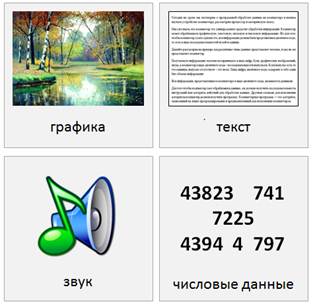
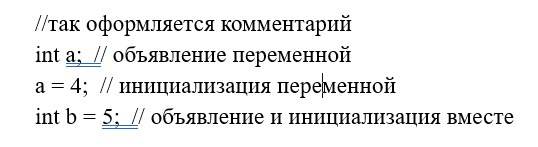
**«Что такое переменная?»**

Любая компьютерная программа производит какие-то математические или логические операции и отображает результат своей работы на экране. Многие программы используют графические и звуковые эффекты. Современные игры почти все работают с объемными 3D-эффектами. Все это, вычисления, эффекты, графика - результат вычислений, который появляется благодаря размещению большого объема чисел в ячейках памяти компьютера и работе с ними. Задача программиста — написать программу, а это значит заполнить память компьютера нужными данными, произвести с ними необходимые действия, получить результат и вывести его пользователю в нужном виде. Соответственно, для того чтобы написать полезную программу, нужно знать, как организовать ячейки памяти для хранения данных, как работать с этими данными и, в конечном итоге, вывести эти данные на экран.



Для того, чтобы записать что-то в память компьютера нужно создать так называемую переменную. Переменная — это «ячейка» оперативной памяти компьютера, в которой может храниться какая-либо информация. И, конечно, логично, что если мы хотим поработать с какой-то ячейкой памяти, то нужно дать ей имя.

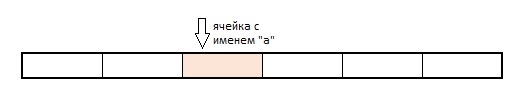
Самой простой и независимой частью программы на любом языке программирования является инструкция или стейтмент. Это такой аналог предложения в русском языке. Фраза, которая несет в себе смысловую нагрузку. И как в русском языке предложение завершается точкой, в программировании стейтмент обязательно завершается точкой с запятой.



Простейшая необходимая инструкция в программе — это объявление и инициализация переменной.

Помимо стейтментов в языке программирования существуют комментарии, которые не несут в себе никакой смысловой нагрузки для компьютера, но очень помогают программисту ориентироваться в коде программы. Комментарии пишутся после двух знаков слеша «/».

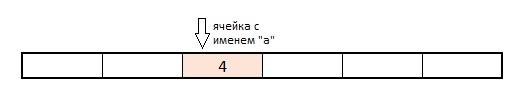
Первая инструкция — это объявление переменной. Объявить переменную – это значит сказать компьютеру, чтобы он выделил ячейку памяти и дал этой ячейке тип и имя, которое мы указали, в данном случае наша переменная имеет тип int, а имя - «a»



Что такое тип переменной. Мы привыкли, что в математике есть целые числа и дробные. В русском языке мы пользуемся буквами и составляем слова из них. Мы можем рассуждать логическими понятиями – «правда» или «ложь», «да» или «нет». Поэтому компьютеру мы тоже хотим дать такую возможность. Для этого и созданы различные типы, про которые мы подробно поговорим на следующих занятиях, а пока попробуем воспользоваться одним из таких типов данных – int, который означает целые числа.

Так вот с помощью строки объявления переменной мы сказали компьютеру, что хотим завести в его памяти ячейку, у которой будет имя «a». Какую конкретно ячейку выделит для нас компьютер и где она будет расположена в его памяти, нам не важно. Важно, что, компьютер будет знать, что в его памяти есть ячейка с именем «a» и когда мы захотим с ней поработать, он будет работать именно с этой конкретной ячейкой.

Второй стейтмент на слайде — это инициализация переменной. Инициализировать переменную – это значит положить в ячейку с ее именем какое-то значение.



Таким образом мы уже используем нашу ячейку, которую завели в предыдущем стейтменте. Компьютер знает, что у него в памяти есть ячейка с именем «a», поэтому он обращается именно к ней и помещает в нее значение «4».

Нужно запомнить, что переменная связывает между собой три понятия – это тип, имя и значение. Без связки этих трех свойств невозможно производить какие-то действия с переменными и памятью компьютера.

То есть первое, что вы должны сделать – это сказать компьютеру, что вам нужна ячейка, в которой вы будете хранить данные определенного типа, в нашем случае целое число, и дадите этой ячейке имя. А затем эту ячейку нужно заполнить какими-то данными.

В языке программирования Java есть правила, которые определяют какие имена переменным можно использовать, а какие нельзя.

1 Идентификатор не может быть ключевым словом. В языке С++ есть список слов, которые зарезервированы, то есть мы не можем использовать их для своих целей, поскольку каждое из этих ключевых слов уже определяет какую-то инструкцию.

Список ключевых слов представлен на следующем слайде:

2 Идентификатор может состоять только из букв (нижнего или верхнего регистра), цифр или символов подчёркивания. Это означает, что все другие символы и пробелы — запрещены.

3 Идентификатор должен начинаться с буквы (нижнего или верхнего регистра). Он не может начинаться с цифры или с другого какого-то символа.

6 И самое важное правило из всех - используйте в качестве идентификаторов только те имена, которые реально описывают то, чем является объект. Особенно это важно для больших программ. Но привыкать нужно сразу.

Если, например, вам нужно в программе посчитать количество яблок, то назовите переменную appleCount и вы всегда сможете по имени переменной понять, что хранится в ячейке с этим именем.

**Данные в памяти**

Не только обычному пользователю, но и современному программисту редко приходится обращаться к внутреннему представлению информации в компьютере. Конечно, для того и разработаны языки программирования высокого уровня, такие как Java или С++, чтобы не задумываться о внутреннем устройстве компьютерной памяти и вычислений. Но хороший программист должен обязательно понимать, как работает компьютер, как он хранит данные, работает с ними, чтобы избежать ошибок, которые могут возникнуть в процессе вычислений в его программе. Например, мы привыкли, что в математике сравнение чисел происходит с помощью знаков «>» и «<», причем неважно какие это числа - целые или дробные. А знаете ли вы, что в программировании нельзя сравнивать два числа вещественного типа с помощью обычных арифметических знаков сравнения. Почему? Попробуем разобраться…

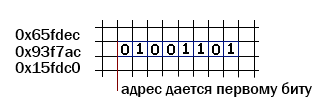


Слайд №1. Понятие бита и хранение данных в памяти компьютера

Компьютерная память состоит из огромного числа маленьких элементов. Каждый такой маленький элемент похож на выключатель, который может быть либо включен, либо выключен. Когда выключатель включен и поддерживает связь в цепочке, ему приписывают значение равное 1, а когда выключатель выключен и связь в цепочке разрывается, ему приписывается значение 0. Каждый такой элемент называют битом. Бит, как вы уже поняли, может принимать значение — 0 или 1. Именно поэтому компьютер использует двоичную систему счисления для хранения данных. Как это ни странно, но в современных компьютерах, биты не используются по отдельности, а объединяются по группам — байтам.

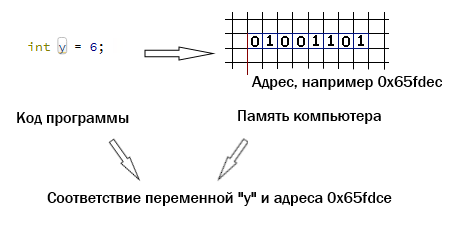
В русском языке отдельные буквы не имеют никакого смысла, но мы из букв составляем слова, а вот слово уже несет в себе какую-то смысловую нагрузку. Также и в компьютере, отдельные биты не используются сами по себе. Для удобства использования, биты собраны в кучки по 8 штук. Эта кучка из 8 битов называется байт. Вот и получается, что бит — это буква, а байт — это уже слово.

А теперь представим себе книгу… Каждое слово в книге мы можем пронумеровать и тогда будем точно знать под каким номером какое слово находится. Также и в компьютере — каждый байт имеет свою позицию, позиция байта в памяти компьютера называется адрес. Адреса позволяют найти и получить доступ к байту, который находится в определённом месте в памяти.



Слайд №2. Адреса в памяти компьютера

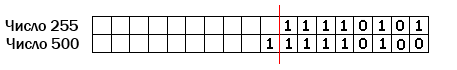
Мы не будем вдаваться в подробный разбор формирования адреса, остановимся на основных аспектах. На слайде представлены различные адреса. Важно то, что компьютер сам занимается распределением адресов для переменных, программист на это повлиять никак не может. Причем каждый раз запуская заново свою программу, вы увидите, что у одной и той же переменной может быть разный адрес. В этом состоит отличие от приведенной нами аналогии размещения слов в книге. Если в книге порядок слов зафиксирован раз и навсегда, то размещение значений переменных непостоянно и зависит только от компьютера.



Слайд №3. Связь переменной и адреса

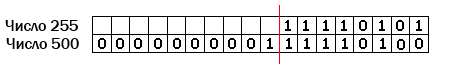
Но как вы уже поняли из предыдущих занятий, программист не работает напрямую с адресами, а использует переменные. Когда программист определяет переменную, например по имени «y», в коде программы, компьютер выделяет память для хранения значения этой переменной и создает связь «имя переменной» – «адрес в памяти». Всякий раз, когда компьютер встречает переменную «y» в коде программы, он понимает, что для того, чтобы получить значение этой переменной «y», ему нужно заглянуть в ячейку памяти c адресом «0x65fdce». Программисту не нужно беспокоиться о том, какой конкретно адрес памяти был выделен для его переменной «y». Он просто использует идентификатор переменной «y», а компьютер конвертирует это имя в соответствующий адрес памяти. Таким образом создается связь: «имя переменной» – «адрес переменной» – «значение переменной». Получается, что компьютеру не нужны переменные, они нужны программистам, потому что работать с адресами достаточно сложно, а с именами переменных намного понятнее и удобнее.

Мы уже знакомились с двоичной системой счисления и способом перевода числа из десятичной системы в двоичную и обратно. Каждый раз, когда программист задает значение переменной, компьютер переводит это значение в двоичную систему счисления.



Слайд №5. Представление чисел

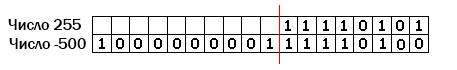
Согласно этому переводу число «255» в двоичной системе счисления имеет вид «11110101». Оно вполне вмещается в 8 бит одного байта. А число «500» в двоичной системе счисления имеет вид «1 11110100». И мы видим, что число «500» уже не помещается в один байт.



Слайд №6. Представление чисел

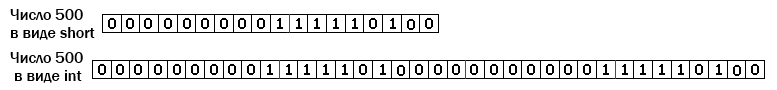
Поэтому для числа «500» компьютер выделит уже не один байт, а два и запишет его в таком виде «00000001 11110100».

А если нам нужно отрицательное число? Принято под знак числа отводить первый бит из задействованных байтов. Если число положительное, то есть знак «+», то бит будет равен 0. А если число отрицательное, то есть имеет знак «-», то бит будет равен 1. Значит число «-500» будет таким «10000001 11110100».



Слайд №7. Представление чисел

А если нам нужно записать очень большое число? Компьютер увеличит количество использованных байтов, например до 4 или 8. А как компьютер узнает, сколько байтов ему нужно использовать для записи числа? Эту информацию ему должен предоставить программист с помощью указания типа для переменной. Мы с вами знакомы уже с несколькими типами данных — это целочисленный тип данных, который называется int и вещественный тип данных, который называется double. Помимо того, что они отличаются тем, что хранят разные типы данных, они занимают разное количество памяти в компьютере. Тип int занимает 4 байта, а тип double 8 байт.

Слайд №8. Различие типов

Чем больше байт выделяется под используемый тип данных, тем большее число мы сможем сохранить в этих байтах. Если же мы попытаемся записать большее значение в переменную, чем позволяет ее тип, то получим неправильное значение.

Таким образом задача программиста правильно определять нужный тип данных для переменных и использовать именно его.

А зачем нужно столько типов данных? Посмотрим на наш слайд №8. Как вы думаете с какой из переменных компьютеру будет легче работать? Конечно, с переменной типа short. Для этого и существует несколько различных типов данных, чтобы там, где нужно облегчать работу компьютеру, а значит ускорять работу программы и использовать меньше компьютерной памяти.

**Типы Данных**

Одной из основных особенностей Java является то, что данный язык является строго типизированным. А это значит, что каждая переменная имеет определенный тип и данный тип строго определен специальными зарезервированными ключевыми словами Java. Тип данных определяет диапазон значений, которые может хранить переменная или константа.

Переменные не что иное, как зарезервированные места памяти для хранения значений. Поэтому при создании переменной Вы резервируете некоторое пространство в памяти, а ОС, основываясь на типе данных, который присвоен переменной, выделяет требуемое количество памяти и решает, что может быть сохранено в зарезервированную память.

Существует два типа данных в Java:

\* простые или примитивные типы данных;

\* ссылочные типы данных (ссылка/объект).

Примитивные типы данных

Есть восемь типов данных, поддерживаемых Java. Все размеры примитивных типов строго фиксированы и не зависят от машинной архитектуры. Основные типы данных предопределены языком и названы по ключевому слову.

\* byte (целые числа, 1 байт)

\* short (целые числа, 2 байта)

\* int (целые числа, 4 байта)

\* long (целые числа, 8 байтов)

\* float (вещественные числа, 4 байта)

\* double (вещественные числа, 8 байтов)

\* char (символ Unicode, 2 байта)

\* boolean (значение истина/ложь, 1 байт)

Типы данных можно разбить на 4 группы:

\* Целые числа (byte, short, int, long)

\* Числа с плавающей точкой (float, double)

\* Символы (char)

\* Логические значения (boolean)

**Представление отрицательных целых чисел**

Числа в компьютере состоят из нулей и единиц. Возникает вопрос, где хранить знак числа?

Естественной идеей является отведение какого-нибудь разряда под знак. Логичнее всего взять старший разряд, самый левый.

При представлении со знаком самый старший (левый) разряд отводится под знак числа, остальные разряды — под само число. Если число положительное, то в знаковый разряд помещается 0, если число отрицательное — 1.

Таким образом, в двоичной системе счисления, используя прямой код, в восьмиразрядной ячейке (байте) можно записать семиразрядное число. Например:

00110101 // 53

10110101 // -53

Соответственно, раз мы один бит забрали под знак, то диапазон значений теперь получается другой от -128 до 127 включительно.

Такое представление чисел называется прямым кодом. В компьютере прямые коды используются для хранения положительных чисел в запоминающих устройствах, для выполнения операции с положительными числами. Но отрицательные целые числа не представляются в ЭВМ с помощью прямого кода, для их представления используется так называемый **дополнительный код**. Почему?

При сложении положительных чисел, все работает хорошо:

(3 + 2 = 5)

+ 00000011

00000010

= 00000101

Но аналогичные действия для чисел с разными знаками или двумя отрицательными дают неверный результат:

(3 + (–2) = –5)

+ 0000 0011

1000 0010

= 1000 0101

То есть для отрицательных и положительных чисел нельзя использовать один и тот же алгоритм. Если бы вычислительная машина работала с прямыми кодами положительных и отрицательных чисел, то при выполнении арифметических операций следовало бы выполнять ряд дополнительных действий. Например, при сложении нужно было бы проверять знаки обоих операндов и определять знак результата. Если знаки одинаковые, то вычисляется сумма операндов и ей присваивается тот же знак. Если знаки разные, то из большего по абсолютной величине числа вычитается меньшее и результату присваивается знак большего числа. То есть при таком представлении чисел (в виде только прямого кода) операция сложения реализуется через достаточно сложный алгоритм. Если же отрицательные числа представлять в виде дополнительного кода, то операция сложения, в том числе и разного знака, сводится к из поразрядному сложению.

Дополнительный код используется для упрощения выполнения арифметических операций. Как получить этот дополнительный код.

Дополнительный код числа, или дополнение до двойки (two’s complement), — это обратный код, к которому прибавлена единица.

Обратный код числа, или дополнение до единицы (one’s complement) - это прямой код, где во всех разрядах, кроме знакового, все нули заменяются на единицы, а единицы на нули.

В дополнительный код переводяться только отрицательные числа.

-2 = 1000 0010

1000 0010 - прямой код

1111 1101 - обратный код

+ 1111 1101 - обратный код

00000001

= 1111 1110 - дополнительный код

Теперь, если мы сложим 3 и -2 в обратном коде, то получим 1.

+ 0000 0011

1111 1110

= 0000 0001

Чтобы не выполнять преобразование каждый раз, в вычислительной технике прямой код используется для представления положительных чисел, а дополнительный – для представления отрицательных.

**Представление вещественных чисел.**

В компьютерной технике вещественными называются числа, имеющие дробную часть.

Вопрос, как хранить целую, а как дробную часть?

На помощь приходит нормализованная (экспоненциальная) запись действительных чисел.

Любое вещественное число A может быть записано в экспоненциальной форме: A = ±m⋅qp , где:

m — мантисса числа;

q — основание системы счисления;

р — порядок числа.

Пример:

1284.7503 — математическая запись числа

1.2847503 \* 103 — нормализированная форма числа

111000111.0110 — математическая запись числа

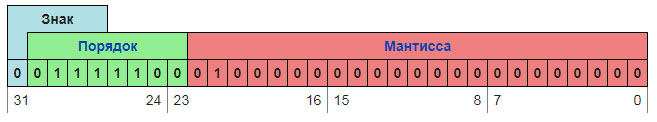
1.110001110110 \* 28 — нормализированная форма числа

0.001110001110 — математическая запись числа

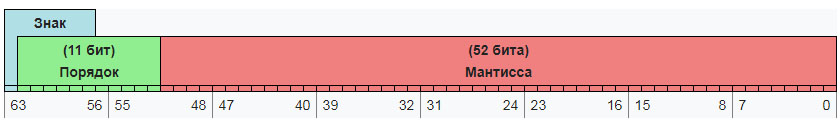
1.110001110 \* 2-3 — нормализированная форма числа

Таким образом у нас число — это мантисса, а степень — порядок.

Так как для числа типа float отведено 4 байта — 32 бита, то имеем такой расклад: 1 бит — храним знак мантиссы, 8 бит — значение порядка, 23 бита — значение мантиссы.



Так как для числа типа double отведено 8 байт — 64 бита, то имеем такой расклад: 1 бит — храним знак мантиссы, 11 бит — значение порядка, 52 бита — значение мантиссы.



Для того, чтобы не хранить знак порядка, был придуман так называемый смещённый порядок, который позволяет нам избавится от отрицательных степеней. Для этого мы берем половину диапазона порядка и прибавляем его к наименьшему значению диапазона.

Например, для числа float количество бит под знак равно 8. Диапазон чисел, которые мы можем поместить в 8 бит = (-127, 128), то есть есть и отрицательные числа и положительные, а чтобы нам получить только положительные числа, мы прибавляем смещение 127, таким образом смещаем диапазон до (0 — 255).

Пример:  
Если для числа float истинный порядок равен -5, тогда смещённый порядок:

127 — 5 = 122.

**Откуда берутся «ошибки»?**

**Пример1:**

Рассмотрим на примере десятичного числа 6.125. Это дробное число в десятичной системе счисления представляется так:



0.125 \* 2 = 0.25  
0.25 \* 2 = 0.5  
0.5 \* 2 = 1

6.125 = 0110.001

Теперь возьмем, например, число



Попробуем перевести дробную часть 0.8 в двоичной системе:

0.8 \* 2 = 1.6  
0.6 \* 2 = 1.2  
0.2 \* 2 = 0.4

0.4 \* 2 = 0.8

0.8 \* 2 = 1.6 ….

Продолжать можно еще довольно долго, но уже сейчас видно, что 0.8 в десятичной системе это 0.11001100… 0.1100 в периоде

Поэтому перевод дробного числа из одной системы счисления в другую чаще всего дает погрешность. Погрешность эта зависит от того, сколько разрядов мы используем для записи дробной части переведенного числа. Возьмем пример с числом 0.8 и используем для записи его двоичного представления шесть разрядов после запятой — 0.110011.

Полученное число вовсе не 0.8, а 0.796875, разница при этом составляет 0.003125. Это и есть наша погрешность перевода десятичного числа 0.8 в двоичный вид при использовании шести разрядов после запятой.

Теперь попробуем составить более сложные инструкции, чтобы связать между собой несколько переменных.

int a = 4;

int b = 2;

int c = a \* b + 10;

a = a + c;

С помощью арифметических операций можно изменять значения переменных в ячейках. Как только вы присваиваете переменной новое значение, старое значение затирается и в ячейку помещается это новое значение.

Таким образом, изначально в ячейке с именем «a» находится значение 4, а в ячейке с именем «b» находится значение 2. Чтобы заполнить значением ячейку с именем «с», компьютер сначала выполняет арифметические операции a \* b + 10, и полученное значение 18 помещает в ячейку с именем «с».

Задание обучающимся – объяснить четвертый стейтмент на слайде.

Строка static void Main(string[] args) определяет целый блок, который называется главная функция. Отправной точкой выполнения любой С# программы является функция, которая называется Main(). Без этой функции программа просто не запустится.

static void — обозначает тип функции, с ними мы познакомимся позже. А в круглых скобках задаются входные параметры функции, с ними мы тоже разберемся чуть позже. Самое главное, что нужно сейчас понять и усвоить, что функция с помощью фигурных скобок обозначает пространство своего кода. Внутри этих скобок пишется тело функции, то, что эта функция будет делать. Как только фигурная скобка закрывается, код уже не принадлежит этой функции. Есть разные типы кода — есть стейтменты выполнения, которые обязательно должны располагаться внутри какой-то функции, а есть стейтменты объявления, они могут располагаться вне функции. Разница в том, что стейтменты выполнения — это описания действий для процессора, как только мы хотим сказать ему - выдели память под переменную, запиши в нее значение, выполни какие-то арифметические действия — это стейтменты действий, они должны располагаться внутри функции.

Задание: Написать программу, которая из двух полей считывает два числа и выводит сумму чисел.