Практическая работа № 6. Построение разветвлённых и циклических алгоритмов. Контрольная работа.

**Цель:** сформировать у студентов навыки построения алгоритмов разветвленных и циклических структур с помощью учебной среды КУМИР.

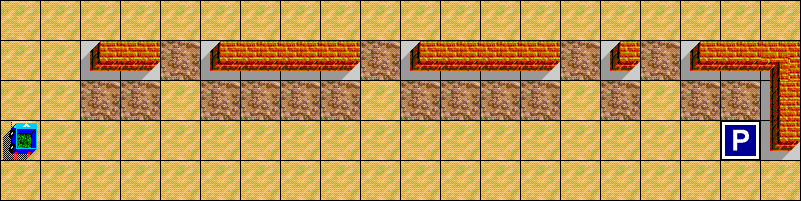
Ход работы:

1. Ознакомиться с теоретическим материалом.
2. Выполнить задания для закрепления изученного материала.

**Методические указания по выполнению работы.**

1. **Теоретические сведения:** Исполнитель
2. Условный оператор
   * 1. Что такое условный оператор (задача z5-3.maz)?

Рассмотрим новую задачу для Робота. Надо посадить цветы во всех клетках вдоль стены, где нет прохода, а если в этом месте есть проход, войти в него и обработать грядку между стенок. Предполагаем, что длина стены и число проходов неизвестны.



Для решения этой задачи нам надо научить Робота выполнять разные действия в зависимости от окружающей обстановки. Это можно изобразить на схеме

есть проход?

да

нет

влево; вперед(1);

посади;

назад(1); вправо;

посади;

Словами это можно сформулировать так: если есть проход (условие ***есть проход*** выполняется), то выполни одну группу команд, если нет – выполни другие команды. В программе для этой цели используется специальный *условный оператор* ***если***

Выбор

{

вперед(1); направо; вперед(1); **/\* подойти к началу стены \*/**

пока ( впереди\_свободно )

{

вперед(1);

если ( слева\_свободно )

{ **/\* войти в проход \*/**

налево; вперед(1);

посади;

назад(1); направо;

}

иначе

{ посади; }

направо;

вперед(1);

}

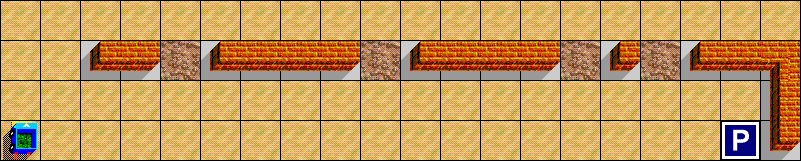
}

Таким образом, мы определили два варианта действий Робота - первый работает тогда, когда обнаружен проход, а второй – когда справа стена.

* + 1. Правила использования условного оператора

1. Условный оператор состоит из двух частей; первая часть начинается ключевым словом ***если*** или ***if*** (от английского “*если*”), после которого в скобках записывается условие.
2. Если это условие верно (или *истинно*), то выполняется группа команд, стоящая ниже в фигурных скобках (*блок-если*).
3. Вторая часть (*блок-иначе*) начинается со слова ***иначе*** или ***else*** (от английского “*иначе*”) и выполняется в том случае, когда условие в скобках *ложно.*
4. Нельзя отделять *блок-если*и *блок-иначе****,*** поскольку они составляют единый оператор.
5. Условие ставится только в заголовке *блока-если****.***
6. *Блок-иначе* может отсутствовать, если он не нужен; в этом случае мы говорим, что условный оператор записан в сокращенной форме.
7. Чтобы было удобнее разбираться в программе, используют отступы так же, как и в циклах: тело *блока-если*и *блока-иначе*сдвигается вправо на 2-3 символа.
   * 1. Сокращенная форма

Немного изменим задачу – пусть теперь Роботу надо обрабатывать только по 1 клетке в начале каждого прохода.



Таким образом, в *блоке-иначе* не осталось ни одной команды – если прохода нет, ничего делать не надо. Поэтому можно использовать сокращенную форму условного оператора – без второй части:

Сад2

{

вперед ( 1 ); направо; вперед ( 1 );

пока ( впереди\_свободно )

{

вперед(1);

если ( слева\_свободно )

{

налево; вперед(1);

посади;

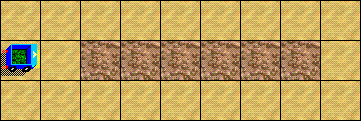
назад(1); направо;

}

}

1. Циклы
   * 1. Что такое цикл (задача z2-3.maz)?

Часто исполнителю надо выполнить какую-то последовательность команд несколько раз. Например, в задаче на рисунке Робот должен подойти к ряду клеток, которые надо закрасить, и затем выполнить 6 раз команды ***вперед(1)*** и ***посади***.



В данном случае эти команды надо повторить только 6 раза и можно легко 6 раз написать одинаковые команды. Но представьте, что надо сделать одинаковые операции 100 или 200 раз! В программировании в таких случаях используется специальная команда (*оператор цикла*), которая говорит исполнителю, что какую-то часть программы надо сделать несколько раз.

1. **Цикл** — это многократное повторение одинаковых действий
2. **Тело цикла ­**– это команды, которые выполняются несколько раз.
3. **Шаг цикла ­**– это однократное выполнение тела цикла.

Для нашей задачи подходит цикл ***повтори*** (или ***repeat***), в котором с известным числом шагов. Программа с использованием оператора цикла выглядит так:

Ряд

{

вперед ( 1 ); /\* подойти к месту работы \*/

повтори ( 6 )

{

вперед ( 1 );

посади;

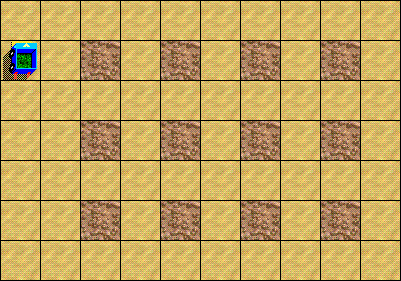
}

}

* + 1. Правила использования оператора цикла

1. Цикл ***повтори*** (или ***repeat***) используется тогда, когда число шагов заранее известно или может быть вычислено.
2. Оператор цикла начинается заголовком цикла – ключевым словом ***повтори***, за которым в скобках указывается нужное количество шагов.
3. Тело цикла начинается открывающей фигурной скобкой **{**и заканчивается закрывающей **}.**
4. Если тело цикла включает всего один оператор, скобки можно не ставить.
5. Для того, чтобы легче разбираться в программе, применяют специальную **систему записи с отступами**: тело цикла смещают вправо на 2-3 символа — это позволяет сразу видеть, где начинается и где заканчивается цикл. Для того, чтобы компьютер автоматически сделал отступы в программе, можно нажать клавишу **F6**.
   * 1. Вложенные циклы (задача z3-3.maz)

Рассмотрим задачу для Робота, в которой требуется сажать цветы во многих местах площадки (на рисунке справа).

Как бы такая задача решалась в реальных условиях? Можно предложить такой вариант: Робот сначала сажает цветы в первом (верхнем) ряду, затем во втором и т.д.

Для обработки одного ряда можно использовать цикл ***повтори(4)***. В программе надо обработать 3 ряда, то есть написать три одинаковых цикла. Тогда получается, что можно снова использовать цикл ***повтори(3)*** для трех рядов, но внутри него также будет находиться цикл.

1. **Вложенный цикл** – это такой цикл, который находится внутри другого цикла.

Ниже даны два возможных решения этой задачи. Они показывают, что внутренний и внешний циклы можно переставлять, если порядок обработки грядок безразличен. Главное – перевести Робота в нужную клетку и в нужное положение перед тем, как начнется следующий цикл. Кроме того, нельзя забывать, что Робот не может ходить по клумбам.

Способ1

{

направо;

повтори ( 3

{

направо;

вперед ( 2 );

налево;

назад ( 8 );

}

}

повтори ( 4 )

{

вперед ( 2 );

посади;

}

Способ2

{

направо; вперед(2);

направо;

повтори ( 4 )

{

налево; вперед(2);

направо;

назад ( 6 );

}

}

повтори ( 3 )

{

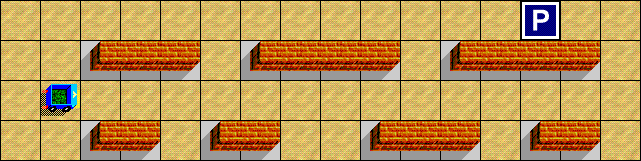
посади;

вперед ( 2 );

}

Что такое сложные условия (задача z6-3.maz)?

Рассмотрим еще одну задачу для Робота. Ему нужно пройти через коридор с проходами и прийти на Базу. Сложность состоит в том, что в обеих стенках есть проходы, сколько их – неизвестно.



Мы замечаем, что внутри коридора нет такой клетки, у которой слева и справа – свободные клетки. Значит, Роботу надо остановиться, когда слева и справа – свободно, это означает конец коридора. Теперь можно сформулировать алгоритм прохода через весь коридор на русском языке – иди вперед, пока слева стена **ИЛИ** справа стена.

В этом словесном алгоритмах мы объединяли логические команды Робота с помощью операции **ИЛИ,** получив из двух простых условий одно *сложное условие*. То же самое можно делать и в программе:

Посадка

{

вперед(1);

пока ( слева\_стена или справа\_стена )

вперед (1);

налево;

вперед(2);

налево;

пока ( не база ) вперед ( 1 );

}

1. **Сложное условие –** это условие, состоящее из простых условий и **логических операций**:

**НЕ** отрицание

**И** логическое умножение

**ИЛИ** логическое сложение

* + 1. Правила использования сложных условий

1. Простейшими условиями являются логические команды исполнителей (например, ***слева\_стена***) и **логические отношения** между значениями[[1]](#footnote-1).

**> <** больше, меньше ***т > 5, 2+n < x***

**>=** больше или равно ***a >= 2\* x+5***

**<=** меньше или равно ***c+2\*d <= 5\*v***

**==** равно ***d = = 2+c***

**<>** не равно ***a != b***

1. В условии “равно” ставится два знака равенства; чтобы не запутаться, надо запомнить, что если переменная изменяется (*оператор присваивания*), то надо ставить один знак “=“, а если не меняется (*логическое отношение*), то два.
2. Сложные условия составляются из нескольких простых; простые условия объединяются с помощью **логических операций.**
3. Операция **"И"** требует *одновременного* выполнения двух условий, например:

сверху\_стена **И** снизу\_стена

1. Операция **"ИЛИ"** обозначается требует выполнения *хотя бы одного* из двух условий (или обоих вместе), например:

сверху\_стена **ИЛИ** снизу\_стена

1. Иногда удобно использовать логическую операцию **“НЕ”**, которая отрицает значение логического выражения, например условия

***a < b*** и ***НЕ (b >= a)***

означают одно и то же.

1. Устанавливается такой **приоритет** (старшинство) логических отношений и операций:

• сначала выполняются операции в скобках, затем ...

• операции **“НЕ”**, затем ...

• логические отношения (>, <, >=, <=, ==, !=), затем ...

• операции **“И”**и в последнюю очередь

• операции **“ИЛИ”**.

Для изменения порядка выполнения операций используются скобки.

* + 1. Цикл с условием

Подход

{

направо;

}

пока ( впереди\_свободно )

{

вперед ( 1 );

}

пока ( не база )

вперед ( 1 );

Мы знаем, что многократное выполнение группы команд называется *циклом*. Однако здесь мы не можем применить цикл ***повтори***, так как число шагов заранее неизвестно – оно определяется во время работы программы.

Тем не менее, есть четкое условие, по которому Робот должен закончить работу: если перед ним оказывается стена. Таким образом, Робот должен выполнять цикл ***пока впереди свободно.*** Для этой цели служит специальный вид цикла – цикл ***пока*** (или ***while***,от английского *while* – пока)***.*** Такой вид цикла называется *циклом с условием*, поскольку он заканчивается, когда нарушается условие в заголовке цикла.

Для того, чтобы придти на Базу, в программе используется цикл ***пока не база***. Это условие истинно (верно), если Робот еще на пришел на Базу и надо двигаться дальше. Если Робот вступил в клетку, где находится База, условие ***база*** стало истинным, а условие ***не база*** – ложным, поэтому цикл закончится.

* + 1. Правила использования цикла *пока*

1. Цикл ***пока*** используется тогда, когда число повторений цикла заранее неизвестно, но ограничено каким-то условием.
2. Оператор цикла начинается заголовком цикла – ключевым словом ***пока***, за которым в скобках указывается **логическая команда** – условие, при котором **выполняется** цикл.
3. Если условие перестает быть верным (**истинным**), выполнение цикла заканчивается и исполнитель переходит к следующей команде.
4. Условие проверяется**в начале цикла**, то есть если перед выполнением цикла условие **ложно**, то цикл не выполнится **ни разу**.
5. В цикле выполняются все операторы, заключенные в фигурные скобки;

Если тело цикла включает всего один оператор, скобки можно не ставить.

1. Для того, чтобы легче разбираться в программе, все команды, входящие в цикл, смещают вправо на 2-3 символа – это позволяет сразу видеть, где начинается и где заканчивается цикл.

***Пример 2.***  При такой программе в той же задаче, что и в примере 1, Робот не будет ничего делать, так как сейчас справа от него нет стенки, и условие ***справа\_стена*** не выполняется.

Ничего

{

пока ( справа\_стена )

вперед ( 1 );

}

Важно помнить, что условие **не**проверяется внутри цикла, то есть датчик срабатывает только тогда, когда выполняется команда в заголовке цикла.

***Пример 3***. В этом примере программа для Робота составлена так, что он врежется в стенку и сообщит об ошибке “**НЕ МОГУ**”.

|  |  |
| --- | --- |
|  | Диверсия  {  пока ( впереди\_свободно )  {  вперед ( 2 );  }  } |

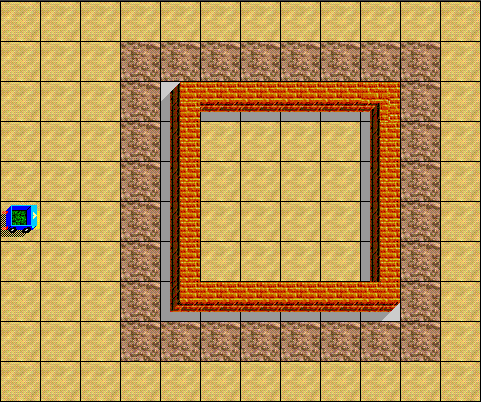
С циклом ***пока*** связано одна из самых неприятных ошибок программистов – **зацикливание**. Оно происходит в тех случаях, когда условие в заголовке цикла ***пока*** никогда не становится ложным.

***Пример 4.*** Эта программа приводит к зацикливанию, так как условие ***справа\_стена*** выполняется всегда и Робот не меняет своего места.

|  |  |
| --- | --- |
|  | Зацикливание  {  пока ( справа\_стена )  {  кругом; кругом;  }  } |

Использование цикла ***пока*** позволяет нам решать задачи, в которых некоторые данные (например, длина стенок) заранее неизвестны.

***Пример 5.*** Посадить цветы во всех клетках по периметру прямоугольной стены, считая, что расстояние до нее и ее размеры неизвестны.



Для решения этой задачи надо использовать несколько циклов с условием. Сначала Роботу надо дойти до стенки, затем перейти к углу. Дальше он пойдет «держась за стенку», обходя таким образом прямоугольник и сажая цветы во всех нужных клетках.

Поскольку при обработке каждой из 4-х стенок Роботу надо выполнять одинаковые команды, здесь можно использовать цикл ***повтори*(*4*)**. Тогда цикл ***пока*** становится вложенным циклом.

Контур

{

пока ( впереди\_свободно )

вперед(1); **/\* подойти к стене \*/**

налево;

пока ( справа\_стена )

назад(1); **/\* в левый нижний угол \*/**

повтори ( 4 )

{

вперед (1); **/\* теперь справа стена \*/**

посади; **/\* угловая клетка \*/**

направо;

}

пока ( справа\_стена )

{

посади;

вперед(1);

}

}

1. **Задания для самостоятельной работы:**

Необходимо записать с помощью разветвленных и циклических алгоритмов следующие задачи для движения робота в учебной среде программирования КУМИР исполнитель Робот.

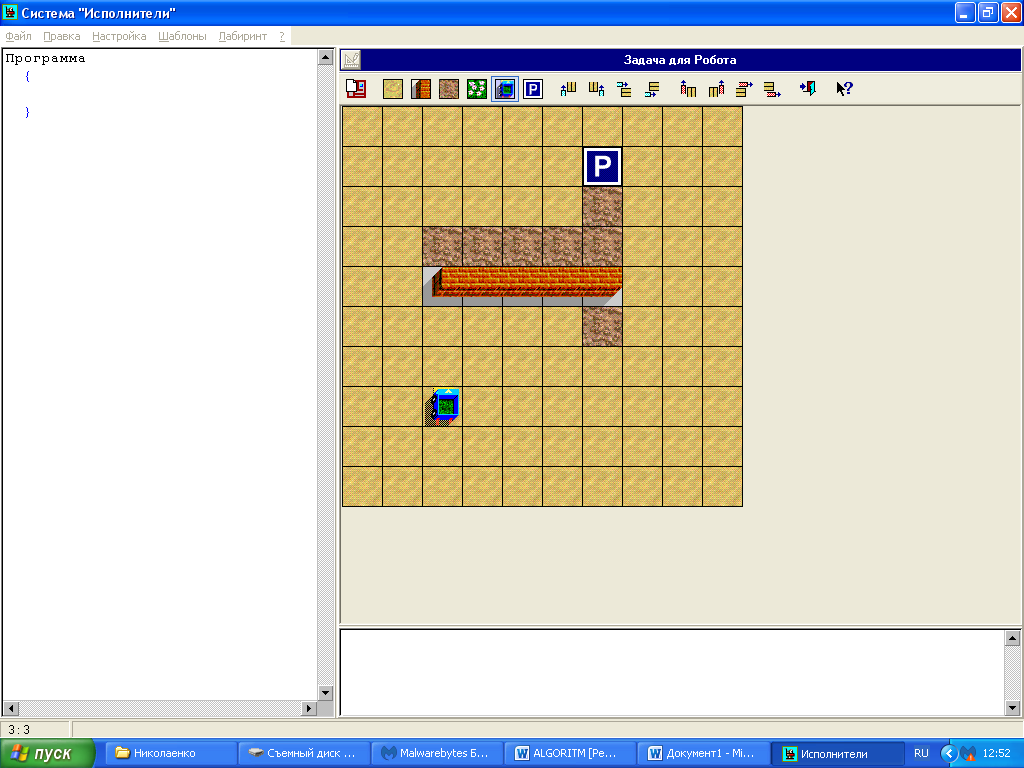
Задание №1

В среде КУМИР «Робот» написать такой алгоритм движения для робота, чтобы он посадил 3-ри грядки и приехал в конце на базу:



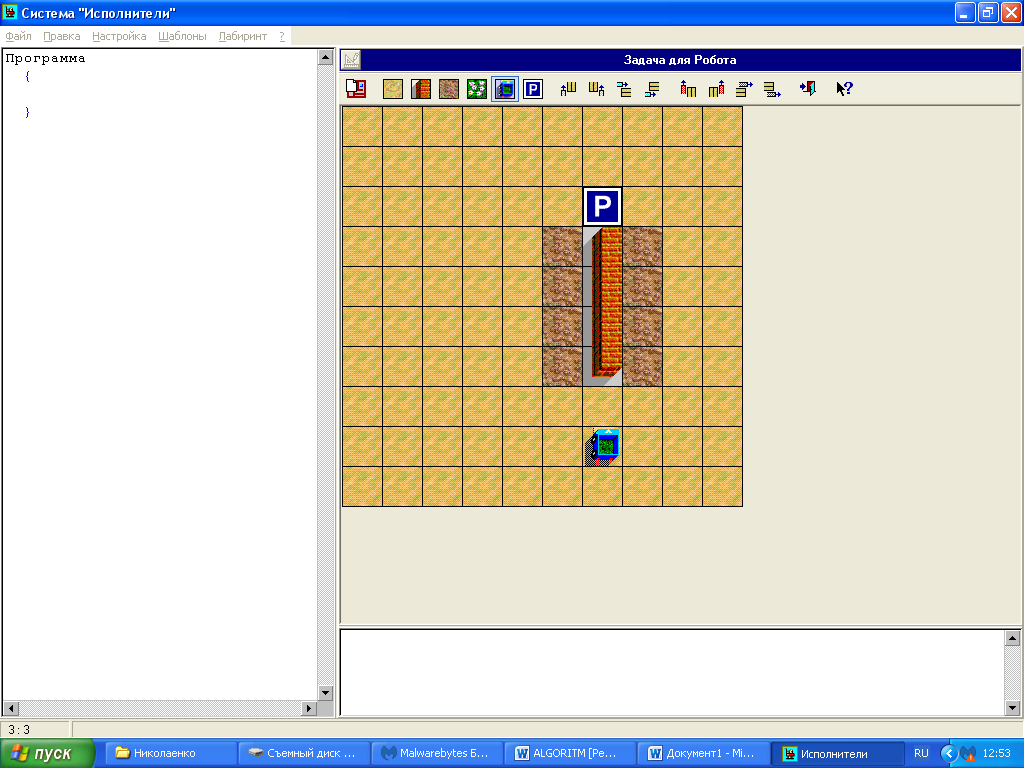
Задание №2

В среде КУМИР «Робот» написать такой алгоритм движения для робота, чтобы он посадил все грядки и приехал в конце на базу:



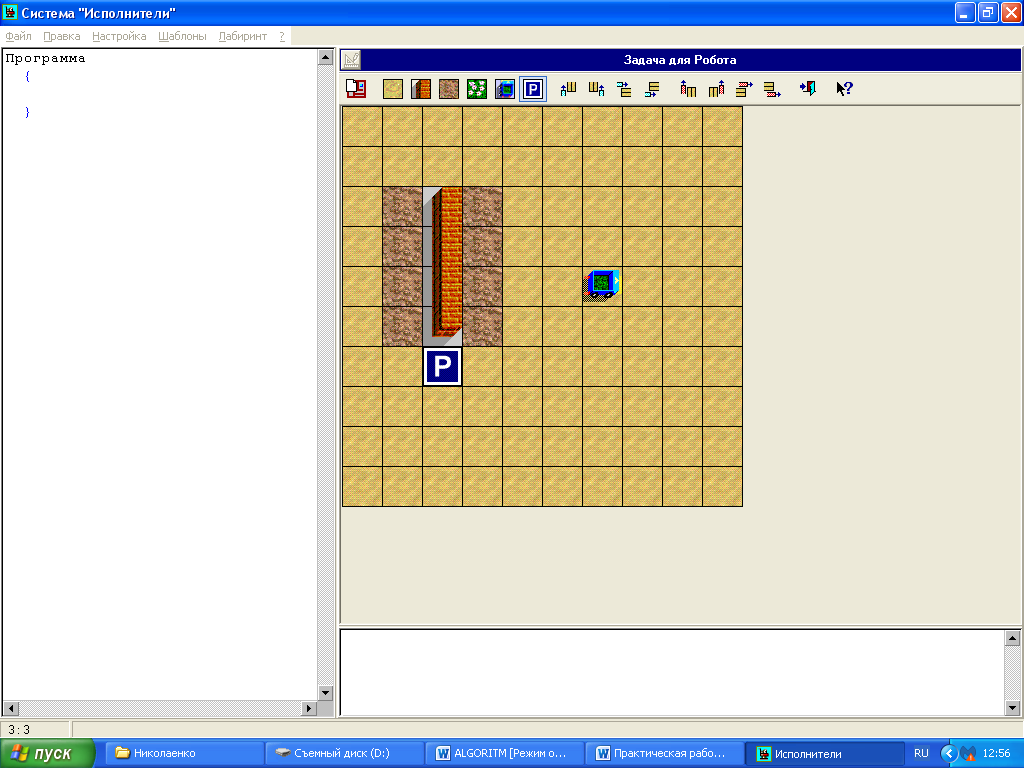
Задание №3

В среде КУМИР «Робот» написать такой алгоритм движения для робота, чтобы он посадил все грядки и приехал в конце на базу:



Задание №4

В среде КУМИР «Робот» написать такой алгоритм движения для робота, чтобы он посадил все грядки и приехал в конце на базу:



Задание №5

В среде КУМИР «Робот» написать такой алгоритм движения для робота, чтобы он посадил все грядки и приехал в конце на базу:



Все алгоритмы движения должны содержать ветвление (условные конструкции) и циклические!

1. Разговор о числовых выражениях целесообразен только после изучения переменных. [↑](#footnote-ref-1)