Условная операция и условный оператор

Проблема

Задача: на вход поступают целые числа x и y, найти результат целочисленного деления x на y, если $y \neq 0$, а иначе вывести "Error".

Арифметикой здесь не обойтись...

Проблема

Задача: на вход поступают целые числа x и y, найти результат целочисленного деления x на y, если $y \neq 0$, а иначе вывести "Error".

Воспользуемся свойствами "короткой" логики:

```
int main() {
  int x, y;
  std::cin >> x >> y;
  y > 0 && std::cout << x / y << '\n';
  y == 0 && std::cout << "Error\n";
  return 0;
}
```

Проблема

Ясно, что текущих инструментов недостаточно для лаконичного решения даже самых простых задач.

Хочется научиться вычислять выражения при выполнении некоторых условий.

Условная операция

В языке C++ существует единственная тернарная операция (принимающая 3 аргумента) - *условная операция* (`?:`).

Она имеет вид `<bool-expr> ? <expr1> : <expr2>`, где

1. `<bool-expr>` - выражение со значением конвертируемым в `bool`,
2. `<expr1>`, `<expr2>` - выражения с "совместимыми" возвращаемыми значениями (что это значит обсудим позже).

Если `<bool-expr>` возвращает `true` (или то, что приводится к `true`), то выполняется `<expr1>`, иначе - `<expr2>`.

```
true ? 1 : 0;
x > 0 ? x * x : x * 2;
x > y && x * y ? x / y : 0;
```

Условная операция

Попробуем решить задачу с помощью условной операции:

На вход поступают целые числа x и y, найти результат целочисленного деления x на y, если $y \neq 0$, а иначе вывести "Error".

```
int main() {
  int x;
  int y;
  std::cin >> x >> y;
  std::cout << (y != 0 ? x / y : "Error") << '\n';
  return 0;
}
```

```
error: operands to ‘?:’ have different types ‘int’ and ‘const char*’
```

Условная операция: возвращаемый тип

```
int main() {
  int x;
  int y;
  std::cout << (y != 0 ? x / y : "Error") << '\n';
  return 0;
}
```

```
error: operands to ‘?:’ have different types ‘int’ and ‘const char*’
```

Проблема заключается в том, что типы `int` и "строка" несовместимы, то есть не существует неявного преобразования одного в другое.

Условная операция: возвращаемый тип

`<bool-expr> ? <expr1> : <expr2>`

Коротко: результатом условной операции является наибольший тип, способный вместить результат `<expr1>` и `<expr2>`.

Результирующий тип, разумеется, не зависит от истинности `<bool-expr>`, так как разрешение типов происходит на этапе компиляции, а не во время выполнения:

```
x > 0 ? 1 : 1.0;      // возвращаемый тип - double
true ? 0 : 5ll;       // возвращаемый тип - long long
false ? "str" : 0;    // ошибка - "строка" и int несовместимы
```

Условная операция

Но можно решить так:

```
int main() {
  int x;
  int y;
  std::cin >> x >> y;
  y != 0 ? std::cout << x / y << '\n' : std::cout << "Error\n";
  return 0;
}
```

Условный оператор

Условный оператор

Условный оператор в C++ имеет вид:

`if ([init] <condition>) <statement-true> [else <statement-false>]`,

где:

- `condition` - либо выражение, либо объявление переменной с инициализатором. В любом случае значение должно быть приводимо к `bool`.
- `statement-true` - оператор, который выполняется, если `condition` - `true`
- `statement-false` - оператор, который выполняется, если `condition` - `false`
- `init` - либо выражение, либо объявление. Область действия объявленной сущности совпадает с областью условного оператора.

Условный оператор: примеры

```
if (x > 0) std::cout << x << '\n';
```

```
if (int x = 0) {  // false
  int y;
  std::cin >> y;
  std::cout << y / x << '\n';
}
// здесь x и y уже не действуют
```

```
if (x != 0 && y / x > 5) return y / x - 5;
```

```
if (x);
```

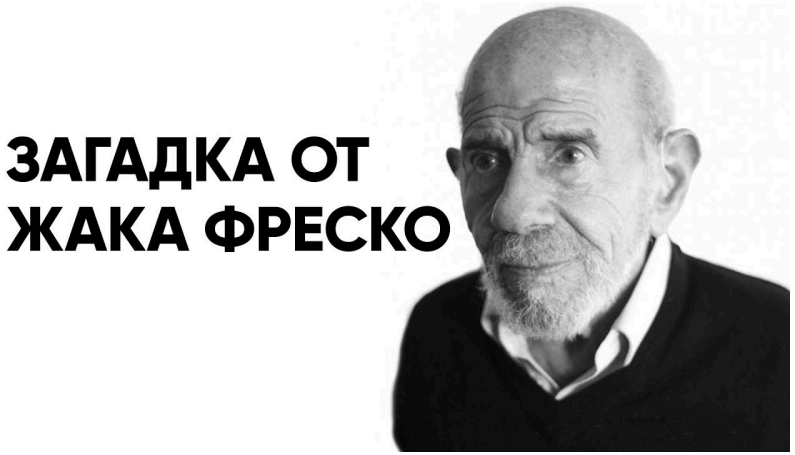
Условный оператор: примеры (else)

```
if (x != 0) std::cout << y / x << '\n';
else std::cout << "Error\n";
```

```
if (x >= 0 && y >= 0) std::cout << "Positives\n";
else {
  int z = x * y;
  std::cout << (z > 0 ? "Negatives\n" : "Different\n");
}
```

```
if (x >= y) {
  if (x >= z) {
    std::cout << x << '\n';
  } else {
    std::cout << z << '\n';
  }
} else if (y >= z) std::cout << y << '\n';
else std::cout << z << '\n';
```

Условный оператор: примеры (else)



Что выведет программа?

```
if (true)
  if (false) std::cout << "A\n";
else std::cout << "B\n";
```

Условный оператор: примеры (else)

Что выведет программа?

```
if (true)
  if (false) std::cout << "A\n";
else std::cout << "B\n";
```

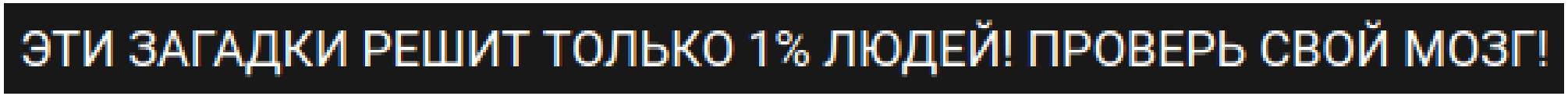
```
B
```

`else` относится к ближайшему неспаренному `if`!

Условный оператор: примеры (else)

`else` относится к ближайшему неспаренному `if`!

```
if ... if ... else ... else ... if ... if ... else ...
```



Условный оператор: примеры (else)

`else` относится к ближайшему неспаренному `if`!

```
if ... if ... else ... if ... else ... else ...
```

Для экономии времени и нервов используйте фигурные скобки:

```
if (...) {
  if (...) {
    ...
  } else {
    ...
  }
  if (...) {
    ...
  } else {
    ...
  }
} else {
  ...
}
```

if с инициализацией

В C++17 появилась возможность выполнять дополнительные действия при входе в условный оператор:

```
if (int x = y + z; x != 5) std::cout << x << '\n';
```

```
if (int x; x = 0) {  // false (в init инициализация не обязательна)
  int y;
  std::cin >> y;
  std::cout << y / x << '\n';
}
// здесь x и y уже не действуют
```

```
// В init может стоять любое выражение (не только объявление)
if (std::cin >> x; x != 0 && y / x > 5) return y / x - 5;
```

Оператор `switch`

Оператор `switch`

`switch ([init] <condition>) statement`

- `init` - либо выражение, либо объявление. Область действия объявленной сущности совпадает с областью оператора.
- `condition` - выражение или определение, имеющее целый или перечислимый тип.
- `statement` - произвольный оператор.

Как правило, в качестве `statement` выступает составной оператор, в котором присутствуют метки `case`, `default` и операторы `break`:

```
switch (x) {
  case 0: std::cout << "Zero\n"; break;
  case 1: std::cout << "One\n"; break;
  default: std::cout << "Other\n";
}
```

Оператор `switch`

```
case <const-expr>: <statements>
default: <statements>
```

- `const-expr` - выражение с целым или перечислимым значением, которое вычислимо **на этапе компиляции**.
- `statements` - последовательность операторов.

После вычисления `switch` условия ищется соответствующая метка и управление передается ее первому оператору.
Если нужного значения найдено не было, то осуществляется переход к `default`.

Важно! После перехода к метке выполняются **все** операторы, расположенные после нее (даже те, которые лежат под другими метками).

Чтобы завершить выполнение операторов необходимо написать оператор `break;`

Оператор `switch`: примеры

```
int x = ...;
switch (x) {
  case 0: std::cout << "Zero\n";
          break;
  case 1: std::cout << "One\n";  // после "One" выведется "Default"
  default: std::cout << "Default\n";
}
```

```
int x = 0;
const int y = 1;
switch (x * y) {
  case 2 * 2: ...  // Ok
  case y: ...      // Ok
  case x * 2: ...  // CE (не вычисляется на этапе компиляции)
}
```

Оператор `switch`: примеры

```
switch (int x = ...) {
  default: std::cout << "Default\n"; break;
  case 0: std::cout << "Zero\n"; break;
  case 1: std::cout << "One\n"; break;
}
```

```
switch (std::cin >> x; x * x) {
  case 0:
  case 1: std::cout << "x * x <= 1\n"; break;
  default: std::cout << "x * x >= 4\n";
}
```

Напоминание: в качестве условия может стоять только целое число:

```
float x = ...;
switch (x) {  // CE
  ...
}
```

Атрибут `[[fallthrough]]`

Зачастую, отсутствие `break` - скорее ошибка разработчика, нежели желаемое поведение. Существует не так много ситуаций, когда мы хотим выполнения сразу нескольких подряд идущих веток.

Поэтому компиляторы заботливо выдают предупреждения (ошибку при флаге `-Werror`), если встречают `case` без `break`:

```
main.cpp:10:26: error: this statement may fall through [-Werror=implicit-fallthrough=]
```

Атрибут `[[fallthrough]]`

Зачастую, отсутствие `break` - скорее ошибка разработчика, нежели желаемое поведение. Существует не так много ситуаций, когда мы хотим выполнения сразу нескольких подряд идущих веток.

Поэтому компиляторы заботливо выдают предупреждения (ошибку при флаге `-Werror`), если встречают `case` без `break`:

```
main.cpp:10:26: error: this statement may fall through [-Werror=implicit-fallthrough=]
```

Атрибут `[[fallthrough]]`

```
main.cpp:10:26: error: this statement may fall through [-Werror=implicit-fallthrough=]
```

Чтобы успокоить компилятор и сказать, что мы так и хотели, можно дописать *атрибут* `[fallthrough]`

```
switch (...) {
  case 0: ...
          [[fallthrough]];
  case 1: ...
          break;
  ...
}
```

Теперь компилятор будет уверен, что вы знаете, что делаете в `case 0`.

Резюме

- Условная операция позволяет выполнять короткие вычисления с условиями
- Типы выражений условной операции должны быть совместимы
- Оператор *if* более универсальный способ работы с ветвлениями.