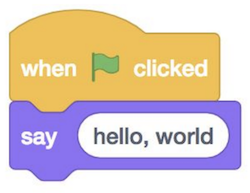
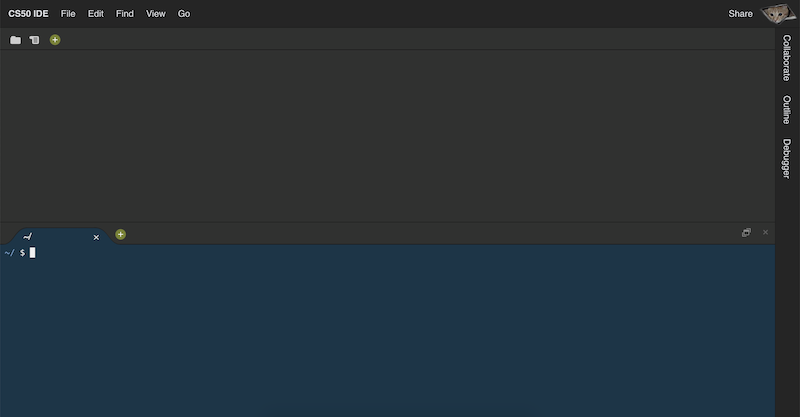
**[Чтение 1](https://cs50.harvard.edu/x/2021/notes/1/" \l "lecture-1)**

* [C](https://cs50.harvard.edu/x/2021/notes/1/#c)
* [CS50 IDE](https://cs50.harvard.edu/x/2021/notes/1/#cs50-ide)
* [Компиляция](https://cs50.harvard.edu/x/2021/notes/1/#compiling)
* [Функции и аргументы](https://cs50.harvard.edu/x/2021/notes/1/#functions-and-arguments)
* [основные, заголовочные файлы](https://cs50.harvard.edu/x/2021/notes/1/#main-header-files)
* [Инструменты](https://cs50.harvard.edu/x/2021/notes/1/#tools)
* [Команды](https://cs50.harvard.edu/x/2021/notes/1/#commands)
* [Типы, коды форматов,](https://cs50.harvard.edu/x/2021/notes/1/#types-format-codes)
* [Операторы, ограничения, усечение](https://cs50.harvard.edu/x/2021/notes/1/#operators-limitations-truncation)
* [Переменные, синтаксический сахар](https://cs50.harvard.edu/x/2021/notes/1/#variables-syntactic-sugar)
* [Условия](https://cs50.harvard.edu/x/2021/notes/1/#conditions)
* [Логические выражения, циклы](https://cs50.harvard.edu/x/2021/notes/1/#boolean-expressions-loops)
* [Абстракция](https://cs50.harvard.edu/x/2021/notes/1/#abstraction)
* [Марио](https://cs50.harvard.edu/x/2021/notes/1/#mario)
* [Память, неточность и переполнение](https://cs50.harvard.edu/x/2021/notes/1/#memory-imprecision-and-overflow)

[**C**](https://cs50.harvard.edu/x/2021/notes/1/#c)

* Сегодня мы изучим новый язык, **C** : язык программирования, который имеет все функции Scratch и многое другое, но, возможно, немного менее дружелюбный, поскольку он чисто текстовый:
* #include <stdio.h>
* int main(void)
* {
* printf("hello, world");
* }
* Хотя сначала, если использовать фразу из Массачусетского технологического института, попытка усвоить все эти новые концепции может показаться вам питьем из пожарного шланга, будьте уверены, что к концу семестра мы получим силы и опыт в изучении и применении этих концепций. .
* Мы можем сравнить множество функций программирования на C с блоками, которые мы уже видели и использовали в Scratch. Детали синтаксиса гораздо менее важны, чем идеи, с которыми мы уже познакомились.
* В нашем примере, хотя слова являются новыми, идеи в точности такие же, как блоки «когда щелкнули зеленый флаг» и «скажи (привет, мир)» в Scratch:  
  
* При написании кода мы можем учитывать следующие качества:
  + **Правильность** , или **правильно** ли работает наш код, как задумано.
  + **Дизайн** или субъективная оценка того, насколько хорошо написан наш код, в зависимости от того, насколько он эффективен и насколько он элегантен или логически читаем, без ненужных повторений.
  + **Стиль** , или насколько эстетично отформатирован наш код, с точки зрения согласованного отступа и другого размещения символов. Различия в стилях не влияют на правильность или смысл нашего кода, но влияют на его визуальную читаемость.

[**CS50 IDE**](https://cs50.harvard.edu/x/2021/notes/1/#cs50-ide)

* Чтобы быстро начать писать наш код, мы воспользуемся инструментом курса [**CS50 IDE**](https://ide.cs50.io/) , *интегрированной средой разработки,* которая включает программы и функции для написания кода. CS50 IDE построена на основе популярной облачной среды IDE, используемой обычными программистами, но с дополнительными образовательными функциями и настройками.
* Мы откроем IDE, и после входа в систему мы увидим такой экран:  
  
  + Пустая верхняя панель будет содержать текстовые файлы, в которые мы можем писать наш код.
  + Нижняя панель, окно **терминала** , позволит нам вводить различные команды и запускать их, включая программы из нашего кода выше.
* Наша IDE работает в облаке и поставляется со стандартным набором инструментов, но мы знаем, что существует множество настольных IDE, предлагающих больше настроек и контроля для различных целей программирования за счет большего времени и усилий на установку.
* В среде IDE мы перейдем к File> New File, а затем File> Save, чтобы сохранить наш файл как hello.c, указывая, что наш файл будет кодом, написанным на C. Мы увидим, что имя нашей вкладки действительно изменилось на hello.c, а теперь вставим наш код сверху:
* #include <stdio.h>
* int main(void)
* {
* printf("hello, world");
* }
* Чтобы запустить нашу программу, мы будем использовать CLI или **интерфейс командной строки** , приглашение, в котором нам нужно вводить текстовые команды. Это контрастирует с **графическим пользовательским интерфейсом** или GUI, таким как Scratch, где у нас есть изображения, значки и кнопки в дополнение к тексту.

[**Компиляция**](https://cs50.harvard.edu/x/2021/notes/1/#compiling)

* В терминале на нижней панели нашей IDE мы **скомпилируем** наш код, прежде чем мы сможем его запустить. Компьютеры понимают только двоичный код, который также используется для представления таких инструкций, как печать чего-либо на экране. Наш **исходный код** был написан символами, которые мы можем прочитать, но его необходимо скомпилировать: преобразовать в **машинный код** , шаблоны нулей и единиц, которые наш компьютер может понять напрямую.
* Программа, называемая **компилятором** , принимает исходный код в качестве входных данных и генерирует машинный код в качестве выходных данных. В среде IDE CS50 у нас уже есть доступ к компилятору с помощью команды **make** . В нашем терминале мы введем текст, make helloкоторый автоматически найдет наш hello.cфайл с нашим исходным кодом и скомпилирует его в программу с именем hello. Будет какой-то вывод, но не будет сообщений об ошибках желтого или красного цвета, поэтому наша программа скомпилирована успешно.
* Чтобы запустить нашу программу, мы введем другую команду, ./helloкоторая ищет в текущей папке, .вызываемую программу hello, и запускает ее.
* Значок $в терминале является индикатором того, где находится приглашение или где мы можем ввести дополнительные команды.

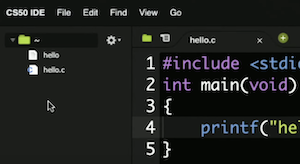
[**Функции и аргументы**](https://cs50.harvard.edu/x/2021/notes/1/#functions-and-arguments)

* Мы будем использовать те же идеи, что исследовали в Scratch.
* **Функции** - это небольшие действия или глаголы, которые мы можем использовать в нашей программе, чтобы что-то делать, а входные данные для функций называются **аргументами** .
  + Например, блок «сказать» в Scratch мог принять что-то вроде «привет, мир» в качестве аргумента. В C вызывается функция вывода чего-либо на экран printf(с обозначением f«форматированный» текст, который мы скоро увидим). А в C мы передаем аргументы в круглых скобках, как в printf("hello, world");. Двойные кавычки указывают на то, что мы хотим распечатать буквы hello, worldбуквально, а точка с запятой в конце указывает на конец нашей строки кода.
* Функции также могут иметь два вида выходов:
  + **побочные эффекты** , например, что-то напечатанное на экране,
  + и **возвращаемые значения** , значение, которое передается нашей программе, которую мы можем использовать или сохранить для дальнейшего использования.
    - Блок «спросить» в Scratch, например, создал блок «ответ».
* Чтобы получить ту же функциональность, что и блок «спросить», мы будем использовать **библиотеку** или уже написанный набор кода. Библиотека CS50 будет включать в себя несколько основных простых функций, которые мы сможем использовать прямо сейчас. Например, get\_stringзапросит у пользователя **строку** или некоторую последовательность текста и вернет ее нашей программе. get\_stringпринимает некоторые входные данные в качестве приглашения для пользователя, например What's your name?, и нам нужно будет сохранить его в переменной с помощью:
* string answer = get\_string("What's your name? ");
  + В C одиночный =символ указывает на **присвоение** или установку значения справа переменной слева. И C вызовет get\_stringфункцию, чтобы сначала получить ее вывод.
  + И нам также нужно указать, что наша переменная с именем answerимеет **тип** строки, поэтому наша программа знает, что нужно интерпретировать нули и единицы как текст.
  + Наконец, нам нужно не забыть добавить точку с запятой в конце нашей строки кода.
* В Scratch мы также использовали блок «answer» в наших блоках «join» и «say». В C мы сделаем это:
* printf("hello, %s", answer);
  + Это %sназывается **кодом формата** , что означает, что мы хотим, чтобы printfфункция подставляла переменную вместо %sзаполнителя. И переменная, которую мы хотим использовать, это answer, которую мы передаем в printfкачестве другого аргумента, отделенного от первого запятой. ( printf("hello, answer")буквально hello, answerкаждый раз распечатывал .)
* Вернувшись в CS50 IDE, мы добавим то, что обнаружили:
* #include <cs50.h>
* #include <stdio.h>
* int main(void)
* {
* string answer = get\_string("What's your name? ");
* printf("hello, %s", answer);
* }
  + Нам нужно указать компилятору включить библиотеку CS50 с #include <cs50.h>, чтобы мы могли использовать get\_stringфункцию.
  + У нас также есть возможность использовать здесь лучший стиль, поскольку мы можем назвать нашу answerпеременную как угодно, но более описательное имя поможет нам лучше понять ее назначение, чем более короткое имя, например aили x.
* После сохранения файла нам нужно будет перекомпилировать нашу программу make hello, поскольку мы изменили только исходный код, но не скомпилированный машинный код. Другие языки или IDE могут не требовать, чтобы мы вручную перекомпилировали наш код после его изменения, но здесь у нас есть возможность для большего контроля и понимания того, что происходит под капотом.
* Теперь ./helloзапустим нашу программу и предложим нам ввести имя, как и предполагалось. Мы могли бы заметить, что следующее приглашение печатается сразу после вывода нашей программы, как в hello, Brian~/ $. Мы можем добавить новую строку после вывода нашей программы, чтобы следующее приглашение было на отдельной строке, с \n:
* printf("hello, %s\n", answer);
  + \nявляется примером **escape-последовательности** или некоторого текста, представляющего другой текст.

[**основные, заголовочные файлы**](https://cs50.harvard.edu/x/2021/notes/1/#main-header-files)

* Блок «при щелчке зеленого флажка» в Scratch запускает то, что мы считаем основной программой. В C первая строка для того же самого - это int main(void), о которой мы узнаем больше в ближайшие недели, за ней следует открытая фигурная скобка {и закрытая фигурная скобка }, обертывающие все, что должно быть в нашей программе.
* int main(void)
* {
* }
  + Мы узнаем больше о том, как мы можем изменить эту строку в ближайшие недели, а пока мы просто воспользуемся этим для запуска нашей программы.
* **Заголовочные файлы,** которые заканчиваются .hссылкой на некоторый другой набор кода, например библиотеку, которую мы затем можем использовать в нашей программе. Мы *включаем* их со строками, как #include <stdio.h>, например, для *стандартной* библиотеки *ввода / вывода* , которая содержит printfфункцию.

[**Инструменты**](https://cs50.harvard.edu/x/2021/notes/1/#tools)

* Благодаря новому синтаксису нам легко сделать ошибку или что-то забыть. У нас есть несколько инструментов, написанных сотрудниками, чтобы помочь нам.
* Мы можем забыть включить строку кода, и когда мы попытаемся скомпилировать нашу программу, увидим много строк сообщений об ошибках, которые трудно понять, поскольку компилятор мог быть разработан для более технической аудитории. **help50**- это команда, которую мы можем запустить для более удобного объяснения проблем в нашем коде. Мы можем запустить его, добавив help50в начало команды, которую мы пытаемся получить, например help50 make hello, чтобы получить более понятный совет.
* Оказывается, в C новые строки и отступы обычно не влияют на работу нашего кода. Например, мы можем изменить нашу mainфункцию на одну строку, int main(void){printf("hello, world");}но ее гораздо труднее читать, поэтому мы сочли бы ее плохим стилем. Мы можем запустить **style50**, как и в случае style50 hello.c, с именем файла нашего исходного кода, чтобы увидеть предложения по новым строкам и отступам.
* Кроме того, мы можем добавлять **комментарии** , заметки в исходный код для себя или других людей, которые не влияют на работу нашего кода. Например, мы могли бы добавить строку вроде // Greet user, с двумя косыми чертами, //чтобы указать, что строка является комментарием, а затем написать цель нашего кода или программы, чтобы помочь нам вспомнить позже.
* **check50**проверим правильность нашего кода с помощью некоторых автоматических тестов. Персонал пишет тесты специально для некоторых программ, которые мы будем писать в курсе, а инструкции по использованию check50будут включены в каждый набор задач или лабораторную работу по мере необходимости. После запуска check50мы увидим какой-то вывод, говорящий нам, прошел ли наш код соответствующие тесты.
* CS50 IDE также дает нам эквивалент нашего собственного компьютера в облаке, где-нибудь в Интернете, с нашими собственными файлами и папками. Если мы щелкнем значок папки в левом верхнем углу, мы увидим дерево файлов, графический интерфейс файлов в нашей среде IDE:  
  
  + Чтобы открыть файл, мы можем просто дважды щелкнуть по нему. hello.c- это исходный код, который мы только что написали, и helloсам по себе будет иметь множество красных точек, каждая из которых является непечатаемым символом, поскольку представляет собой двоичные инструкции для наших компьютеров.

[**Команды**](https://cs50.harvard.edu/x/2021/notes/1/#commands)

* Поскольку CS50 IDE - это виртуальный компьютер в облаке, мы также можем запускать команды, доступные в Linux, такой операционной системе, как macOS или Windows.
* В терминале мы можем ввести ls, сокращенно list, чтобы увидеть список файлов и папок в текущей папке:
* ~/ $ ls
* hello\* hello.c
  + hello зеленый со звездочкой, чтобы указать, что мы можем запустить его как программу.
* Мы также можем *удалить* файлы с rmпомощью такой команды, как rm hello. Он предложит нам для подтверждения, и мы можем ответить yили nна да или нет.
* С помощью mvили *перемещением* мы можем переименовывать файлы. С помощью mv hello.c goodbye.cмы переименовали наш hello.cфайл, чтобы он был назван goodbye.c.
* С помощью mkdirили *создания каталога* мы можем создавать папки или каталоги. Если мы запустим mkdir lecture, мы увидим папку с именем lecture, и мы сможем перемещать файлы в каталоги с помощью такой команды, как mv hello.c lecture/.
* Чтобы *изменить каталоги* в нашем терминале, мы можем использовать cd, как и в случае с cd lecture/. Наше приглашение изменится с ~/на ~/lecture/, показывая, что мы находимся в lectureкаталоге внутри ~. ~обозначает наш домашний каталог или папку верхнего уровня по умолчанию для нашей учетной записи.
* Мы также можем использовать ..как сокращение для родительской или содержащей папку. Внутри ~/lecture/мы можем запустить, mv hello.c ..чтобы переместить его обратно ~, поскольку это родительская папка для lecture/. cd ..аналогично изменит каталог нашего терминала на текущий родительский. Одиночная точка, .относится к текущему каталогу, как в ./hello.
* Теперь, когда наша lecture/папка пуста, мы можем удалить и ее с помощью rmdir lecture/.

[**Типы, коды форматов,**](https://cs50.harvard.edu/x/2021/notes/1/#types-format-codes)

* Есть много **типов** данных, **которые** мы можем использовать для наших переменных, которые указывают компьютеру, какой тип данных они представляют:
  + bool, логическое выражение либо trueилиfalse
  + char, один символ ASCII, например aили2
  + double, значение с плавающей запятой с большим количеством цифр, чем float
  + float, значение с плавающей запятой или вещественное число с десятичным значением
  + int, целые числа до определенного размера или количество бит
  + long, целые числа с большим количеством бит, поэтому они могут считать больше, чем int
  + string, строка символов
* А в библиотеке CS50 есть соответствующие функции для получения входных данных различных типов:
  + get\_char
  + get\_double
  + get\_float
  + get\_int
  + get\_long
  + get\_string
* Для printfкаждого типа тоже есть разные заполнители:
  + %c для символов
  + %f для поплавков, парных
  + %i для целых
  + %li надолго
  + %s для струнных

[**Операторы, ограничения, усечение**](https://cs50.harvard.edu/x/2021/notes/1/#operators-limitations-truncation)

* Мы также можем использовать несколько математических операторов:
  + + для дополнения
  + - для вычитания
  + \* для умножения
  + / для разделения
  + % на остаток
* Составим новую программу addition.c:
* #include <cs50.h>
* #include <stdio.h>
* int main(void)
* {
* int x = get\_int("x: ");
* int y = get\_int("y: ");
* printf("%i\n", x + y);
* }
  + Мы включим файлы заголовков для библиотек, которые мы знаем, что хотим использовать, а затем мы вызовем, get\_intчтобы получить целые числа от пользователя, сохраняя их в переменных с именами xи y.
  + Затем в printf, мы напечатаем заполнитель для целого числа %i, а затем новую строку. Так как мы хотим вывести сумму xи y, мы передадим значение x + yдля printfзамены в строке.
  + Сохраним, запустим make additionв терминале, а потом ./additionпосмотрим, как работает наша программа. Если мы введем что-то, что не является целым числом, мы снова увидим, что get\_intу нас запрашивается целое число. Если мы введем действительно большое число, вроде 4000000000, get\_intтоже снова подскажет. Это связано с тем, что, как и во многих компьютерных системах, intв CS50 IDE 32 бита, которые могут содержать только около четырех миллиардов различных значений. А поскольку целые числа могут быть положительными или отрицательными, максимальное положительное значение для an intможет составлять только около двух миллиардов, а наименьшее отрицательное значение - около двух отрицательных миллиардов, что в сумме составляет около четырех миллиардов значений.
* Мы можем изменить нашу программу, чтобы использовать longтип:
* #include <cs50.h>
* #include <stdio.h>
* int main(void)
* {
* long x = get\_long("x: ");
* long y = get\_long("y: ");
* printf("%li\n", x + y);
* }
  + Теперь мы можем ввести более крупные целые числа и увидеть правильный результат, как и ожидалось.
* Всякий раз, когда мы получаем ошибку во время компиляции, рекомендуется прокрутить вверх, чтобы увидеть первую ошибку и исправить ее в первую очередь, поскольку иногда ошибка в начале программы приводит к тому, что остальная часть программы также интерпретируется с ошибками. .
* Давайте посмотрим на другой пример truncation.c:
* #include <cs50.h>
* #include <stdio.h>
* int main(void)
* {
* // Get numbers from user
* int x = get\_int("x: ");
* int y = get\_int("y: ");
* // Divide x by y
* float z = x / y;
* printf("%f\n", z);
* }
  + Мы будем хранить результат xделится yна z, значения с плавающей точкой, или вещественное число, и распечатать его как поплавок тоже.
  + Но когда мы компилируем и запускаем нашу программу, мы видим zраспечатанные целые числа, такие как 0.000000или 1.000000. Оказывается, в нашем коде *сначала*x / y делится на два целых числа , поэтому результат, возвращаемый операцией деления, также является целым числом. Результат **усекается** , и значение после десятичной запятой теряется. Несмотря на то, что это a , значение, которое мы храним в нем, уже является целым числом.zfloat
  + Чтобы это исправить, мы **бросили** , или преобразовать наши целые числа поплавков , прежде чем разделить их:
  + float z = (float) x / (float) y;
  + Результатом будет float, как мы и ожидали, и на самом деле мы можем использовать только одно из xили yи также получить float.

[**Переменные, синтаксический сахар**](https://cs50.harvard.edu/x/2021/notes/1/#variables-syntactic-sugar)

* В Scratch у нас были такие блоки, как «установить [счетчик] в (0)», которые устанавливают для **переменной** какое-то значение. В C мы бы написали int counter = 0;для того же эффекта.
* Мы можем увеличить значение переменной с помощью counter = counter + 1;, где мы сначала смотрим на правую сторону, беря исходное значение counter, добавляя 1, а затем сохраняя его в левой части ( counterв данном случае обратно ).
* C также поддерживает **синтаксический сахар** или сокращенные выражения для тех же функций. В этом случае мы могли бы эквивалентным образом сказать, counter += 1;чтобы добавить один, counterпрежде чем снова сохранить его. Мы также могли бы просто написать counter++;, и мы можем изучить это (и другие примеры), просмотрев документацию или другие ссылки в Интернете.

[**Условия**](https://cs50.harvard.edu/x/2021/notes/1/#conditions)

* Мы можем переводить условия или блоки «если» с помощью:
* if (x < y)
* {
* printf("x is less than y\n");
* }
  + Обратите внимание, что в C мы используем {и }(а также отступ), чтобы указать, как должны быть вложены строки кода.
* У нас могут быть условия if и else:
* if (x < y)
* {
* printf("x is less than y\n");
* }
* else
* {
* printf("x is not less than y\n");
* }
* И даже «иначе, если»:
* if (x < y)
* {
* printf("x is less than y\n");
* }
* else if (x > y)
* {
* printf("x is greater than y\n");
* }
* else if (x == y)
* {
* printf("x is equal to y\n");
* }
  + Обратите внимание, что для сравнения двух значений в C мы используем ==два знака равенства.
  + И, по логике, нам не нужно if (x == y)в конечном условии, поскольку это единственный оставшийся случай, поэтому мы можем просто сказать else:
  + if (x < y)
  + {
  + printf("x is less than y\n");
  + }
  + else if (x > y)
  + {
  + printf("x is greater than y\n");
  + }
  + else
  + {
  + printf("x is equal to y\n");
  + }
* Давайте посмотрим на другой пример conditions.c:
* #include <cs50.h>
* #include <stdio.h>
* int main(void)
* {
* // Prompt user for x
* int x = get\_int("x: ");
* // Prompt user for y
* int y = get\_int("y: ");
* // Compare x and y
* if (x < y)
* {
* printf("x is less than y\n");
* }
* else if (x > y)
* {
* printf("x is greater than y\n");
* }
* else
* {
* printf("x is equal to y\n");
* }
* }
  + Мы включили только что увиденные условия, а также два вызова или использования get\_intдля получения xи yот пользователя.
  + Мы скомпилируем и запустим нашу программу, чтобы убедиться, что она действительно работает так, как задумано.
* В agree.c, мы можем попросить пользователя подтвердить или опровергнуть что-то:
* #include <cs50.h>
* #include <stdio.h>
* int main(void)
* {
* char c = get\_char("Do you agree? ");
* // Check whether agreed
* if (c == 'Y' || c == 'y')
* {
* printf("Agreed.\n");
* }
* else if (c == 'N' || c == 'n')
* {
* printf("Not agreed.\n");
* }
* }
  + С помощью get\_charмы можем получить один символ, и, поскольку в нашей программе есть только один, кажется разумным называть его c.
  + Мы используем две вертикальные полосы,, ||чтобы указать логическое «или», может ли какое-либо выражение быть истинным для условия, которому необходимо следовать. (Два амперсанда,, &&обозначают логическое «и», где должны выполняться оба условия.) И обратите внимание, что мы используем два знака равенства,, ==для сравнения двух значений, а также одинарные кавычки ', чтобы заключить наши значения одиночных символы.
  + Если ни одно из выражений не является истинным, ничего не произойдет, поскольку в нашей программе нет цикла.

[**Логические выражения, циклы**](https://cs50.harvard.edu/x/2021/notes/1/#boolean-expressions-loops)

* Мы можем перевести блок «навсегда» в Scratch с помощью:
* while (true)
* {
* printf("hello, world\n");
* }
  + whileКлючевое слово требует состояние, поэтому мы используем trueкак логическое выражение , чтобы убедиться , что наш цикл будет работать вечно. whileсообщит компьютеру, чтобы он проверил, соответствует ли выражение выражению true, а затем выполнит строки внутри фигурных скобок. Затем он будет повторять это до тех пор, пока выражение не перестанет быть верным. В этом случае trueвсегда будет true, поэтому наш цикл будет **бесконечным** или будет выполняться вечно.
* Мы могли бы делать что-то определенное количество раз с помощью while:
* int i = 0;
* while (i < 50)
* {
* printf("hello, world\n");
* i++;
* }
  + Мы создаем переменную, iи устанавливаем ее в 0. Затем, пока iона меньше 50, мы запускаем несколько строк кода, в том числе одну, в которой iкаждый раз добавляем 1 . Таким образом, наш цикл в конечном итоге завершится, когда iдостигнет значения 50.
  + В этом случае мы используем переменную iкак счетчик, но, поскольку она не служит какой-либо дополнительной цели, мы можем просто дать ей имя i.
* Несмотря на то, что мы *могли бы* сделать следующее и начать отсчет с 1, по соглашению мы должны начать с 0:
* int i = 1;
* while (i <= 50)
* {
* printf("hello, world\n");
* i++;
* }
* Другое правильное, но, возможно, менее продуманное решение может начинаться с 50 и считаться в обратном порядке:
* int i = 50;
* while (i > 0)
* {
* printf("hello, world\n");
* i--;
* }
  + В этом случае логику нашего цикла труднее осмыслить, не служа какой-либо дополнительной цели, и она может даже сбить с толку читателей.
* Наконец, чаще мы можем использовать forключевое слово:
* for (int i = 0; i < 50; i++)
* {
* printf("hello, world\n");
* }
  + Опять же, сначала мы создаем переменную с именем iи устанавливаем ее в 0. Затем мы проверяем это i < 50каждый раз, когда достигаем вершины цикла, прежде чем запускать какой-либо код внутри. Если это выражение истинно, мы запускаем код внутри. Наконец, после того, как мы запустим код внутри, мы i++добавляем единицу i, и цикл повторяется.
  + В этом случае forцикл более элегантен, чем whileцикл, поскольку все, что связано с циклом, находится в одной строке, и только код, который мы действительно хотим запускать несколько раз, находится внутри цикла.
* Обратите внимание, что для многих из этих строк кода, таких как ifусловия и forциклы, мы не ставим точку с запятой в конце. Именно так много лет назад был разработан язык Си, и по общему правилу только строки для действий или глаголов имеют точку с запятой в конце.

[**Абстракция**](https://cs50.harvard.edu/x/2021/notes/1/#abstraction)

* Мы можем написать программу, которая печатает meowтри раза:
* #include <stdio.h>
* int main(void)
* {
* printf("meow\n");
* printf("meow\n");
* printf("meow\n");
* }
* Мы могли бы использовать forцикл, поэтому нам не нужно копировать и вставлять столько строк:
* #include <stdio.h>
* int main(void)
* {
* for (int i = 0; i < 3; i++)
* {
* printf("meow\n");
* }
* }
* Мы можем переместить printfстроку в ее собственную функцию, например, в нашу собственную головоломку:
* #include <stdio.h>
* void meow(void) {
* printf("meow\n");
* }
* int main(void)
* {
* for (int i = 0; i < 3; i++)
* {
* meow();
* }
* }
  + Мы определили функцию meowнад нашей mainфункцией.
* Но условно наша mainфункция должна быть первой функцией в нашей программе, поэтому нам нужно еще несколько строк:
* #include <stdio.h>
* void meow(void);
* int main(void)
* {
* for (int i = 0; i < 3; i++)
* {
* meow();
* }
* }
* void meow(void)
* {
* printf("meow\n");
* }
  + Оказывается, нам нужно meowсначала объявить нашу функцию с **прототипом** , прежде чем использовать ее main, и фактически определить ее после. Компилятор читает наш исходный код сверху вниз, поэтому ему нужно знать, что он meowбудет существовать позже в файле.
* Мы даже можем изменить нашу meowфункцию, чтобы она принимала некоторый ввод n, и nвремя мяуканья :
* #include <stdio.h>
* void meow(int n);
* int main(void)
* {
* meow(3);
* }
* void meow(int n)
* {
* for (int i = 0; i < n; i++)
* {
* printf("meow\n");
* }
* }
  + voidДо meowфункции означает , что она не возвращает значения, а также в mainмы ничего не можем с результатом делать meow, так что мы просто называем.
* Абстракция здесь приводит к лучшему дизайну, поскольку теперь у нас есть возможность повторно использовать нашу meowфункцию в нескольких местах в будущем.
* Давайте посмотрим на другой пример абстракции get\_positive\_int.c:
* #include <cs50.h>
* #include <stdio.h>
* int get\_positive\_int(void);
* int main(void)
* {
* int i = get\_positive\_int();
* printf("%i\n", i);
* }
* // Prompt user for positive integer
* int get\_positive\_int(void)
* {
* int n;
* do
* {
* n = get\_int("Positive Integer: ");
* }
* while (n < 1);
* return n;
* }
  + У нас есть собственная функция, которая вызывает get\_intнесколько раз, пока мы не получим какое-то целое число, которое *не* меньше 1. В цикле do-while наша программа сначала что-то сделает, затем проверит какое-то условие и будет повторять, пока условие истинно. С другой стороны, цикл while сначала проверяет условие.
  + Нам нужно объявить наше целое число nвне цикла do-while, поскольку нам нужно использовать его после завершения цикла. Область **видимости** переменной в C относится к контексту или строкам кода, в которых она существует. Во многих случаях это будут фигурные скобки, окружающие переменную.
  + Обратите внимание, что функция get\_positive\_intтеперь начинается с int, что указывает на то, что она имеет возвращаемое значение типа int, и mainмы действительно сохраняем его iпосле вызова get\_positive\_int(). В get\_positive\_int, у нас есть новое ключевое слово, returnчтобы вернуть значение nтуда, где была вызвана функция.

[**Марио**](https://cs50.harvard.edu/x/2021/notes/1/#mario)

* Нам может понадобиться программа, которая печатает часть экрана из видеоигры, такой как Super Mario Bros.В mario.c, мы можем напечатать четыре вопросительных знака, имитируя блоки:
* #include <stdio.h>
* int main(void)
* {
* printf("????\n");
* }
* С помощью цикла мы можем вывести несколько вопросительных знаков, после них ставить одну новую строку после цикла:
* #include <stdio.h>
* int main(void)
* {
* for (int i = 0; i < 4; i++)
* {
* printf("?");
* }
* printf("\n");
* }
* Мы можем получить от пользователя положительное целое число и распечатать это количество вопросительных знаков, используя nдля нашего цикла:
* #include <cs50.h>
* #include <stdio.h>
* int main(void)
* {
* // Get positive integer from user
* int n;
* do
* {
* n = get\_int("Width: ");
* }
* while (n < 1);
* // Print out that many question marks
* for (int i = 0; i < n; i++)
* {
* printf("?");
* }
* printf("\n");
* }
* И мы можем распечатать двумерный набор блоков с вложенными циклами, один внутри другого:
* #include <cs50.h>
* #include <stdio.h>
* int main(void)
* {
* for (int i = 0; i < 3; i++)
* {
* for (int j = 0; j < 3; j++)
* {
* printf("#");
* }
* printf("\n");
* }
* }
  + У нас есть два вложенных цикла, где внешний цикл использует iвсе, что находится внутри, 3 раза, а внутренний цикл использует jдругую переменную, чтобы делать что-то 3 раза в каждый из *этих* циклов . Другими словами, внешний цикл печатает 3 «строки» или строки, заканчивая каждую из них новой строкой, а внутренний цикл печатает 3 «столбца» или #символа *без* новой строки.

[**Память, неточность и переполнение**](https://cs50.harvard.edu/x/2021/notes/1/#memory-imprecision-and-overflow)

* В нашем компьютере есть оперативная память в виде аппаратных микросхем, называемая ОЗУ. Наши программы используют эту оперативную память для хранения данных во время работы, но объем этой памяти ограничен.
* С помощью imprecision.cмы можем увидеть, что происходит, когда мы используем числа с плавающей запятой:
* #include <cs50.h>
* #include <stdio.h>
* int main(void)
* {
* float x = get\_float("x: ");
* float y = get\_float("y: ");
* printf("%.50f\n", x / y);
* }
  + С помощью %.50fмы можем указать количество отображаемых десятичных знаков.
  + Хм, теперь мы получаем ...
  + x: 1
  + y: 10
  + 0.10000000149011611938476562500000000000000000000000
  + Оказывается, это называется **неточностью с плавающей запятой** , когда у нас недостаточно битов для хранения всех возможных значений. Имея конечное число битов для a float, мы не можем представить все возможные действительные числа (которых *бесконечное* количество), поэтому компьютер должен сохранить максимально близкое значение. И это может привести к проблемам, когда складываются даже небольшие различия в значениях, если программист не использует какой-либо другой способ представления десятичных значений настолько точно, насколько это необходимо.
* На прошлой неделе, когда у нас было три бита и нужно рассчитывать больше , чем семь (или 111), мы добавили еще немного , чтобы получить восемь, 1000. Но если бы у нас было всего три бита, у нас не было бы места для лишних 1. Он исчезнет, ​​и мы вернемся 000. Эта проблема называется **целочисленным переполнением** , когда целое число может быть настолько большим, пока не закончатся биты.
* Проблема 2000 года возникла из-за того, что многие программы сохраняли календарный год только с двумя цифрами, например 98 для 1998 года и 99 для 1999 года. Но когда приблизился 2000 год, программы должны были хранить только 00, что привело к путанице между 1900 и 2000 годами. .
* В 2038 году у нас также закончится бит для отслеживания времени, поскольку много лет назад некоторые люди решили использовать 32 бита в качестве стандартного количества бит для подсчета количества секунд с 1 января 1970 года. Но с 32 битами, представляющими только положительные числа, мы можем сосчитать только до четырех миллиардов, и в 2038 году мы достигнем этого предела, если не обновим программное обеспечение во всех наших компьютерных системах.