

El Retorno de

WALL·E



© 2008 WALT DISNEY PICTURES

Proyecto de Programación

Carrera de Ciencia de la Computación

Universidad de La Habana

2018-2019



ВВЕДЕНИЕ

Набор объектов расположен в прямоугольном мире, разделенном на коробки. Некоторые из этих объектов являются роботами, которых можно запрограммировать на выполнение задач (даже на соревнование друг с другом). Роботы могут двигаться вперед или назад, а также поворачиваться на 90 градусов влево или вправо. Направление робота всегда будет одним из углов (0° , 90° , 180° или 270° по отношению к северу). На местности могут быть крупные объекты (препятствия), средние объекты (подвижные) и мелкие объекты (переносимые).

Если робот движется над препятствием, он не двигается, если он движется над подвижным объектом (когда это возможно), он перемещает его, а если он движется над небольшим объектом, он поднимает его (схватывает внутри) или перемещает, если он занят внутри. Если квадрат перед роботом свободен, робот может сбросить (выгрузить) предмет внутри него.

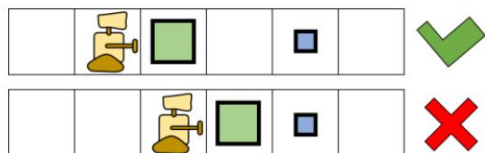
Робот имеет центральное вычислительное ядро, способное выполнять набор инструкций. Он также имеет датчики, позволяющие «воспринимать» характеристики окружающей среды. Например, ультразвуковой датчик для определения расстояния или веб-камера для определения цвета ближайшего объекта впереди. Датчики позволяют получать информацию из окружающей среды, и робот может принимать решения в соответствии с ней.

ОБЪЕКТЫ

Объекты можно классифицировать по различным признакам (размеру, форме и цвету). Размер и форма объекта определяют, можно ли его поднять, толкнуть или он является препятствием, мешающим продвижению робота. Размер объекта перед роботом можно узнать через атрибут `size`, а его форму — через атрибут `shape`. Кроме того, объекты имеют цвет (достоверный через атрибут `color`) и штрих-код (атрибут `number`). Последние не влияют на взаимодействие между роботом и объектами, но могут использоваться для определения задач робота (например, найти черный шар).

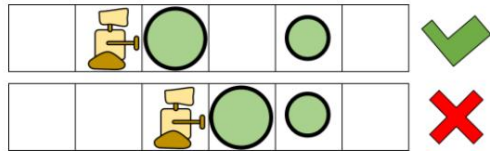
Если предмет маленький (форма значения не имеет), робот может схватить его (переместившись в квадрат, где находится предмет). Если объект средний (или маленький, а в работе уже есть небольшой объект), его можно толкнуть. Правила перемещения объекта следующие:

- Робот никогда не может вытеснить другого робота.
- Среднюю или маленькую коробку можно перемещать до тех пор, пока следующая клетка свободна.

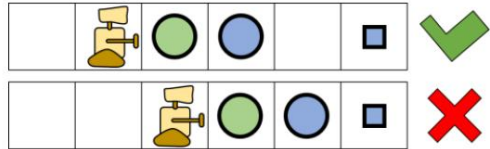


- Большой шар подвижен до тех пор, пока следующая клетка пуста.





- Средний или малый мяч подвижен до тех пор, пока после серии маленьких или к медианам примыкает незанятый квадрат.



Если робот роняет предмет, он делает это на квадрат перед собой. Если в этом поле уже есть объект, действие не имеет никакого эффекта.

Для оценки поведения роботов в контролируемой среде был разработан язык программирования MATLAB. В этом языке инструкции появляются в массиве, и выполнение (в отличие от последовательных инструкций) происходит линейно по адресу в массиве. С помощью языка вы можете объявить программу

каждого робота.



МАТЛАН ЯЗЫК

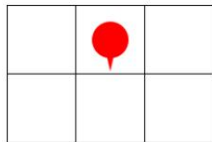
Язык MATL AN (Matrix L anguage) — это визуальный и императивный язык с единой памятью, разделяемой всеми подпрограммами. В нем вы можете объявлять подпрограммы (метод, процедуру) или описывать набор инструкций, которые должен выполнить робот.

Каждая процедура (включая основную процедуру) состоит из двумерного массива инструкций. Одинаковый Он начинает выполнение с одной инструкции «с тарт», которая должна находиться в каком-то блоке. Начальное направление всегда вниз.



«Выполнение» проходит по всем ячейкам в определенном направлении и выполняет каждую инструкцию на своем пути. Если выполнение выходит за границы массива инструкций, говоря о том, что программа (или подпрограмма) завершена. Если вы находитесь в основной программе, симуляция завершается, если вы находитесь в подпрограмме, вы возвращаетесь к точке, где она была вызвана, и продолжаете в том же направлении, что и до вызова.

На следующем рисунке представлена возможная программа MATL AN, которая ничего не делает.



Пустые поля не влияют на выполнение программы (это инструкции, которые ничего не делают) и не меняют ход выполнения.






Если вы хотите прервать выполнение подпрограммы, когда достигнете конца, вы можете сделать это явным образом с помощью оператора (return).



ИНСТРУКЦИИ ПО РАБОТУ

Основная идея программы — выразить логику работы робота. Что ему делать, как реагировать на ту среду, в которой он оказался. В языке есть два типа инструкций о роботах: действия и датчики. Действия указывают, что робот должен делать, а датчики позволяют фиксировать значения характеристик, присутствующих в объектах, с которыми робот сталкивается или вокруг которых.

В следующей таблице описаны основные действия, которые может выполнять робот.

Название инструкции	Описание
	Робот продвигается на одну клетку вперед, пока перед ним нет препятствия или он не выйдет за пределы габаритов местности (представьте, что края окружены блоками). Если найден небольшой предмет и робот не заполнен, он поднимает его. Если нет, то если объект найден, он попытается его переместить.
	Робот перемещается на одну клетку назад, если это пустая клетка. Назад не поднимает предметы и не двигается.
	Поверните робота на 90 градусов вправо. Это действие возможно всегда.
	Поверните робота на 90 градусов влево. Это действие возможно всегда.
	Выбрасывает объект, который находится внутри робота (если он есть).

Следующая программа иллюстрирует, как можно реализовать робота, который делает два шага, поворачивает направо и перемещается вперед еще на одну клетку (пустые клетки справа не имеют никакого эффекта, но показывают, что массив может иметь любую размерность, пока квадраты могут быть организованы). Необходимые инструкции).

ПАМЯТЬ

У роботов есть память для хранения программ, стек значений, фиксированный набор регистров для каждого вызова подпрограммы и одномерная память, к которой можно обратиться для чтения и записи в любом месте. Стек начинается выполнение пустым, а линейная память начинается со всех значений, равных 0 (линейная память имеет фиксированную емкость 1 000 000 целых чисел). Все регистры имеют начальное значение 0 в начале подпрограммы.

Инструкции в программе могут складывать значения, извлекать значения или извлекать, оперировать и складывать. Некоторые инструкции, которые обращаются к стеку, описаны ниже.

СОХРАНЕНИЕ КОНСТАНТЫ В стеке

В MATL AN вы не можете записывать числовые значения. Вместо этого есть три инструкции, которые позволяют вам работать с остеком и «сстроить» любое целочисленное значение из его двоичного представления. Для этого есть следующие инструкции (хотя их можно использовать и для чего-то еще, что вам нужно).

Инструкция	Имя	Описание
	номер_	Сложите значение 0.
	ноль (ноль)	Извлекает значение из стека и удваивает это значение.
	Один (один)	Извлекает значение из стека и дважды помещает это значение, увеличенное на 1.

Обратите внимание, что последовательность инструкций **# 1 0 1 1** позволяет поместить в стек значение $11 = 1*2^3 + 0*2^2 + 1*2^1 + 1*2^0$.

0, **1** 0*2+1=1, **0** 1*2=2, **1** 2*2+1=5, **1** 5*2+1=11

ОПЕРАТОРЫ

Операторы в MATL AN могут работать с 1, 2 или 3 значениями из стека. Они всегда доставляют значение, полученное в результате операции, в стек. Допустимые операции в MATL AN такие же, как и в C#. Первое значение, извлеченное из стека, является последним операндом. Ниже приведены некоторые операторы и способы их использования.

Лето 5 + 3: **#101#11+**

Получите -6: **##110-**

Условные и логические операторы (&, || и !) работают с целыми числами, предполагая, что значения, отличные от 0, являются истинными, а 0 — ложными. Истинный результат выражается как 1, а ложный результат как 0.

Обратите внимание, что следующая последовательность инструкций оставляет 1 в стеке.

1 0 # 0 1 &&



В отличие от логического оператора &, который выполняет оператор and побитно, поэтому в стеке остается 0.

1 0 # 0 1 &

Операторы +,	Имена	количество операндов
-, *, /, %	add, sub, mul, div, mod inc, dec	2
++, --	land, lor, lxor, lnot and, or, not	1
&, , ^, ~		2, 2, 2, 1
&&, , !		2, 2, 1
==, !=	eq, neq lt,	2
<, <=, >, >=	leq, gt, geq тройной	2
?		3



ЗАПИСИ

Регистры функционируют как локальные переменные при каждом вызове подпрограммы. Они могут брать значение из стека или помещать свое значение в стек. Регистры обозначаются буквами от A до Z и для каждого по два инструкции.

Название инструкции	Описание
 положить (получить)	Сложите значение A.
 Сохранить в A (набор A)	Извлекает значение из стека и присваивает его регистру A.




ЛИНЕЙНАЯ ПАМЯТЬ





Линейная память работает как большой массив, к которому можно получить доступ для чтения или сохранения значения в любом из его местоположений. Этот массив является общим для всех подпрограмм робота. Для этого есть две инструкции.

Инструкция	Имя	Описание
 читать из массива (получить)		Извлекает из стека значение, указывая еще индекс в запрашиваемом массиве. Сохраняет значение массива в этой позиции в стеке.
 Назначить в массиве (установлен на)		Выталкивает два значения из стека. Первый указывает индекс, а второй значение, которое будет сохранено. Выделяет это значение в линейной памяти в позиции, указанной индексом.

ДАТЧИКИ

Датчики позволяют роботу воспринимать особенности окружающей среды и наклеивать «прочитанное» значение каждый раз, когда выполняется инструкция.

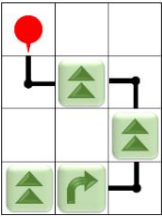
Инструкция датчика	Имя	Значения и описание
 Ультразвуковой	расстояние	Целое число, указывающее количество незанятых квадратов перед роботом (в мировых пределах).
 Веб-камера	цвет	Целое число, указывающее цвет объекта впереди. Если объекта нет, складывается 0.
 Кинект	форма	Целое число, указывающее форму объекта впереди. Если объекта нет, складывается 0.

	Сканер штрих-кода	код	Целое число, указывающее штрих-код объекта впереди. Если объекта нет, складывается 0.
	Денги	загружен	1, если внутри робота есть объект, иначе 0.
	Хронометр	время	Целое число, указывающее время с начала «жизни» робота. (Количество раундов моделирования). В каждом раунде роботы выполняют свою программу до следующей «инструкции» робота.
	Компас	направление	Целое число, указывающее ориентацию робота.

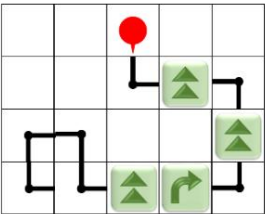
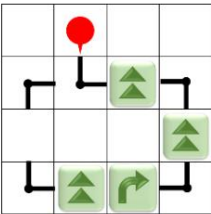
КОНТРОЛЬ ПОТОКА

Выполнение в MATLAB не всегда последовательно. Некоторые инструкции позволяют изменить направление выполнения и даже прекратить его.

Углы позволяют изменить ориентацию потока, чтобы он не всегда был в одном и том же направлении. Следующая программа действительно и представляет тот же пример, что и предыдущая программа.







Обратите внимание, что на основе angular мы можем получить циклы, но они будут бесконечными. Даже повторить инструкции, но в обратном направлении.



Если достигнута команда запуска, процесс повторяется снова, всегда в исходном направлении. Обратите внимание, что углы влияют на направление выполнения только в том случае, если оно достигает коробки на одном из концов угла.

Если кто-то прибывает с одной из двух других сторон, исполнение сменяет свое направление.

Инструкция	Имя
	NE
	SE
	SW
	NW

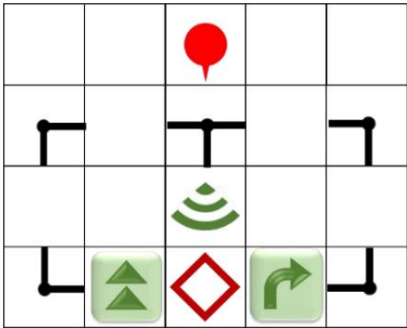
УСЛОВИЯ

Условные операторы (ветвь) — это инструкции, которые работают следующим образом. Извлечь значение из стека. Если это ложно, оно отклоняет выполнение влево (в зависимости от того, откуда оно исходит), а если оно истинно (отлично от 0), оно отклоняет выполнение вправо.

Следующая программа выполняет следующее, робот запрашивает датчик глубины (создает расстояние до ближайшего объекта или ребра в стеке), затем условная проверка, если оно отличается от 0 (перед ним есть место) поток переходит к справа (который, как видно сверху, будет циклом слева) и приказывает роботу двигаться вперед.

Если в стеке должен храниться 0 (ближайший объект находится прямо перед роботом), поток поворачивается влево, и робот получает команду повернуться. Углы помогают повторить процесс. В общем, эта программа заставляет робота двигаться вперед, когда это возможно, а при столкновении с объектом поворачивает вправо и продолжает попытки.

двигаться вперед






Обратите внимание на использование Ts. T — это обработчик потока, который работает следующим образом. У трех концов. Два выровненных (A и B) и один перпендикулярный (C). Всякий раз, когда вы достигнете одного из концов A или B, вы поворачиваете, чтобы выйти через конец C. Но если вы прибываете в конец C, один из концов A или B выбирается случайным образом для продолжения.

Инструкция	Имя
	ТС
	НАШИ
	Тамбаси
	ТВ




РУТИНЫ

В робот загружается ряд подпрограмм (индексация начинается с индекса 0 для основной программы). Каждая подпрограмма имеет свой собственный массив для определения своих инструкций и инструкций запуска. Подпрограммы могут вызывать себя, допуская рекурсию!

Чтобы вызвать подпрограмму, необходимо использовать оператор call. Следующие инструкции относятся к вызову подпрограмм.




Инс трукц ия	Имя	Опис ание
	Вызов (вызов)	Извлекает из стека число, являющееся индексом вызываемой подпрограммы. Выполните вызов подпрограммы. По его окончании он возвращается в точку вызова и продолжается в том же направлении, в котором был.
	Индекс чего-либо рутина (рутина)	Складывает индекс текущей подпрограммы, которую он выполняет.
	Выполняет вызов самой подпрограммы. Это эквивалентно выполнению процедуры, а затем вызову. Рекурсивный вызов (recCall)	

Следующая программа вычисляет N-е значение Фибоначчи.

	F	I	B	O	N	A	C	C	I
N ←									
└─	N	#	1	>	└─				
└─				◇	#	1			
└─	N	#	1	-					
└─									
└─	N	#	1	0	-		+		

АВТО-ГИДЫ

Одним из вариантов, за который будет получен бонус, является автоматическое добавление указания (серая линия) в пустые поля, где будет происходить исполнение, в дополнение к предоставлению возврата во все возможные поля, где известно, что конец достигнуто выполнение подпрограммы. Авто-гид для приведенного выше примера будет выглядеть так, как показано на рисунке ниже.

	F	I	B	O	N	A	C	C	I
N ←									
└─	N	#	1	>	└─				
└─				◇	#	1	⊗		
└─	N	#	1	-					
└─									
└─	N	#	1	0	-		+	⊗	

Обратите внимание, что запись и в первой строке не достигаются, но они также не являются проблемой. Ячейки, не достигнутые выполнением, могут иметь любые инструкции. Это можно использовать для оставления «комментариев» в коде.



МОДЕЛИРОВАНИЕ СРЕДЫ

Как указано в начале документа, изначально роботы располагаются на прямоугольном поле, разделенном на квадратные ячейки. В ячейки также можно помещать предметы с различными свойствами, такими как размер, цвет, форма, код.

В начале есть пустая карта по умолчанию 10x20 ячеек, хотя эту конфигурацию можно изменить. После того, как карта создана, на ней можно размещать объекты. Каждый робот может загрузить программу (последовательность процедур).

Роботы всегда имеют фиксированные атрибуты `shape=bot` и `size=large`.

Чтобы все роботы продвигались шаг за шагом (выполняли свой код до достижения физических ограничений), в программе симулятора должна быть активирована кнопка продвижения. Вы также можете дать команду выполнять одну инструкцию

за раз. Выполнение на каждом роботе должно графически иллюстрироваться, на какой инструкции (блоке) он находится в данный момент. Кнопка воспроизведения должна автоматически запускать симуляцию с небольшим интервалом между каждым рондом.

ЛОС РОБОТС



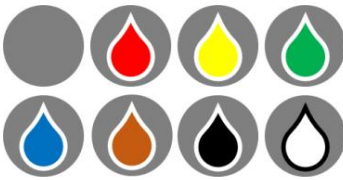




У каждого робота есть программа, которая постоянно описывает его поведение. Используемый язык будет МАТЛАН. Кроме того, есть некоторые атрибуты, которые выражают состояние его датчиков.

Так как на карте есть несколько роботов, симулятор должен имитировать одновременное выполнение инструкций с всей программой. Однако на деле это не произойдет. В каждом раунде симулятора все роботы должны «сдвинуться» на один шаг. Предполагается, что общие инструкции программы выполняются мгновенно, поэтому единственное, что может означать «требуется времени», — это какое-то физическое действие робота, например, движение вперед, поворот, падение и т. д. Время этих действий будет одинаковым для всех роботов, их ориентировочно оно может составлять одну секунду, это возможный параметр симуляции, и поэтому можно увидеть, что процесс происходит медленнее или быстрее.

Порядок роботов следует считать случайным в каждом «раунде». Таким образом, ни один робот не будет иметь преимущества перед другими из-за порядка, в котором они были созданы. Как только будет установлен порядок, в котором будет моделировать «следующий шаг» каждого робота, код будет выполняться в каждом из них (с последней позиции, в которой он оставался с предыдущего раунда) до физического или физического действие выполнено достигнут конец кода. Если достигается физическое действие, оно выполняется и передается следующему роботу. Если достигнут конец кода, робот останавливается и начинает выполнение с начала в следующем раунде.

Если робот выполняет действие, которое не происходит из-за особенностей карты (например, движется против препятствия), он все равно «теряет» свой ход (как автомобили с дистанционным управлением, когда они застревают в стене, но все равно двигаются). колеса).

В языке существуют различные целочисленные константы для выражения возможных значений цвета, формы, направление.

инс трукц ии	Nombre	Значение, которое суммируется
	прозрачный красный желтый зеленый синий коричневый черный белый	0 1 2 3 4 5 6 7
	коробка со сферой завод робот	0 1 2 3 4
	пустой маленький средний	0 1 2 3
	большой север восток юг запад	0 1 2 3
	ложь истинный	0 1

Роботы имеют собственную память для своей программы, переменных, подпрограмм и т.д. по той причине, что между ними нет иной формы общения, кроме как через окружающую среду.



ПРИЛОЖЕНИЕ _

Визуальное приложение должно позволять программно проектировать сцену, в которой размещаются объекты и роботы и выполняются моделирование.

Все программы должны иметь возможность сохранения в файлах для использования в будущем. Формат, в котором будут сохраняться программы, будет следующим:

Число N с количеством инструкций

В каждой следующей строке:

Столбец строки Name_of_the_Instruction

В рамках спецификации проекта вам будет представлено примерное приложение, позволяющее обрабатывать набор кодов и имеющее процесс отображения матрицы с некоторыми значками. Вы можете использовать этот проект, чтобы превратить его в свой собственный симулятор.

В этом документе представлен небольшой набор новых инструкций, датчиков и объектов... Не оглашайте названий...

Представляются новые типы датчиков и инструкций. Обратите внимание, что большой вес оценки определяется предлагаемой расширяемостью. То есть насколько легко включить новый оператор, команду, объект и т. д.

Поставьте своей целью сделать ваше решение достаточно расширяемым, чтобы новый возможный оператор для степени ($2^3 == 8$) или новая концепция могли быть включены без особых трудностей.

Будь креативным!

Однако вы не можете имитировать ни одну из команд, описанных в этом документе, поскольку ваш проект будет оцениваться с помощью кодов, предложенных учителями.



О Г О Т О В Н О С Т И К О Д А

И Д Е Н Т И Ф И К А Т О Р Ы

Все идентификаторы (имена переменных, методов, классов и т. д.) должны быть установлены тщательно, чтобы кто-то, кроме их одного проگرامмиста, мог легко понять, для чего они используются.

Имена переменных должны максимально точно указывать информацию, которая в них хранится.

Например, если «количество препятствий, найденных на данный момент», хранится в переменной, ее имя должно быть `numberOfObstaclesFound` или `numberOfObstacles`, если первое кажется слишком длинным, но никогда не `countObstacles`, `temp`, `myVariable`, `john`, `counter`, `counting` или для `NoFlash`. Обратите внимание, что последний некорректный идентификатор отлично читается, но указывает не на то, «какая информация хранится в переменной», а на то, что, возможно, для чего она используется информацию, которую я там храню», что тоже не то, чем она должна быть.

- Между именем переменной, которое немного длинное и описательное, и именем, которое нелегко понять для всех предпочтительнее длинный.
- Как правило, имена переменных не должны быть словами или фразами, обозначающими действия, например `удаление`, `прыжок` или `остановка`.

Есть несколько (очень немногих) случаев, когда для переменных могут использоваться не очень описательные идентификаторы.

Как правило, это очень распространенные небольшие фрагменты кода, которые «все знают, для чего они нужны».

Например:

```
интервал темп = a;  
a = b;  
b = температура
```

«Все» знают, что приведенный выше код представляет собой обмен или swap между значениями переменных `a` и `b`, а также что переменная `temp` используется для хранения одного из двух значений на мгновение. Практически в любом другом контексте использование `temp` в качестве имени переменной некорректно, так как оно указывает лишь на то, что оно будет использоваться для «временного» хранения значения, а в конечном итоге для этого используются все переменные.

В качестве второго примера, если вы хотите выполнить что-то десять раз, вы можете сделать

```
для (целое я = 0; я < 10; я++)  
...
```

вместо

```
for (int iterationCurrent = 0; iterationCurrent < 10; iterationCurrent++)  
...
```

Для имен свойств (properties на английском языке) применяется тот же принцип, что и для переменных, то есть для выражения «что они возвращают» или «что они представляют», только идентификаторы должны начинаться с заглавных букв. Они не должны быть словосочетаниями, обозначающими действия, непонятными сокращениями и т. п.

Имена методов должны отражать «что делает метод», и обычно рекомендуется использовать для этого инфинитив или повелительный глагол: Add, Remove, ConcatenateArrays, CountWords, Start, Stop и т. д.

В случаях с классами, очевидно, их идентификаторы также должны давать понять, что представляет собой класс: Робот, Препятствие, Среда, Программа.

КОММЕНТАРИИ

Комментарии также являются важным элементом в понимании кода человеком, которому необходимо его адаптировать или исправить, и который не обязательно был тем, кто его прокомментировал или не делал этого в последнее время. Включая комментарии в свой код, помните, что они адресованы не только вам, но и любому программисту. Например, может быть, вам досадно читать длинные комментарии, чтобы понять, что делает определенный фрагмент кода или для чего он используется переменная:

```
// что произошло с оной в тот день
```

Очевидно, кому-то другому эти комментарии будут не очень полезны.

Некоторые рекомендации о том, где размещать комментарии

- При объявлении переменной, даже если для нее используется хорошее имя, могут возникнуть сомнения относительно информации, которую она хранит, или того, как она используется.
- Практически в определении всех методов указать, что они делают, их характеристики и параметры, которые они получают, и результат, который они возвращают
- Не слишком короткие внутренние методы, чтобы указать, что делает каждая часть метода.

Это правда, что всегда сложно определить в коде, что очевидно, а что нужно прокомментировать, особенно вам, вероятно, не имеющему большого опыта программирования и работы с чужим кодом. Поэтому желательно, чтобы «на всякий случай» вы комментировали свой код как можно больше.

Еще один аспект, который следует учитывать, заключается в том, что комментарии представляют собой текстовые фрагменты на естественном языке, в которых вы должны выражаться как можно четче, заботясь об орфографии, грамматике, связности и других важных элементах, необходимых для правильного написания.

Все эти элементы важны для качества вашего кода, который вы создаете на протяжении всей своей карьеры, но они также будут иметь значительный вес при оценке вашего проекта по программированию.