Занятие 2.
Условный оператор

Сегодня мы изучим условный оператор. Он соответствует естественным понятиям «если» и «иначе».

Что такое условный оператор? Это очень просто: если выполнено условие, то нужно выполнять одно действие, а иначе — другое. По-английски «если» звучит как “if”, а «иначе» как “else”. Пусть перед нами стоит задача вывести модуль числа, не пользуясь функцией abs, с помощью конструкции if-else. В сегодняшних примерах мы будем писать не программы полностью, а только те команды, которые непосредственно выполняются — остальное вы можете дописать сами. Итак, решение задачи:

```
int a;
cin >> a;
if (a > 0) {
    cout << a;
} else {
    cout << -a;
}
```

Что можно понять из этой программы? Первые две строки мы уже хорошо понимаем — это создание переменной и её считывание. В следующей строке записано «если a больше нуля». Если это условие выполнено, то будут выполняться действия, заключённые в фигурные скобки после if. Если же условие не выполнено, то будут выполняться команды, заключённые в фигурные скобки после слова else (иначе). Таким образом, выполнение программы разветвляется, и будет выполнена только одна из двух ветвей. После окончания блока else выполнение программы будет продолжаться.

Можно попробовать решить задачу без использования команды else:

```
int a;
cin >> a;
if (a > 0) {
    cout << a;
}
if (a < 0) {
    cout << -a;
}
```

Кто скажет в каком случае эта программа будет работать неправильно?
Если ввести число 0. Не выполнится ни один из условных операторов.

Рассмотрим подробнее, как работает условный оператор. Он должен выглядеть как:

if (<логическое выражение>) {
    команды;
}

Сначала пишется слово if, затем в круглых скобках пишется логическое выражение, а затем в фигурных скобках перечисляются команды, которые должны быть выполнены.

Что такое логическое выражение? Это выражение, которое может принимать значения «истина» (“true”) и «ложь» (“false”).

Логическое выражение имеет вид:
<арифметическое выражение> <сравнение> <арифметическое выражение>

Например, если у нас есть переменные x и y с какими-то значениями, то логическое выражение x + y < 3 * y в качестве первого арифметического выражения имеет x + y, в качестве знака сравнения < (меньше), а второе арифметическое выражение в нём 3 * y.
В логических выражениях допустимы следующие знаки сравнений:

==	равно
!=	не равно
<	меньше
>	больше
<=	меньше или равно
>=	больше или равно

Очень часто при использовании сравнения на равенство (==) возникает ошибка: вместо двух значков «равно» можно написать один, и это будет корректная программа, но делать она будет совсем не то, чего мы от нее ждём. Например, попробуем написать программу, которая считывает два числа и выводит Yes, если они одинаковые, и No в противном случае:

```
int a, b;
cin >> a >> b;
if (a = b) {
    cout << "Yes";
} else {
    cout << "No";
}
```

Эта программа будет выводить Yes, даже если числа различны (и b не равно нулю). Чтобы отловить эту ошибку, нужно внимательно читать предупреждения, которые выдаёт компилятор.

Вложенные инструкции

Внутри блока инструкций могут находиться другие ветвления программы. Посмотрим сразу на примере. По заданному количеству глаз и ног нужно научиться отличать кошку, паука, морского гребешка и жучка. У морского гребешка бывает более сотни глаз, а у пауков их восемь. Также у пауков восемь ног, а у морского гребешка их нет совсем. У кошки четыре ноги (и длинный хвост), а у жучка — шесть ног, но глаз у обоих по два. Решение:

```
int eyes, legs;
cin >> eyes >> legs;
if (eyes >= 8) {
      if (legs == 8) {
            cout << "spider";
      } else {
            cout << "scallop";
      }
} else {
      if (legs == 6) {
            cout << "bug";
      } else {
            cout << "cat";
      }
}
```

Если вложенных условных операторов несколько, то, к какому из них относится else, компилятор понимает по количеству закрывшихся фигурных скобок. Обратите внимание, как красиво выглядит пример: в нем правильно расставлены отступы.

Напомню правило: если открылась фигурная скобка, то со следующей строки добавляется один отступ (если вы его случайно удалили, то можно поставить новый с помощью кнопки tab), а в строке, где есть закрывающаяся фигурная скобка, один отступ убирается.

Программы с правильно расставленными отступами приятнее читать и понимать, а программы с неправильно расставленными отступами проверяться не будут.

Ещё в нашей программе очень красиво названы переменные. Нужно стараться называть переменные так, чтобы был понятен их смысл. Лучше всего использовать английские слова (не транслит!), их стандартные аббревиатуры и сокращения. Если переменная, например, означает «количество слов», то её подойдёт название “word_cnt”. Здесь cnt — сокращение от слова “count” (количество), а word — «слово». Отдельные слова следует разделять символом подчёркивания. Длина названия переменной не должна превышать 10–12 символов. Если вы знаете английский язык недостаточно хорошо, вам поможет сервис Яндекс.Перевод или Google Translate.

Логические операторы

Иногда возникает необходимость проверить более сложное условие, чем просто сравнение между собой двух чисел. Для этого можно воспользоваться логическими операторами. Посмотрим на примере. Задача похожа на предыдущую: программе на вход дается количество глаз и ног у животного, и она должна пугаться, если это паук (с 8 глазами и 8 ногами). Решение:

```
int eyes, legs;
cin >> eyes >> legs;
if (eyes == 8 && legs == 8) {
      cout << "AAAAA!!!";
}
```

Логический оператор && — это «и». То есть условие будет верным, если и левое, и правое логические выражения истинны одновременно.

Также существует логическая связка «или», которая обозначается символами «||», и унарная операция «отрицание» — её знак «!» ставится перед выражением.

В логических выражениях также существует порядок действий. Сначала выполняются отрицания, затем все арифметические операции, затем все «и», только потом все «или». На порядок действий также можно влиять с помощью скобок.

Рассмотрим ещё один пример. Как известно, в России есть Новый год и старый Новый год. Новый год отмечается по григорианскому календарю, а старый Новый год — по юлианскому. В юлианском календаре високосность года проверяется очень легко: если номер года делится на 4, то он является високосным. Напишем программу, которая определяет количество дней в году по его номеру:

```
int year;
cin >> year;
if (year % 4 == 0) {
      cout << 366;
} else {
      cout << 365;
}
```

Чтобы проверить любое число на делимость, необходимо посчитать остаток от деления и сравнить его с нулём. Теперь решим более сложную задачу: посчитать количество дней в году по григорианскому календарю. Правила определения високосности года в григорианском календаре такие: год является високосным, если его номер делится на 4, но не делится на 100 или делится на 400.

В этой задаче логическое выражение получается очень длинным, и его можно разбить на части, сохранив некоторые промежуточные значения в переменных. Для хранения результата логического выражения используется переменная типа bool. Переменные такого типа могут хранить лишь два возможных значения: «истина» и «ложь». Они обозначаются, соответственно, true и false.

Здесь в переменной mod400 хранится «истина», если остаток от деления номера года был равен нулю, и «ложь» в противном случае. Если логическое выражение получилось слишком длинным и не помещается на экран, то можно разбить его на несколько строк — программа продолжит работать нормально.

Если вы боитесь перепутать приоритеты логических операций, то можно для уверенности поставить скобки. Наше выражение можно записать так:

((year % 4 == 0 && year % 100 != 0) || mod400).

Конструкция «иначе-если»

В некоторых ситуациях нужно выбрать больше чем из двух вариантов. Пусть нам звонит кто-то из друзей и предлагает заняться чем-нибудь. Числом 1 обозначим друга Ваню, который зовет нас играть в футбол, числом 2 — друга Сашу, который зовет нас вместе учить C++. А если нам звонит кто-нибудь другой, то мы сидим дома и смотрим сериал. Нужно по введенному числу сказать, чем мы будем заниматься. Решение этой задачи, использующее два оператора if, выглядит так:

```
int who_call;
cin >> who_call;
if (who_call == 1) {
      cout << "Football!!";
} else if (who_call == 2) {
      cout << "C++!!!!!11";
} else {
      cout << "Show";
}
```

Конструкция else if имеет смысл «иначе-если». Дойти до сравнения цифры с двойкой можно только в случае, если уже произошло сравнение с единицей и мы пошли по ветви «иначе». Последний else, таким образом, выполнится только после того, как число сравнилось и с единицей, и с двойкой и ни одно из этих сравнений не было верным.

Благодаря таким конструкциям можно обработать большое количество различных вариантов поведения программы.