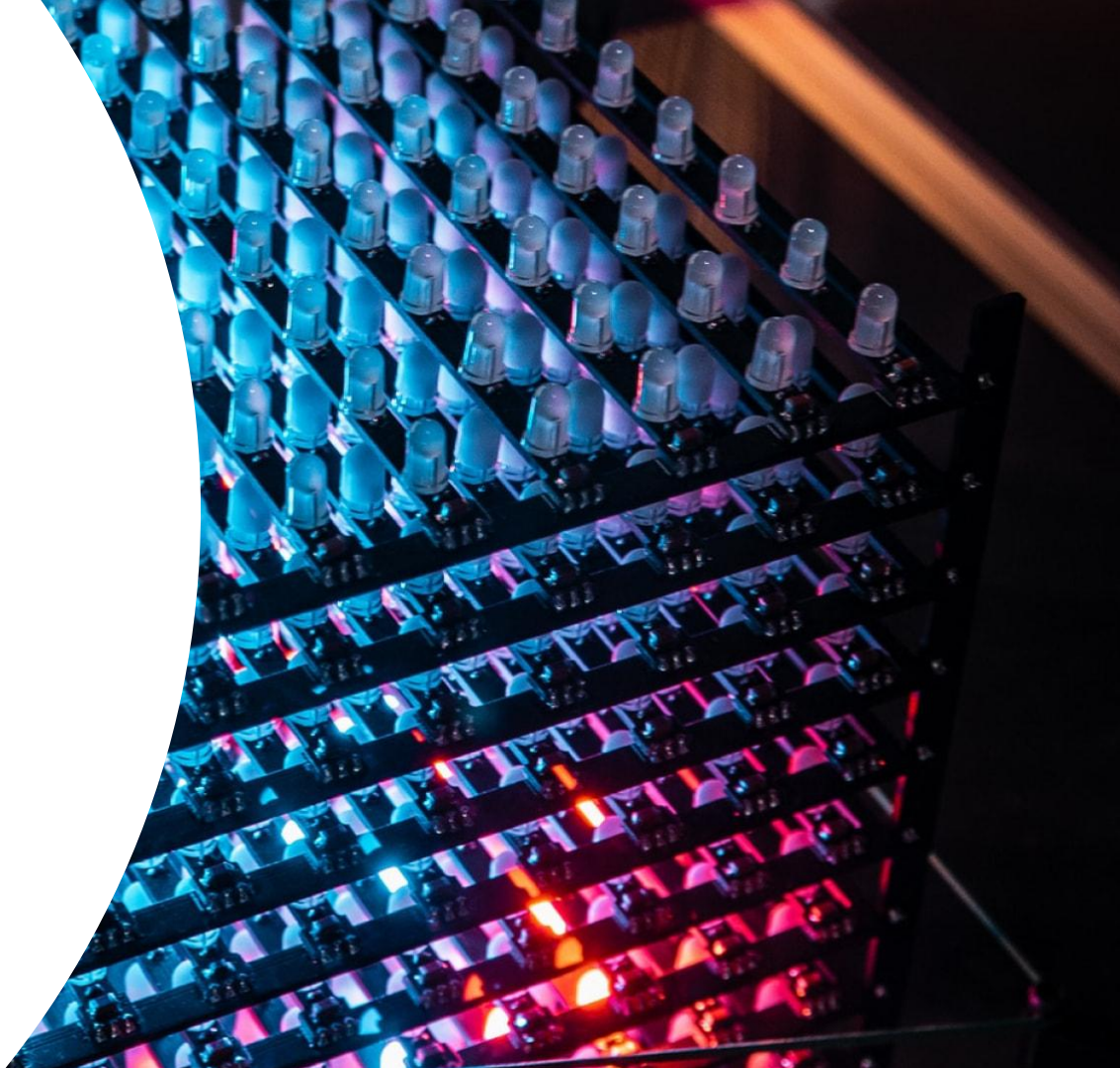
```
Условие ? Выражение-1 : Выражение-2
```

Подсказка: как запомнить правила вычисления тернарного оператора:

```
Это условие истинно ? да : нет
```

Разберём пример

тернарного оператора



Тернарный оператор

Тернарный оператор **?:** позволяет лаконично записывать условный код:

```
std::string result = a >= 0 ? "positive" : "negative";
std::cout << result << std::endl;
```

Тот же код, написанный с использованием оператора **if-else:**

```
int a = 0;
std::string result;
if(a >= 0) {
    result = "positive";
} else {
    result = "negative";
}
std::cout << result << std::endl;
```

Ссылка на готовый код в Repl

Вложенный тернарный оператор

Оператор имеет правую ассоциативность, то есть выражение:

```
a ? b : c ? d : e
```

Вычисляется как:

```
a ? b : (c ? d : e)
```

Вложенный тернарный оператор

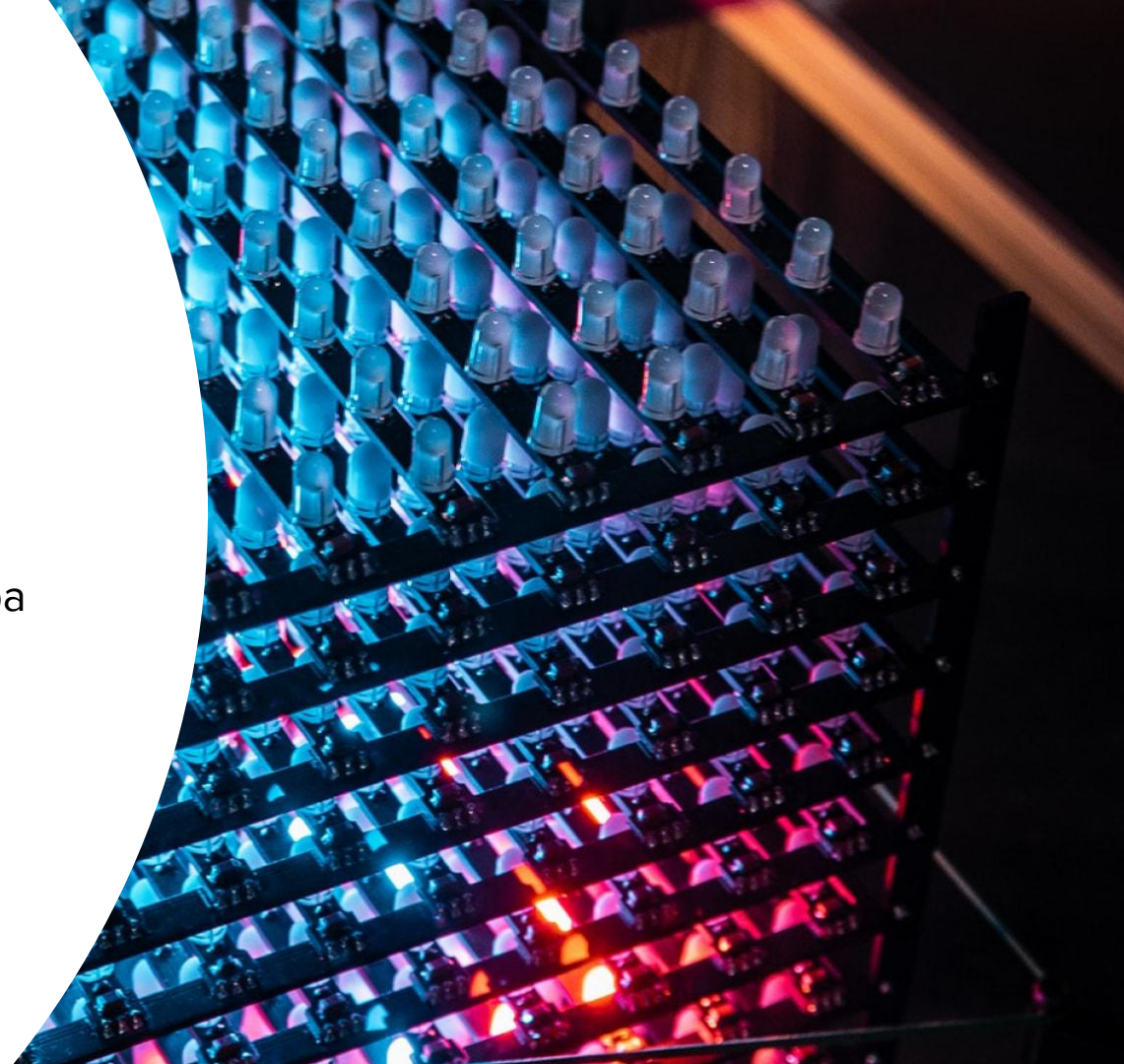
Тернарный оператор может иметь произвольное количество вложений.

Поиск максимума из трёх чисел:

```
int a = 1, b = 2, c = 3;
int result = a > b
                ? a > c
                    ? a : c
                : b > c
                    ? b : c;
std::cout << result << std::endl;
```

Разберём пример

вложенного тернарного оператора



Вложенный тернарный оператор

Пример использования вложенного тернарного оператора, который хорошо читается из-за нестандартного форматирования:

```
int a = 3;
std::string result = a == 1 ? "one"
                   : a == 2 ? "two"
                   : a == 3 ? "three"
                   : "Unknown";
std::cout << result << std::endl;
```

Ссылка на готовый код в Repl

Итоги

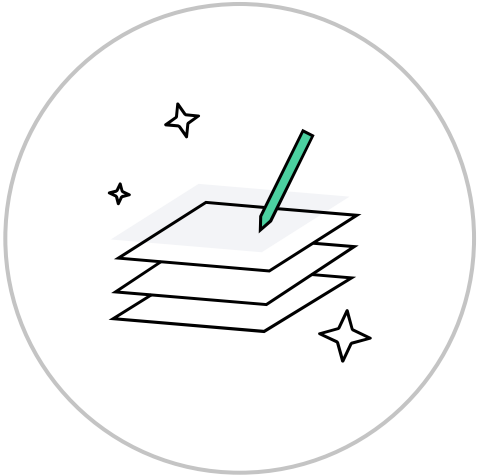
Сегодня мы

1. Разобрали логические операторы ! && ||
2. Разобрали операторы сравнения == != < > <= >=
3. Научились комбинировать операторы в условные выражения
4. Узнали операторы if-else-if-else и тернарный оператор ?:
5. Разобрали примеры вложенных операторов

Домашнее задание

Давайте посмотрим ваше домашнее задание:

1. Вопросы по домашней работе задавайте в чате группы
2. Задачи можно сдавать по частям
3. Зачёт по домашней работе ставят после того, как приняты все задачи



Задавайте вопросы и пишите отзыв о лекции

Михаил Марков
C++ - разработчик