

§ 1

Назначение программной среды Robot

ROBOT - одна из программ серии ALGO, которая разработана не для профессионального программирования, а для обучения программированию в первую очередь школьников младших и средних классов.

Идея использования робота для изучения основ программирования используется очень давно. Можно привести несколько десятков программ и учебников, в которых реализован такой подход.

Это не детская игрушка, а солидная среда, с помощью которой можно решать серьезные задачи, ставить и решать, с помощью робота широкий диапазон практических, творческих проблем конструирования и моделирования.

От простых линейных программ к алгоритмам идентификации положения робота, программная реализация которых требует несколько дней работы.

Но постановка задачи во всех случаях остается простой и понятной для учеников, независимо от уровня знаний математики и других предметов. Как раз при программировании поведения робота очевидным становится факт, что машина не думает, а лишь выполняет запрограммированные человеком действия.

Описанная здесь программная реализация робота-грузчика имеет три существенные отличия от иных разработок этого класса.

1. Для программирования поведения робота использован формальный профессиональный язык Паскаль (Pascal), а не специально придуманный «детский» язык программирования. При переходе к

программированию в старших классах вас не нужно переучивать. Это экономит время и не вызывает путаницы между разнообразными синтаксисами записи алгоритмических конструкций. Это важно потому еще, что вы действительно дальше будете использовать при решении задач язык Pascal и его более высокого уровня визуальную среду Delphi.

2. Запись программ родным-русским языком выполнен как перевод зарезервированных слов языка Паскаль с английского языка на русский и наоборот. Это делает интуитивно понятными операторы Паскаля и не воспринимается как «игрушечный» язык программирования.
3. Робот выполнен как дистанционно управляемый грузчик на складе. Команды и функции заданны относительно робота, а не относительно рабочего поля. Это несущественно усложняет программирование, но поставленные задачи целиком отвечают настоящим техническим реализациям аналогичных устройств.

Соответствие к иным системам программирования.

В среде ALGO реализован язык программирования Pascal. Выбраны только те элементы языка, которые необходимы начинающим для усвоения основ программирования.

Вот некоторые основные отличия данного языка от языка среды Turbo Pascal – более мощной программной среды.

- 1) Не поддерживаются указатели, объекты, UNIT, USES.

- 2) Не реализованы множественные, перечислимые и диапазонные типы, а также тип **STRING**.
- 3) Файлы поддерживаются только текстовые.
- 4) Описание меток должно быть последним при объявлении, а метки задаются только целыми числами.

Но авторы данного программирования заявляют, что в следующих версиях ALGO будут поочередно сниматься эти ограничения, в конечном итоге данная программная среда будет иметь полный набор всех возможных команд, функций и процедур и решать практически весь спектр задач сегодня решаемых языком Turbo Pascal.

В качестве иллюстрации общего вида графического интерфейса среды Robot на *Рис. 1*, 2, 3 показаны все важные объекты, на которых будет сосредоточено внимание в дальнейшем, по мере изучения его возможностей.

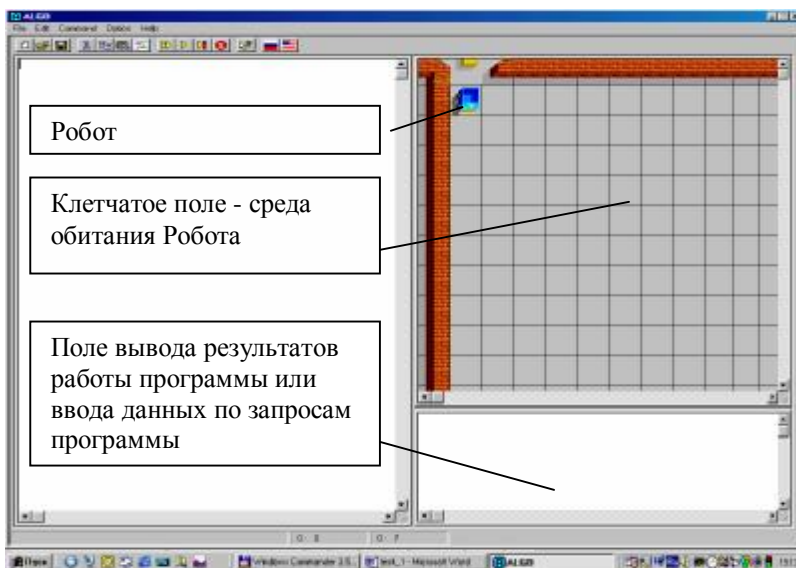


Рис. 1 Общий вид после запуска среды программирования «Robot»

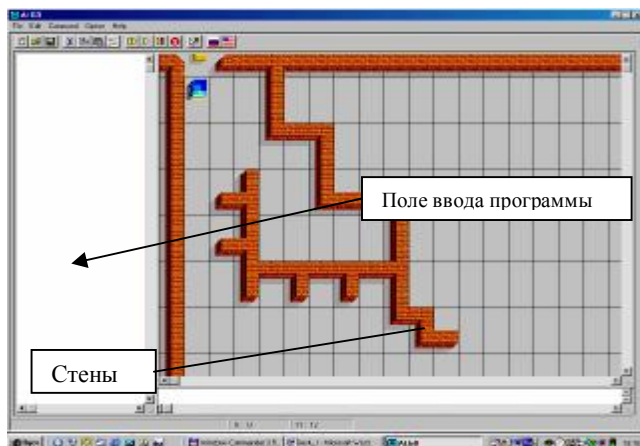


Рис. 2. Вид среды «Robot» с измененными размерами полей и добавленными стенами лабиринта

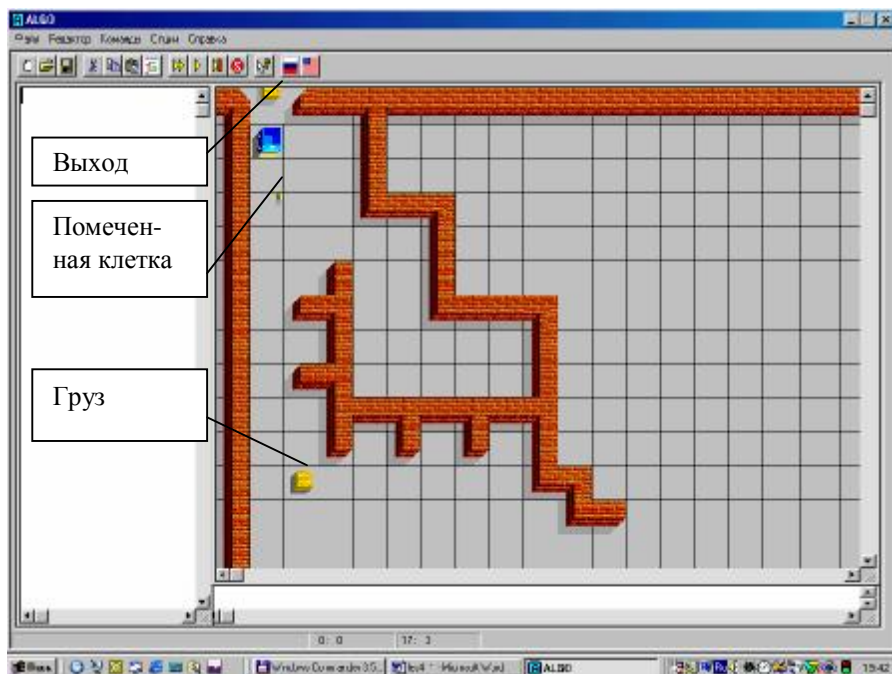


Рис. 3. Используемые объекты

§ 2

Робот, его возможности и управление им.

Робот-грузчик изображен в виде маленького грузового автомобиля, который перемещается только по клетчатому полю, называемому в нашей среде программирования – складом. Он может двигаться только по вертикали или по горизонтали. Шаг перемещения - одна клетка склада. При запуске программы он всегда устанавливается в одном и том же месте - верхний левый угол и направление вниз.

Робот может:

- перемещаться по командам **Вперед** и **Назад**;
- поворачиваться по командам **Налево** и **Направо**;
- опрашивать свободный путь по командам **СвободноВперед**, **СвободноПозади**, **СвободноСлева**, **СвободноСправа**;
- опрашивать обстановку впереди по команде **Впереди**; - работать с грузами по командам **Взять** и **Положить**;
- сообщить свое местонахождение по команде **Координаты** и направление движения по команде **Направление** – сообщить, **Пустой** ли робот.

Все эти команды сейчас вам не следует запоминать, с ними мы конкретно будем знакомиться при решении задач. Однако хочется заметить, что некоторые команды состоят из двух слов, не разделенных пробелом. Такой порядок языка команд и еще, чтобы было понятно назначение команды, применяют для вашего возраста вот такие длинные названия.

А теперь посмотрите на *Рис.4* и внимательно рассмотрите где, что и как располагается при запуске нашей среды программирования.

Надписи и линии указывают расположение всех объектов в окне среды исполнителя (клетчатом поле). Склад содержит 18 клеток по горизонтали и 13 клеток по вертикали. Чтобы просмотреть все поле склада воспользуйтесь

линейками прокрутки – вертикальной, чтобы увидеть границу склада внизу, горизонтальной, чтобы увидеть границу склада справа.

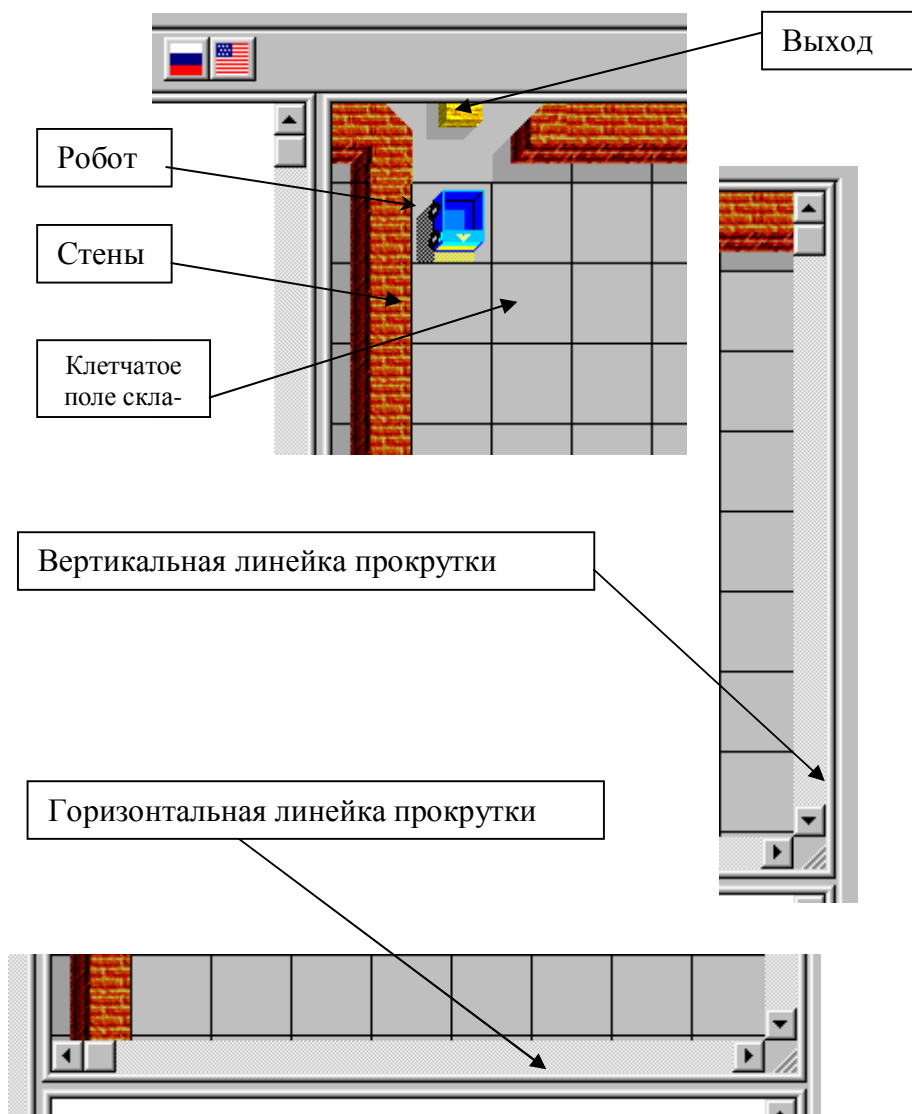


Рис. 4

Схема склада.

Для демонстрации использования программирования использована модель программированного робота-грузчика на складе. Модель выполнена таким образом, что при необходимости можно сделать настоящий макет с дистанционным управлением.

Склад разделен на клетки, по которым двигается робот. Клетки нумеруют сверху вниз и слева направо, как показано на рисунке. Есть четыре типа клеток: стена, свободная, груз и отмеченная. Изменять начальную конфигурацию склада можно нажатием один раз левой кнопки мышки в соответствующей клетке. На *Рис. 5* указана нумерация клеток и различные установки объектов

Чтобы пометить клетку следует на ней щелкнуть дважды, а установить груз – трижды. Четвертым щелчком с клетки удаляется все.

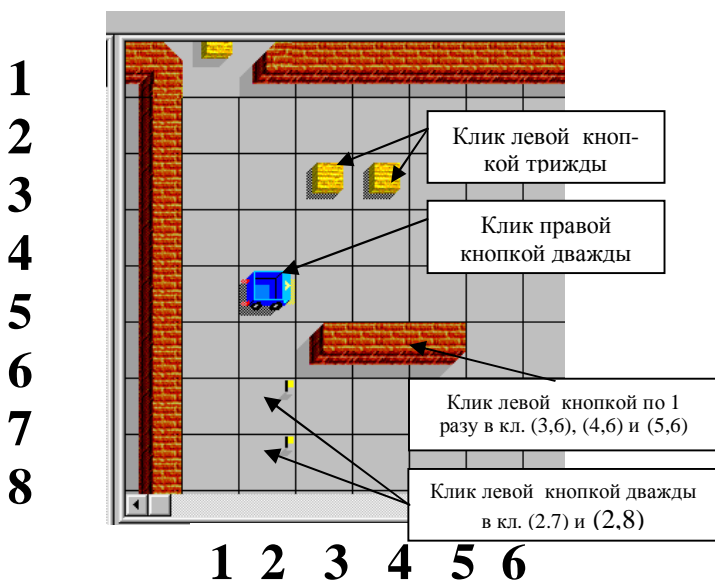


Рис. 5

Установить начальную позицию и направление робота можно нажатием правой кнопки мышки – один раз, установка робота в клетке, где стоял указатель мыши; второй щелчок в этой же клетке – робот повернется на угол 90^0 (прямой) против часовой стрелки; третий – еще на 90^0 и т.д..

Отдельный статус имеет клетка выход. Роботу запрещено заходить на эту клетку, но он может брать с нее и класть на нее неограниченное количество грузов. Это логично, так как робот обслуживает склад, на вход которого постоянно привозят и с него же забирают грузы.

Задание 1.

В качестве тренировки умений пользоваться мышкой для установки объектов на поле склада выполните все действия указанные на *Рис.5*. Последовательность объектов определите самостоятельно.

Задание 2.

Выполните следующий алгоритм:

- 1) Растяните склад по горизонтали на все окно приложения.
- 2) В клетках (4,1), (4,2), (4,3) и (4,4) – клетки считать в скобках первое число по горизонтали, второе по вертикали, установите стены.
- 3) Параллельно в столбцах 8, 12 и 16 клеток выполнить аналогичное построение.
- 4) В каждой полученной секции склада у горизонтальной стены установите по три груза.
- 5) Вход в каждую секцию слева и справа пометьте.
- 6) На пятой горизонтальной строке клеток постройте стену длиной в 15 клеток и сделайте поворот

вниз на 4 клетки, вправо на три и вверх на 4 клетки.

- 7) В позиции (18,5) установите робота в направлении справа налево по горизонтали.

Если все действия выполнены правильно, вы должны получить схему склада с расположенными всеми объектами как на *Рис. 6*

Задание для домашнего моделирования

В рабочих тетрадах придумайте оригинальную схему склада с изображением изученных объектов нашего приложения.

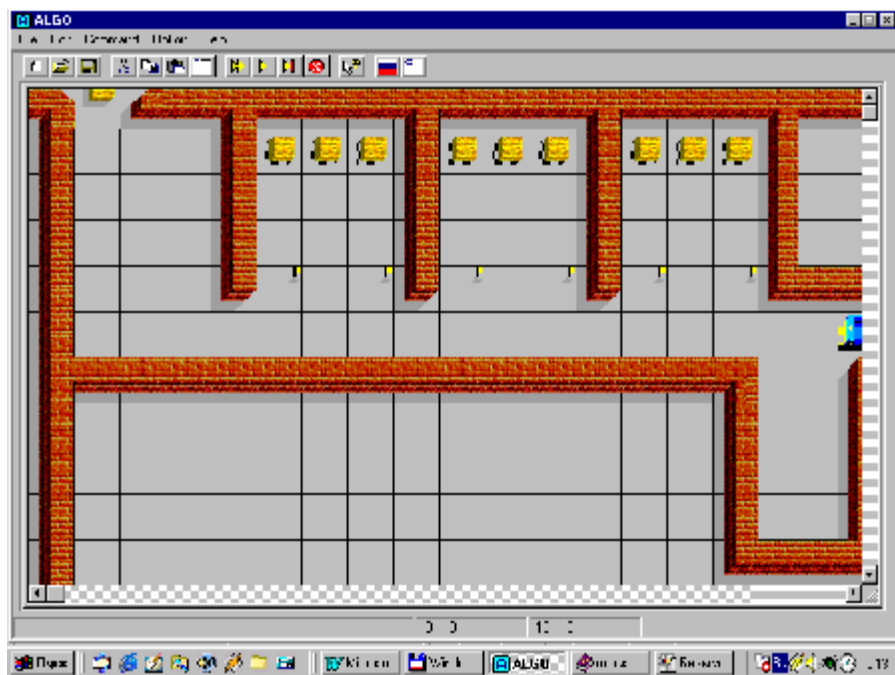


Рис.6

Структура языка программирования Algo

Пользование оболочкой

Программная среда Robot как и Algo организует общение с пользователем, т.е. с вами уважаемые учащиеся, используя нарисованное окно с несколькими полями, кнопками, надписями, которые и образуют так называемый интерфейс – диалоговый контакт. Произошло это название от двух слов: inter – между, face – лицо. На первом уроке на рисунках эти поля были указаны.



Для нас, начинающих учиться писать программы нужным будет поле для ввода текста программы. Оно расположено слева сверху и занимает довольно обширную часть окна программы. Вы заметили, что этот подпункт называется «Пользование оболочкой». Так вот оболочкой как раз и называется все это окно, посредством ее мы и будем общаться со средой нашего программного приложения.

Забегая немного вперед, следует отметить, что для того, чтобы выполнить программу, написанную языком Паскаль (Robot), необходима специальная встроенная программа - компилятор, который переведет написанную вами программу в специальный машинный язык - коды компьютера и только после этого ее выполняет. Если компилятор обнаружил ошибку в программе, он немедленно выводит сообщение и прекращает дальше работать, надо ошибку в программе исправить и заново запустить ее на выполнение. Как видите нажатием кнопки «Исполнение программы»



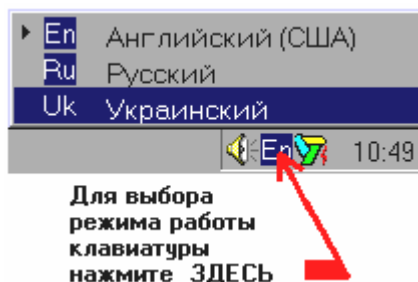
запускается первым компилятор и лишь, если в программе нет ошибок, она выполняется. Но для этого нужен текстовый файл с программой на языке Robot. Чтобы под-

готовить такой текст и нужно воспользоваться экранным редактором – полем, о котором только что шла речь.

Оболочка выполнена двумя языками - английским и русским. Для перехода на английский язык нужно нажать кнопку , а на русский - кнопку . Это же можно сделать с клавиатуры нажатием Ctrl+E и Ctrl+U соответственно, или через меню (Опции Английский язык; Опции Русский язык)

Для перехода к меню указывают мышкой на соответствующий его пункт или нажимают клавишу Alt, выбирают нужный пункт клавишами управления курсором, после чего нажимают Enter.

Язык оболочки никак не связан с режимом работы клавиатуры. Для переключения клавиатуры с латинских букв на русские и, наоборот, в большинстве случаев используют правый Ctrl+Shift. Это нужно выяснять в преподавателя или собственника компьютера. Но всегда можно переключить режим работы клавиатуры мышкой с помощью иконки в правом нижнем углу экрана, как показано на рисунке.



Структурный язык программирования

Одним из первых языков, которые начали использовать в школе после введения предмета Информатика в учебный план, был язык Basic. Это довольно простая среда

программирования, не требовала никаких заранее установок и объявлений, никаких ограничений использования значений, хотя конечно были и свои правила. Длинные программы на языке Basic читать было очень трудно, так как они порой были настолько запутаны переходами и возвратами, что не опытному программисту, а тем более, начинающему или ученику это было не под силу.

Для упорядочивания чтения текстов программ, обращения к отдельным ее частям и были созданы языки, примером которых и является ROBOT (ALGO, PASCAL). По-другому, была разработана определенная последовательность блоков, связанных с основной частью, закреплено за ними место следования – очередность, а также были введены обязательные служебные слова, без которых нельзя обойтись в работоспособной программе. Эти блоки и организовали структуру языка программирования, а сам язык такого строения стали называть структурным.

Обычный разговорный язык состоит из четырех основных элементов: символов, слов, словосочетаний и предложений. Возник разговорный язык из естественных потребностей человека общаться между собой. Разговорный язык имеет две формы: устную, состоящую из звуков, сочетания которых образуют слова, а слова – предложения и письменную – символьную или ее иногда называю текстовую, состоящую из набора букв, который называют алфавитом и знаков препинания. Разговорные языки в литературе названы естественными или национальными, каждый из которых отличается количеством и начертанием букв

ЯЗЫК программирования (его часто называют алгоритмическим языком) содержит подобные элементы, только слова называют элементарными конструкциями, словосочетания - выражениями, предложения - операторами. Символы, элементарные конструкции, выражения и опе-

раторы составляют иерархическую структуру, поскольку элементарные конструкции образуются из последовательности символов, выражения - это последовательность элементарных конструкций и символов, а оператор - последовательность выражений, элементарных конструкций и символов.

ОПИСАНИЕ ЯЗЫКА есть описание четырех названных элементов. Описание символов заключается в перечислении допустимых символов языка. Под описанием элементарных конструкций понимают правила их образования. Описание выражений- это правила образования любых выражений, имеющих смысл в данном языке. Описание операторов состоит из рассмотрения всех типов операторов, допустимых в языке. Описание каждого элемента языка задается его **СИНТАКСИСОМ** и **СЕМАНТИКОЙ**. Синтаксические определения устанавливают правила построения элементов языка. Семантика определяет смысл и правила использования тех элементов языка, для которых были даны синтаксические определения.

СИМВОЛЫ языка - это основные неделимые знаки, в терминах которых пишутся все тексты на языке.

ЭЛЕМЕНТАРНЫЕ КОНСТРУКЦИИ -это минимальные единицы языка, имеющие самостоятельный смысл. Они образуются из основных символов языка.

ВЫРАЖЕНИЕ в алгоритмическом языке состоит из элементарных конструкций и символов, оно задает правило вычисления некоторого значения.

ОПЕРАТОР задает полное описание некоторого действия, которое необходимо выполнить. Для описания сложного действия может потребоваться группа операторов. В этом случае операторы объединяются в **СОСТАВНОЙ ОПЕРАТОР** или **БЛОК**.

В общем алгоритм структуры текста программы таков:

1. Заголовок
2. Объявления:
 - переменных;
 - процедур и функций;
 - меток.
3. Начало
4. Тело программы
5. Конец

Заголовок программы

В своей простейшей форме программа Robot (Algo) состоит из заголовка программы, который именует программу, и основного программного блока, выполняющего назначение программы. В основном программном блоке находится секция кода, заключенная между ключевыми словами `begin` и `end`. Приведем простейшую программу, иллюстрирующую эти принципы:

```
program Привет;  
  begin  
    Writeln('Добро пожаловать в среду програм-  
мирования Algo-Robot');  
  end.
```

Первая строка - это заголовок программы, который именует данную программу. Остальная часть программы - это исходный код, который начинается ключевым словом `begin` и заканчивается `end`.

На русском языке это будет выглядеть так:

**программа Привет;
начало
 вывестистр('Добро пожаловать в среду про-
граммирования Algo-Robot');
конец.**

Хотя данная конкретная программа содержит только одну строку, их может быть много. В любой программе Pascal, в том числе и Algo, все действия выполняются между begin и end.

Код между последними операторами begin и end программы управляет логикой программы. В очень простой программе в этой секции кода может содержаться все, что вам нужно. В более крупных и сложных программах размещение в этой секции всего программного кода может затруднить чтение и понимание программы. К тому же ее будет труднее разрабатывать.

Объявления

Объявление переменных, процедуры и функции, метки позволяют разделить логику программы на более мелкие и управляемые фрагменты и аналогичны подпрограммам в других языках, например, в Basic. Как и в основном блоке программы, все действия в процедурах и функциях заключаются в begin и end, после последнего только нужно ставить точку с запятой. Каждый из этих сегментов кода выполняет конкретную задачу.

✓ Переменные величины, имена

Переменными величинами называют числа, слова, целые тексты, которые в ходе выполнения программы могут менять свои значения. Обозначаются переменные бук-

вами латинского алфавита. Имя переменной (ее часто называют идентификатором) может иметь любую длину, однако только первые его 16 символов являются значимыми.

Идентификатор должен начинаться с буквы и не может содержать пробелов. После первого символа идентификатора можно использовать буквы, цифры и символы подчеркивания. Как и в зарезервированных словах, в идентификаторах можно использовать как строчные, так и прописные буквы. Имя не должно содержать пробелов. Не разрешается в языке Algo использовать в качестве имен служебные слова и стандартные имена, которыми названы стандартные константы, типы, процедуры, функции и файлы.

Примеры имен языка Algo:

A b12 r1m SIGMA gamma I80_86 ... G, alfa,
test17, x2y, Сума2Чисел, конец_массива.

В математике принято классифицировать переменные в соответствии с некоторыми важными характеристиками. Производится строгое разграничение между вещественными, целыми и логическими переменными, между переменными, представляющими отдельные значения и множество значений и так далее.

При обработке данных на ЭВМ такая классификация еще более важна. В любом алгоритмическом языке каждая переменная, выражение или функция бывают определенного типа. В языке Algo существует правило: тип явно задается в описании переменной, которое предшествует их использованию. Например:

Var i, f, : integer

На русском языке:

Переменные i, f, : целые;

Любая переменная наряду с именем обязательно должна иметь и значение. Запомните, пожалуйста, следующую схему:



При описании переменной, как указывалось раньше, необходимо указать ее тип. Тип переменной определяет набор значений, которые она может принимать, форму записи их в памяти и действия, которые могут быть над нею выполнены. Типы разделяются на простые и сложные. Переменная простого типа всегда имеет одно значение (число, символ и т.п.), переменная сложного типа имеет таблицу значений одинакового типа (массив) или набор полей различного типа (запись). К простым типам в ALGO относятся:

- логический тип `boolean` - логические;
- символьный тип `char` - символы;
- целый тип `integer` - целые;
- действительный тип `real` - действительные.

По мере изучения языка программирования мы постепенно познакомимся со всеми типами переменных.

✓ *Функции и процедуры*

О функциях и процедурах, как и о типах переменных, сейчас, на первых уроках, вести разговор нет необхо-

димости. К ним мы еще вернемся, когда будем изучать и решать задачи с этими данными.

▼ Метки

В этом разделе оглашаются метки, которые определяются операторам перехода. Каждая метка может использоваться не один раз, а количество их в программе зависит от конкретной задачи и надобности в них. Все метки перед употреблением должны быть объявлены с помощью такой записи:

**Label 1, 2,...N;
Метки 1, 2,...N;**

Метка есть последовательность цифр, то есть целое число без знака, которое находится в диапазоне от 0 до 9999. Нули перед числами игнорируются. Организация перехода организуется так:

```
program Привет;  
  label 1;  
  begin  
    1;;  
    Writeln('Добро пожаловать в среду про-  
граммирования Algo-Robot');  
    goto 1;  
  end.
```

На русском языке эта программа выглядеть будет так:

**программа Привет;
метки 1;**

начало
1;;
вывестистр('Добро пожаловать в среду про-
граммирования Algo-Robot');
перейти 1;
конец.

Если исполнить эту программа, то экран монитора будет заполняться выражением «Добро пожаловать в среду программирования Algo-Robot» до тех пор, пока не будет нажата кнопка Stop.

Начало, тело и конец программы

Начало состоит из одного служебного слова begin (начало), конец аналогично – из одного end. (конец.)

Все операторы, команды и т.д. записываются между этими двумя служебными словами, и называется этот блок телом программы.

§ 4

Движение вперед

Для того чтобы робот двигался вперед, нужно обратиться к процедуре вперед, что можно объявить следующим образом:

procedure Forward(x : integer);
процедура Вперед (x : целые);

где x - количество клеточек, на которое должен переместиться робот

Если количество клеточек не указано, то робот перемещается на одну клетку вперед. При этом скобки употреблять нельзя. Движение осуществляется в том направлении, в котором ориентирован робот в момент обращения к процедуре. Если на дороге встречается стена или груз, то выдается сообщение об ошибке.

В реальной программе эта процедура не объявляется, в ее тексте записывается только команда **Forward;** (**Вперед;**) или **Forward(x);** (**Вперед(x);**)

Движение назад

Для того, чтобы робот двигался назад, нужно обратиться к процедуре Назад, что объявляется следующим образом:

procedure Backward (x : integer);
процедура Назад (x : целые);

где x - количество клеточек, на которое должен переместиться робот.

Если количество клеточек не указано, то робот перемещается на одну клетку назад. При этом скобки употреблять нельзя. Движение осуществляется в направлении, противоположном тому, в котором ориентирован робот в момент обращения к процедуре. Если на дороге встречается стена или груз, то выдается сообщение об ошибке.

В реальной программе эта процедура не объявляется, в ее тексте записывается только команда **Backward;** (**Назад;**) или **Backward (x);** (**Назад (x);**)

Поворот направо и налево

Для того, чтобы робот повернулся направо или налево, нужно обратиться к процедуре Направо или Налево, что объявлена следующим образом:

procedure Right (x : integer);
процедура Направо (x : целые);
procedure Left (x : integer);
процедура Налево (x : целые);

Если аргумент не указан, то робот поворачивается на 90 градусов направо или налево. При этом скобки употреблять нельзя. Если указать аргументом целое положительное число, то поворот повторяется заданное число раз по 90 градусов.

Обратите внимание, что направление поворота задается относительно робота, а не относительно наблюдателя
Рис.7.



Рис .7

В реальной программе эти процедура также не объявляется, в ее тексте записывается только команды.

Редактирование текста программы

Для указания места, на которое записывается введенный с клавиатуры текст, используется курсор - мигающая вертикальная черточка. Введенный символ всегда записывается перед курсором, а остальной текст раздвигается на одну позицию.

Переход на новую строку осуществляется нажатием клавиши **Enter**. Клавиша пробел (длинная клавиша внизу клавиатуры) обрабатываются так же, как и остальные символы, только после ее нажатия остается свободное место. Ею можно выравнивать текст программы, сдвигая его вправо.

Курсор можно переместить с помощью клавиш управления курсором, вертикальной и горизонтальной полос прокрутки и нажатием левой кнопки мышки в соответствующей позиции. Курсор никогда не устанавливается вне текста. Например, когда Вы укажете позицию после строки, то курсор будет установлен на конец строки.

Вытереть неправильный символ перед курсором можно клавишей **BackSpace**, а после курсора - клавишей **Delete**.

При исправлении программ часто возникает необходимость вытереть или переставить в иное место большие фрагменты текста (блоки). Для этого


фрагменты текста (блоки). Для этого предусмотрены так называемые блочные операции. Чтобы выделить блок нужно поставить курсор в его начало и, держа нажатой клавишу **Shift**, переместить курсор до конца блока. Блок при этом будет выделен черным цветом. Это же можно сделать мышкой: нажать левую кнопку в начале блока и провести до конца, не отпуская кнопки. Выделение автоматически отменяется любой операцией перемещения курсора.

С выделенным блоком можно выполнять такие операции:

- удалить клавишей **Delete**;
- удалить и запомнить в специальном буфере нажа-



тием кнопки на панели инструментов или выбором пункта **Редактор Удалить** в меню, или нажатием **Shift+Delete** на клавиатуре;


- скопировать в буфер нажатием кнопки  на панели инструментов или выбором пункта **Редактор Скопировать** в меню, или нажатием **Ctrl+Insert** на клавиатуре;
- скопировать содержимое буфера в текст про-




граммы на место курсора нажатием кнопки на панели инструментов или выбором пункта **Редактор Вставить** в меню, или нажатием **Shift+Insert** на клавиатуре.

В буфере одновременно может находиться лишь один фрагмент текста, который хранится на протяжении всего времени работы программы. Это дает возможность переставлять фрагменты текста с одного файла в другой.

ALGO предусматривает возможность автоматического упорядочения текста программы (записи каждого оператора с новой строки, отступов перед вложенными

операторами и т.п.). Для выполнения этой операции нажмите кнопку  на панели инструментов или выберите пункт **Редактор Упорядочить** в меню.

Считывание и запись программ

Для того, чтобы считать (загрузить) программу с диска, нужно нажать кнопку  на панели инструментов или выбрать пункт **Файл Открыть** в меню, или нажать клавишу **F3**. На экране появится окно системного диалога, которое имеет такой вид *Рис.8*.

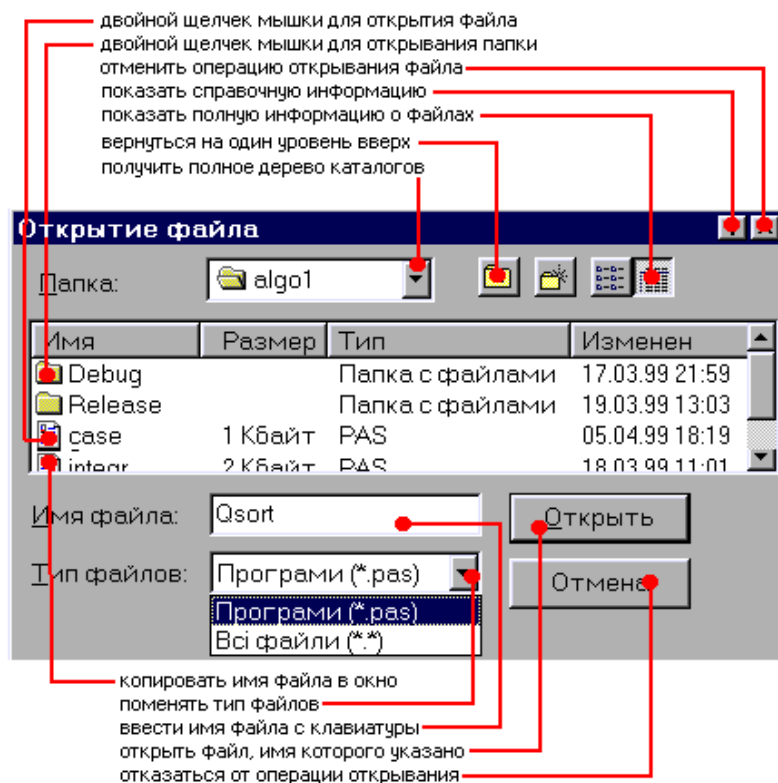





Рис.8


С помощью этого диалога Вы можете выбрать и прочитать любой доступный текстовый файл. Нельзя вводить имена файлов, которых нет.

Для того чтобы создать новый файл, нужно нажать кнопку  на панели инструментов или выбрать пункт **Файл Новый** в меню, или нажать клавишу **F4**.

Аналогичным способом записывают (сохраняют) файлы на диске. Для этого нужно нажать кнопку  на панели инструментов или выбрать пункт **Файл Сохранить** в меню, или нажать клавишу **F2**. Окно системного диалога имеет такой же вид. Когда Вы записываете новый файл, то обязательно надлежит ввести с клавиатуры его имя в соответствующем окне диалога. Если при этом будет указано расширение (.pas, .dat, .txt), то файл будет записан с этим расширением, иначе система автоматически запишет файл с расширением (.pas).

При необходимости записи файла в новую папку (подкаталог) можно создать ее с помощью этого же диалога, нажав кнопку .


Выполнение программы



Чтобы выполнить программу нужно нажать кнопку  на панели инструментов или выбрать пункт **Команды Выполнить** программу в меню, или нажать клавишу **F9**.

Как не досадно, но в большинстве случаев вместо ожидаемого результата выполнения получаем сообщения об ошибке в программе. ALGO выделяет красным цветом строку, в которой прекратилась компиляция и устанавливает курсор на месте остановки. Это не всегда означает, что как раз в этой строке допущена ошибка. Например, если

курсор установлен в начале выделенной строки, а в тексте сообщения есть слово "пропущен", то это в большинстве случаев означает, что пропущено что-то в предыдущей строке.

При выполнении программы зеленым цветом выделяется строка, которая выполняется, а в окне режима работы указан текущий режим.

Кнопкой  или клавишей F6 всегда можно прекратить выполнение программы.

Кнопками  и  или клавишами F8 и F7 можно выполнять программу пошагово и до курсора соответственно.

§ 5

Взять груз, положить груз

Перемещения по складу робота – это не просто хождение по клеткам, в первую очередь это работа и работа однообразная трудоемкая. В реальной жизни для облегчения изнурительного, тяжелого труда и устранения пагубного влияния его на здоровье и психику человек и придумал различные приспособления, автоматы, и, наконец, роботы. Естественно для этих машин не существует понятия усталости, психологической перегрузки от монотонного, однообразного, тяжелого труда.

Наша модель робота также способна выполнять работу. Она не требует ограничения веса, количества перевезенного груза, времени на совершения этой работы. Что программист в своей программе предусмотрит, то он и будет делать. Только условие одно – выполнять он может те команды, которые имеются в его системе команд (СКИ). Что же еще он может делать? Вот с очередными командами **Взять** и **Положить** мы сейчас и будем знакомиться.

Для того чтобы взять груз нужно вызывать процедуру

Procedure Take; Процедура Взять;

По команде **Take (Взять)** робот берет, как настоящий автопогрузчик, груз, что находится на клетке непосредственно перед ним. Если груза нет или робот уже загружен, то выдается сообщение об ошибке. Помните, робота нужно всегда ориентировать так, чтобы груз находился впереди его, т.е. загружается он не как в реальной жизни -

через задний борт, а через его переднюю – носовую часть и еще что важное, ему не нужен дополнительный погрузчик.

Для того, чтобы **Положить** груз нужно вызывать процедуру

Procedure Put;

Процедура Положить;

По команде **Положить** робот кладет, как настоящий автопогрузчик, груз, на клетку непосредственно перед ним. Если грузом робот не загружен или клетка перед ним не свободная, то выдается сообщение об ошибке. Если клетка, на которую кладут груз, отмеченная, то метка вытирается.

Как вы думаете, что будет, если на клетке, куда выгружается груз, уже есть груз? Ответ на этот вопрос сделайте сами в практической работе.

Несколько практических советов. Внимательно рассмотрите рисунки.

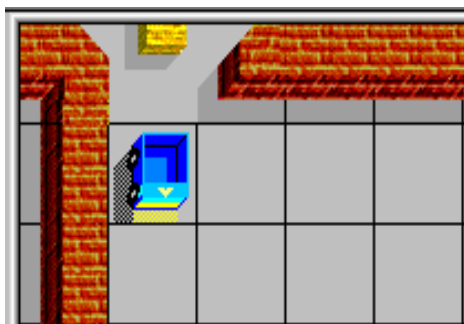


Рис. 11 В таком положении груз не
взять



Рис. 12 Груз взять можно

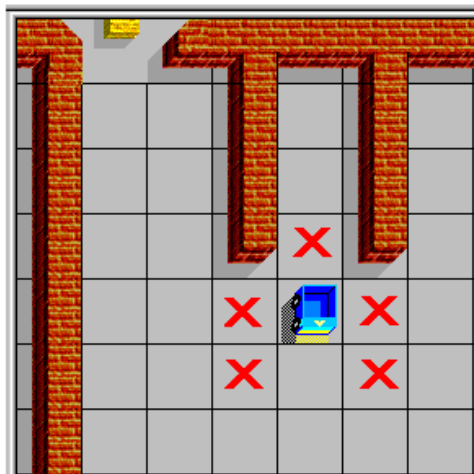


Рис. 13 Что означают крестики?

Задача 4_4

Используя изученные команды написать программу по Рис. 14.

Задача 4_5

Загрузите с сервера эту программу (имя на диске z4_5.pas), внимательно рассмотрите текст программы, если есть неточности или ошибки отредактируйте и проверьте ее работоспособность.

Program Задача4_5

Begin

Left(2);

Take

Right(2);

Forwar(5);

Put(2);

End.



Рис. 14

Вопросы:

1. Чем отличается загрузка (погрузка) модели робота от аналогичных видов работ в реальном мире?
2. Почему у команд **Take** и **Put** отсутствуют аргументы (числа)?
3. Что произойдет, если последовательно записать в программе команды **Put** и **Take**, а наоборот?

Творческое задание

В одной из секций размещено три груза. Требуется взять груз, который находится за двумя первыми. Как нужно поступить роботу в данной ситуации? Изобразить на странице тетради схему и написать фрагмент программы

§ 6

Пометить, стереть клетку

Во многих задачах возникает необходимость однажды пометить клетки. Например, при поиске выхода с лабиринта, в котором есть запертые дороги. Робот может пометить клетку, на которой он находится, маленьким флажком. Для этого нужно вызывать процедуру

**procedure Select;
процедура Пометить;**

Если клетка уже была помеченная, то команда игнорируется роботом, т.е. при движении помеченные клетки остаются помеченными, двойных меток не ставится. Некоторые замечания по клетке, с которой робот начинает движение (исходная ли это, или любая другая). Если клетка была помечена и робот начинает движение с нее, то метка удаляется, но такое бывает только тогда, когда вы завершили программу движения по меткам и запустили снова ее на выполнение. А если клетка помечена и робот начинает движение с нее, но с заданием пометить клетки на которых он был, то метка будет поставлена после съезда с нее.

Примеры 1 и 2. Показывают эти два случая.

Пример 1. (Рис.15)

**программа вперед;
начало
вперед;
вперед;**

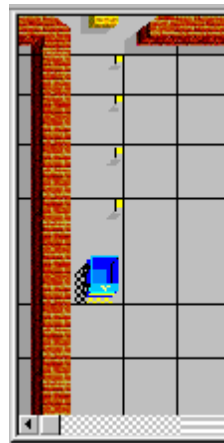


Рис. 15

вперед;

конец.

Пример 2. (Рис.15)

программа вперед_метить;

начало

пометить;

вперед;

пометить;

вперед;

пометить;

вперед;

пометить;

конец.

Для стирания метки нужно вызывать процедуру

Procedure Clear;

Процедура Стереть;

Если клетка не была помеченная, то команда игнорируется роботом.

При движении робота по клеткам с метками, метки распознаются внутренней командой *отмечен* и вытираются по команде **Clear (Стереть)**;

Пример (Рис. 16)

На русском языке

программа удаление;

начало

вперед;

стереть;

вперед;

стереть;

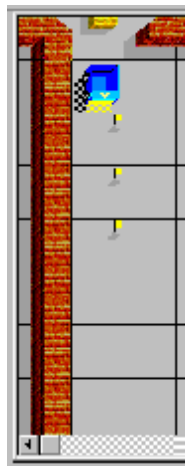


Рис. 16

вперед;
стереть;
конец.

На английском языке
program удаление;

begin
 forward;
 clear;
 forward;
 clear;
 forward;
 clear;
 end.

Задание 1.

С файлом z4_6.pas проведите испытания:

- 1) Загрузите файл и приведите форму записи в порядок.
- 2) Запустить программу и убедиться, что метки расставлены как предусмотрено программой.
- 3) Не меняя позиции робота снова выполните программу. Обратите внимание, что произошло с метками?
- 4) Не меняя позиции робота, удалите команды **Select**, приведите в порядок текст программы и снова запустите.
- 5) Если сделано все как сказано в задании, то первая метка должна быть удалена.

Задание 2.

В этой же программе замените команду **Select** на команду **Clear** и проведите несколько раз ее выполнение,

вручную устанавливая метки.

Измените движение робота на горизонтальное.
Подумайте, что для этого нужно сделать

Определение направления движения.

Все команды робота учитывают его ориентацию, т.е. направление передней части модели. Если возникает необходимость определить ориентацию робота относительно наблюдателя, нужно обратиться к функции **Direction (Направление)**, которая объявляется следующим образом:

Function Direction: integer;

Функция Направление: целые;

Как мы уже с вами договорились объявлять саму функцию не надо, она встроена в программную среду, в частности, чтобы узнать ориентацию робота относительно пользователя, достаточно использовать ее имя (идентификатор), или как мы их назвали, команду.

В зависимости от ориентации робота функция возвращает такое значение:

- 0 - робот направлен вниз (на юг);*
- 1 - робот направлен вправо (на восток);*
- 2 - робот направлен вверх (на север);*
- 3 - робот направлен влево (на запад);*

Результат выводится в третьем поле (ниже среды робота) в цифровом формате. Забегая вперед, для получения результатов в третьем окне использовать надо команду **WriteLn (ВывестиСтр)**, о которой еще будем говорить в дальнейшем.

Пример (Рис. 17):



Рис. 17

```

program Направления;
begin
  writeln(Direction);
end.

```

Опрашивание обстановки впереди.

В программах со сложной обстановкой: всевозможные лабиринты в складе, много расставленных грузов и меток, робот пользуется встроенной функцией **Впереди**. Чтобы выяснить какие параметры заложены в этой функции можно это показать на примере программы, которая выводит на экран действительную картину перед роботом относительно его передней части. Для детального опрашивания обстановки перед роботом нужно обратиться к функции **Впереди**, что объявляется следующим образом:

Function InFront: integer;
Функция Впереди: целые;

В зависимости от ситуации в направлении движения робота функция возвращает такое значение в поле вывода результатов числовых и текстовых операций, числа:

- 0 - перед роботом стена;
- 1 - перед роботом груз;
- 2 - перед роботом выход;
- N - перед роботом N свободных клеток;

В качестве примера может служить следующая программа:

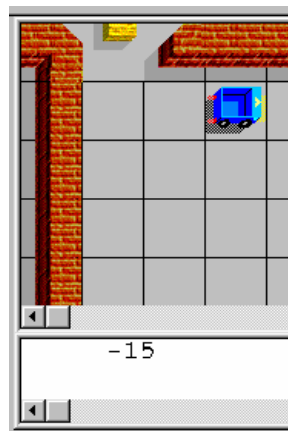


Рис. 18

Пример (Рис. 18)

```
program Обстановка;  
  begin  
    writeln(InFront);  
  end.
```

Задание

Напишите программу, которая будет выводить одновременно и обстановку перед роботом и его направление. Какой можно сделать вывод из этого, глядя на поле с результатами работы программы? Ситуация должна быть примерно такой Рис.19).

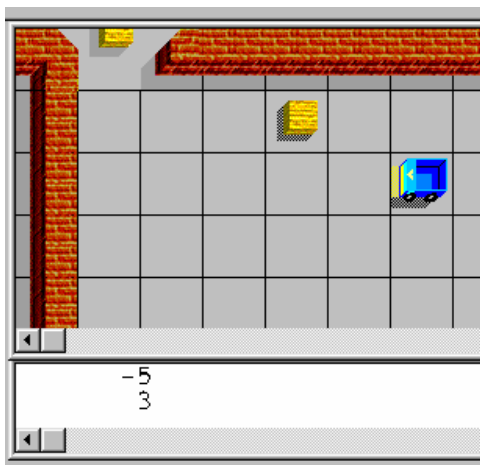


Рис. 19

Опрашивание текущих координат.

Координатами называются пара чисел (x,y) – пересечение нумерации строк и нумерации столбцов клеток. Как ранее уже упоминалось клеток по горизонтали 18, а по вертикали 13. Текущие координаты т.е. координаты клетки на которой стоит в настоящий момент робот, можно опреде-

лечь, вызвав процедуру **Координаты**, что объявляется следующим образом:

procedure Coordinates(var x, y : integer);
процедура Координаты(переменные x, y : целые);

После выполнения процедуры переменной x присваивается значение номера строки, а переменной y - номера столбца, на пересечении которых находится робот. Нумерация строк ведется слева направо, а нумерация столбцов - сверху вниз, как показано на (Рис.???)

Чтобы определить позицию робота на клетчатом его поле (складе), в качестве примера можно рассмотреть демонстрационную задачу demo2.pas.

Замечание:

- 1) Сама процедура не объявляется в программе;
- 2) Вводятся только переменные x и y в блоке объявлений;
- 3) В теле программы обязательно вводится идентификатор с параметрами;
- 4) Среда программирования не может вывести в одну строку эти координаты, поэтому используются две команды **ВывестиСтр(x)** и **ВывестиСтр(y)**.

На русском языке:

программа Местоположение;
переменные x,y:целые;

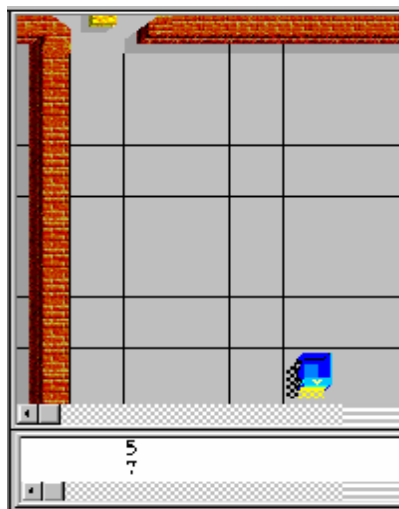


Рис. 20

```
начало  
    координаты(x,y);  
    вывестистр(x);  
    вывестистр(y);  
конец.
```

На английском языке:

```
program Местоположение;  
var x,y:integer;  
    begin  
    coordinates(x,y);  
    writeln(x);  
    writeln(y);  
end.
```

Задание

Откройте файл demo2.pas, установите робота как показано на (Рис.20). Щелчком правой клавиши мышки установите робота в другую позицию и выполните программу. Такое сделайте несколько раз. Вы должны заметить, что координаты положения робота каждый раз точно определяются так, как было описано выше.

Опрашивание свободной дороги.

При перемещении в неизвестной обстановке робот должен контролировать с помощью сенсоров, свободна ли клетка в заданном направлении. Иначе он может разбиться о стену или расчавить груз. Сенсоры активизируются при вызове логических функций

СвободноВперед, СвободноПозади, СвободноСле-

ва, СвободноСправа;

FreeForw, FreeBack, FreeLeft, FreeRight.

Эти функции не имеют параметров и возвращают истинное значение (true), если клетка в соответствующем направлении свободная или отмеченная, и ложное значение(false), если в соответствующем направлении груз или стена.

Обратите внимание, что направление опрашивания задается относительно робота, а не относительно наблюдателя.

Пример:

в результате выполнения фрагмента программы

```
while FreeForward do
```

```
    Forward ;
```

робот будет двигаться вперед, пока не встретит препятствие (груз или стену).

Творческое задание

Написать программу, которая могла бы при совершении некоторых передвижений робота сообщить о параметрах опроса обстановки и местоположения его на складе.

Пример: (ideya4_1.pas)

§ 8

Выполняемые операторы (команды)

Программа - это некоторый алгоритм преобразования и пересылки информации, записанный конкретным языком программирования. В отличие от описательного предоставления алгоритма, которое предусматривает чтение его человеком, в программе каждый шаг алгоритма должен быть записан в форме одной из немногих инструкций, которые компилятор способен переписать в кодах машины. Такие инструкции (на английском языке statement - высказывание, указание) в отечественной литературе получили название операторы.

Рассмотрим операторы языка Паскаль.

1. **Оператор присвоения** вычисляет значение выражений и записывает полученный результат в ячейку памяти, обозначенную переменной.
2. **Оператор перехода** указывает, что следующим предстоит выполнить не тот оператора, который записан после него, а помеченный меткой.
3. **Оператор обращения к процедуре** передает управление отдельному программному модулю (процедуре), после выполнения которой управление возвращается следующему оператору.
4. **Пустой оператор** состоит с единственного символа точка с запятой и не выполняет никаких действий.
5. **Условный оператор** позволяет в зависимости от некоторого условия выбрать для выполнения один из двух операторов, которые входят в него, или не выполнить ни одного.

6. **Оператор выбора** позволяет выбрать для выполнения один из многих операторов, которые входят в него, или не выполнить ни одного в зависимости от значения некоторой переменной.
7. **Оператор цикла с предусловием** повторяет выполнение оператора, который входит в него, 0 или больше раз, пока значение некоторого условия не станет ложным.
8. **Оператор цикла с послеусловием** повторяет выполнение группы операторов, которые входят в него, 1 или больше раз, пока значение некоторого условия не станет истинным.
9. **Оператор цикла с параметром** повторяет выполнение оператора, который входит в него, для всех значений некоторого параметра, которые находятся в заданном диапазоне.
10. **Составной оператор** объединяет группу любых операторов в один оператор.

Условный оператор

IF...THEN...ELSE - Важная конструкция языка

Итак, мы с вами подошли к изучению конструкций языка. Что же такое конструкция? В программировании нужны некоторые стандартные вещи, которые позволяли бы как-то усложнять программу, реализовывать ее написание не только через чередование процедур и переменных, но и заставлять ее проводить какие-нибудь проверки, как-то влиять на ход ее работы непосредственно при ее выполнении. Для этого в языках программирования существуют специальные конструкции - стандартные языковые средства, которые позволяют проверять что-либо прямо внутри

программы, повторять ее несколько раз и т.д., в общем, влиять на ее выполнение.

Одна из самых заметных конструкций - это проверка каких-то условий. Что это значит? Для того, чтобы вам это лучше понять, приведем пример задачи, которую нужно решить.

Нам нужно написать программу, которая:

1. Проверяет свободно ли впереди.
2. Если свободно, двигаться вперед.
3. Иначе – повернуть налево и двигаться вперед

В программе явно нет ничего сложного. Здесь нам и понадобится конструкция **If...Then...Else**, о которой и пойдет рассказ.

IF..Then..Else - (англ. Если....То...Иначе). Очень важная и необходимая конструкция Паскаля. Используется для проверки каких-либо условий. В этой конструкции используются знаки сравнения и опрос обстановки (см. предыдущий урок), которые в Паскале обозначаются так:

>	Больше
<	Меньше
>=	Больше или равно
<=	Меньше или равно
<>	Не равно

Запомните эти знаки, они вам дальше очень понадобятся. И не пугайтесь, если что-то не понятно. Сейчас все разъяснится.

Итак, как же оформить конструкцию **if...then...else**? Для начала давайте запишем словами:

1. Если **FreeForw**, то **Forward(3)**
2. Иначе **Left; Forward(3)**

Посмотрите внимательней на приведенный пример. Разобрались? Теперь посмотрим, как выглядит эта конструкция на Паскале:

**If FreeForw Then FreeForw
else Left; Forward(3)**

Вот вам и пример использования изучаемой конструкции. Посмотрите, разве сложно? На самом деле нет, хотя это и может так показаться с первого взгляда. Вот все комментарии, которые здесь требуются.

1. Сначала идет служебное слово **If**
2. После него стоит первая проверка, в которой как раз и выясняется свободно ли впереди.
3. Если эта проверка выполняется верно, то есть впереди свободно, то выполняется действие, указанное после слова **Then**. Обратите внимание, что после действия нет точки с запятой - ";". Она не ставиться перед словом **Else**.
4. Если же проверка не верна, то есть впереди препятствие, то выполняется действие, указанное после слова **Else**. После этого действия уже ставиться точка с запятой.

Замечания.

- Слова **Else** может и не быть. Это используется тогда, когда нужно выполнить всего одно условие и не делать ничего, если оно не верно. В таком случае после действия, указанного за словом **Then** ставиться ";;";
- Если после **Then** или **Else** несколько команд, не одна, то эти команды заключают в «операторные скобки» (вложенность) служебными словами **Begin ... End**, в ниже составленной программе.

Собственно, это и все про проверку. Теперь вспомним нашу программу, которую мы захотели написать. Сейчас мы наконец-то напишем ее, текст будет выглядеть так:

Программа Проверка;

Начало

Если СвободноВперед То Вперед

Иначе

Начало

Налево;

Вперед;

Конец;

Конец.

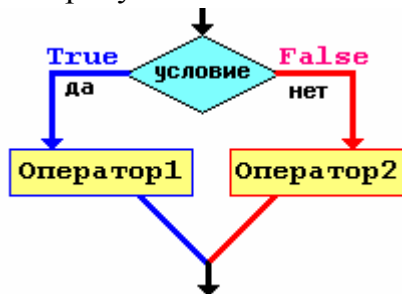
(z4_7.pas)

Полный условный оператор позволяет выбрать для выполнения один из двух простых операторов (команд, действий).

**If выражение then оператор1
else оператор2**

**если выражение то оператор1
иначе оператор2**

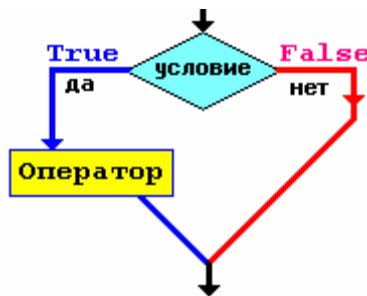
Блок-схема (графическое изображение конструкции) его приводится на рисунке ниже



Неполный условный оператор выполняет внутренний оператор только в том случае, когда значение выражения есть истинным (при выполнении условия).

If выражение then оператор если выражение то оператор

А так выглядит блок-схема (графическая конструкция) неполного условного оператора



В выражении должен образовываться результат, что имеет стандартный логический тип. Если результатом выражения есть истинное значение (true), то выполняется оператор, который следует за ключевым словом then. Если результатом выражения есть ложное значение false и присутствующее ключевое слово else, то выполняется оператор, который следует за ключевым словом else, а если оно отсутствует, то никакой оператор не выполняется.

В обеих частях условного оператора можно употреблять лишь по одному оператору. При необходимости поместить группу операторов, нужно образовать из них составной оператор.

Синтаксическая неоднозначность, которая возникает в конструкции:

if e1 then if e2 then s1else s2

решается путем такой интерпретации этой конструкции:

```
if e1 then
  begin
    if e2 then
      s1
    else
      s2
  end;
```

В общем случае ключевое слово else связывается с самым близким ключевым словом if, которое еще не связано с ключевым словом else.

Приведем пример оператора if:

```
if X < 1.5 then
  Z:= X+Y
else
  Z:= 1.5;
```

Задание 1

Используя опрос обстановки **Впереди (InFront)** составить программу движения робота как в задаче (z4_7.pas) , только на 5 клеточек.

(z4_8.pas)

```
program Проверка;
begin
  if infront=-12 then forward(5)
  else
    begin
      left;
      forward(5);
    end;
```

end.

Задание 2

Написать программу, которая выясняет какое из введенных чисел больше. Использовать процедуры ввода и вывода (**readln, writeln**)

(z4_9.pas)

Так выглядит программа на английском языке

```
program Проверка;  
  var a,b:integer;  
  begin  
    writeln('Введите через пробел 2 числа');  
    readln(a,b);  
    if a>b then  
      writeln('Вы ввели A>B')  
    else  
      writeln('Вы ввели B>A');  
  end.
```

Так выглядит программа на русском языке

```
программа Проверка;  
  переменные a, b: целые;  
  начало  
    вывестистр('Введите через пробел 2 числа');  
    вывестистр(a,b);  
    если a>b то  
      вывестистр('Вы ввели A>B')  
    иначе  
      вывестистр('Вы ввели B>A');  
  конец.
```


Конструкция Goto Label (Безусловный оператор)

Процедура GOTO

Обучающиеся часто спрашивают, как же сделать программу по настоящему интерактивной, то есть работающей в зависимости от действий пользователя, не всегда заканчивающей работу после того, как она выполнит какие-нибудь действия. Это возможно, причем просто и мы сейчас этим займемся. А поможет нам в этом процедура goto.

Процедура Goto - осуществляет переход в какое-нибудь место в программе из любой ее точки. То есть с ее помощью можно "прыгать" по программе. Обычно эти прыжки происходят из-за каких либо условий, вот пример:

1. Вводим два числа;
2. Складываем их и выводим сумму на экран;
3. Спрашиваем, повторить ли действие?
4. Если ответ утвердительный, переходим (прыгаем) на пункт 1 данного алгоритма.
5. Завершаем программу.

В этом примере программа может выполняться бесконечно, пока ответ пользователя будет утвердительным. Как же это сделать?

Команда **goto** переходит к определенному месту, которое помечается так называемой "**меткой**". Эти метки - особенный тип Паскаля, можно даже сказать и не тип, а часть языка. До того, как их использовать, метки нужно описать - сообщить Паскалю о их наличии. Описание меток происходит также, как и описание переменных.

Для описания меток имеется специальный раздел, наподобие **var** (помните, что **var** и **begin-end** не единствен-

ные разделы?) только гораздо проще его. Метки не имеют типа, нужно просто задать им имя, в Algo это натуральные числа. Раздел с метками называется **label**, описывается последним и оформляется следующим образом:

```
Program Метки_Переход;
```

```
  var
```

```
    A,B: Integer;
```

```
    Label 1;
```

```
    begin
```

```
1:
```

```
    Writeln('Введите A и B: ');
```

```
    Readln(A,B);
```

```
    If A > B Then goto 1;
```

```
    end.
```

Этот пример хорошо демонстрирует использование процедуры `goto`. Посмотрите внимательно на программу. Что она делает? Запрашивает два числа и если 1-е больше чем 2-е, то повторяется сначала. Необходимые комментарии:

1. **label**

Это и есть раздел описания меток. Служебное слово **label** озаглавливает этот раздел, после него идут имена меток. Если меток несколько, то они перечисляются через запятую.

2. **1:**

Так устанавливается метка в программе. Обратите внимание на синтаксис - после имени метки ставиться двоеточие - **":"**.

3. **If A > B Then goto 1;**

А это и есть переход при выполнении условия. Заметьте, в конструкции **if...then...else** отсутствует

слово **else**, оно нам не нужно, так как мы не делаем ничего при невыполнении условия.
Вот, собственно и все.

Задание 1

А теперь давайте вспомним ту задачу, которая была представлена вначале - ввести два числа, сложить и запросить повтор. (см. выше).

Итак, напишем эту программу с учетом пройденного материала:

Program Сложение;

var

A,B,S: Integer;

label 1;

Begin

1:

Write('Введите A B: ');

Readln(A,B);

Writeln('A+B = ',A+B);

Write('Еще раз? (Y/N): ');

Readln(S);

If S = 1 Then goto 1;

end.

Откройте Algo и введите эту программу. Запустите ее на выполнение - видите, она уже полностью реагирует на действия пользователя и как бы самостоятельно принимает решение, что делать дальше. Вот это уже полностью интерактивная программа, хотя и простая.

Особых комментариев, думаю, не требуется - все рассказано ранее. Вот только использовали мы две процедуры, которые стоит объяснить.

*Процедура **Writeln*** - эта процедура работает так: после того, как выведет строку, она переносит курсор на новую строку.

В нашем примере выводится с ее помощью несколько строк, после каждой, как вы заметили, курсор переходил на новую строку

*Процедура **Readln*** - эта процедура вводит данные после нажатия **Enter**, которые вы набираете с клавиатуры, а затем она переносит курсор на новую строку.

§ 10

Решение задач

Многие учебные пособия, в том числе и по информатике, мало содержат обучающего материала на способы и методы решения задач. И это потому, что каждая задача имеет не одно решение и все их разобрать просто невозможно. Однако существуют общие подходы и правила решения задач. Этот блок как раз и содержит это общее, что поможет вам в дальнейшем при изучении среды программирования.

С чего начинается решение любой задачи? Конечно с условия ее. И так, каков же алгоритм такого решения?

1. Внимательно прочитать условие задачи. *(Делать это надо до тех пор, пока не станет совершенно ясно, что требует от вас условие задачи).*
2. Составить математическую модель *(Последовательность основных действий).*
3. Разобраться с данными (действиями), какие являются исходные и какие надо получить.
4. Определить все необходимые команды, конструкции и операторы, которые будут использоваться в программе.
5. Составление программы *(ее можно осуществлять пошагово или составить полностью, а затем вносить коррективы (редактировать)).*

Вот примерно такой алгоритм решения задач. Рассмотрим конкретно следующую задачу.

Задача (z4_13.pas)

Сегодня в экстремальных ситуациях прибегают к обследованию помещения с помощью роботов. Составить

программу, по которой робот обойдет все помещение и выйдет из него. Помещение представляет спиральный лабиринт в два прохода, т.е. ширина между стенами равна 2-м клеткам.

- 1) Читаем условие и уясняем, что помещение представляет собой прямоугольную спиральную форму с шириной прохода в две клеточки. Такую схему можно сделать, но это уроки последующих наших дней изучения возможностей среды программирования. Поэтому загружаем ее уже готовую (z4_13.maz). Она выглядит как показано на Рис. 21.

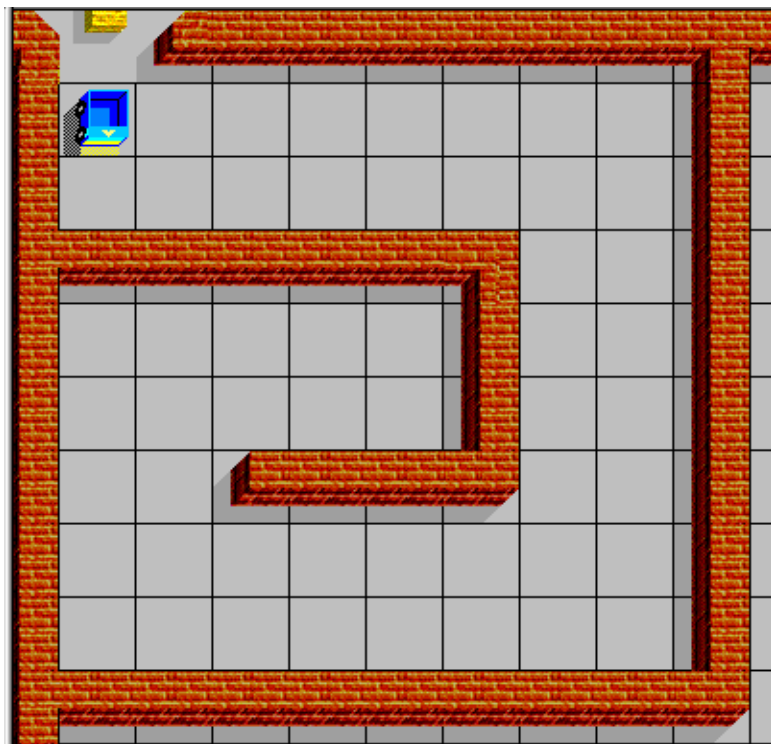


Рис. 21

- 1) *Рассуждаем дальше. Робот, войдя в помещение должен пройти все клетки, двигаясь вдоль стены либо начиная движение вперед, делая повороты; либо повернувшись налево двигается вперед, поворачивая в нужном месте. При этом он должен знать обстановку впереди. Давайте выберем движение вперед, когда у робота будет стена слева. Последовательность команд будет следующей: налево, вперед(6), направо, вперед(7), направо, вперед(7), направо, вперед(4), направо, вперед(4), направо, вперед, направо, вперед(3), налево, вперед(2), налево, вперед(5), налево, вперед(5), налево, вперед(6), направо, вперед(2). Получили двадцать четыре команды. Если их записать в таком виде то получим целую страницу программы. Такой текст понятен, но очень неграмотный.*
- 2) *Как уменьшить его? Нам в этом поможет изученная конструкция ветвления и опрашивание обстановки. Включаем в текст программы ветвление и проверяем «СвободноСлева», поворачиваем налево и делаем шаг вперед.*

If treeleft then

Begin

Left;

Forwad;

End

- 3) *В новой обстановке слева стена, поэтому теперь проверять надо свободно ли впереди и двигаться вперед. Но это уже на втором заходе проверке. Значит надо сделать так, чтобы вернуться в начало и проверить ситуацию. Этому нам поможет оператор goto, который поможет дойти до стены. Запомним, вводить его еще рано.*

Else

If freeforf then

Forward;

4) Назрела третья ситуация, надо повернуть направо

Else right;

5) Снова вернуться проверить все, если есть возможность совершить движение вперед. Вот только сейчас и вводится переход goto.

6) Смотрим готовую программу:

```
program Обследование;  
  label 1;  
  begin  
1;;  
  if freeleft then  
    begin  
      left;  
      forward  
    end  
  else  
    if freeforw then  
      forward  
    else right;  
  goto 1;  
end.
```

На русском языке:

```
программа Обследование;  
  метка 1;  
  начало  
1;;  
  если свободнослева то  
    начало
```


**налево;
вперед
конец
иначе
если свободно впереди то
вперед
иначе направо;
идти 1;
конец.**

Вот примерно такие рассуждения необходимо проводить и при решении других задач.

Задание 1

Составить программу по образцу предыдущей задачи, помечая обследованные клетки.

Задание 2

Загрузите схему склада (z4_15.maz), откройте файл (z4_13.pas), напоминающий подземный переход и выясните, будет ли работать в этой схеме программа. Ответьте на вопросы:

- 1) Почему такое происходит?
- 2) Чем это можно объяснить?

Задание 3

Загрузите схему склада (z4_16.maz), поэкспериментируйте с программой (z4_13.pas). Дайте название такой схеме. Если можно дополните ее (z4_15.pas).

§ 11

Оператор выбора

Оператор выбора (варианта) состоит из выражения (переключателя) и списка операторов, каждому из которых предшествует список констант, которые называются константами выбора, или ключевое слово *else*. В каждом списке есть хотя бы одна константа. Переключатель должен иметь порядковый тип. Все константы выбора должны быть уникальными (встречаться лишь один раз в данном операторе) и иметь тип, совместимый с типом переключателя.

Case выражение of

K11,...,K1M : оператор1;
.....
KN1,...,KNL : операторN;
else операторN1;...операторNk
end

Выбор выражение из

K11,...,K1M : оператор1;
.....
KN1,...,KNL : операторN;
иначе операторN1;...операторNk
конец

В каждом варианте записывают лишь один оператор, который может быть составным оператором. Оператор завершается точкой с запятой. Часть оператора, начинающаяся словом *else*, может отсутствовать.

Оператор выбора выполняет оператора, перед которым есть константа выбора, равная значению переключателя в момент выполнения оператора. Если такой константы выбора нет и есть слово `else`, то выполняется оператор, который следует за ключевым словом `else`. Если же ветвь `else` отсутствует и константы выбора нет, то не выполняется никакой оператор.

Пример оператора выбора рассмотрим, решая следующую задачу.

Задача

Написать программу, которая будет реагировать на введенные символы (числовые или буквенные) так, что робот будет совершать какое-то действие, например, за буквой `f` закрепим команду `forward` (вперед), за буквой `b` закрепим команду `backward` (назад) и т.д.

Решение.

Пусть это будет выглядеть так:

'f': вперед;

'b': назад;

'l': налево;

'r': направо;

's': пометить;

'c': стереть;

't': взять;

'p': положить;

Почему? В первой части было сказано, что в каждой строке должна быть константа, у нас это буква в кавычках так как используем тип `char`, после двоеточия команда или составной оператор и завершается строка точкой с запятой.

Ну а теперь сама программа:

программа Пульт;

переменные сот: символы;
метка 1;
начало
1.;
вестистр(сот);
выбор сот из
'f': вперед;
'b': назад;
'l': налево;
'r': направо;
's': пометить;
'c': стереть;
't': взять;
'p': положить;

иначе вывести(вестистр('неизвестная команда'))
конец;
идти 1;
конец.

Несколько комментариев по поводу введенных команд в программу.

Название программы не вызывает вопросов, действительно это нечто похожее на пульт управления. Где же вводить буквы для управления? Для этой цели мы используем третье окно под клетчатым полем. В нем и осуществляется диалог с программой. Буквы только латинские следует использовать для управления. После набора буквы обязательно нажать клавишу Enter.

Чтобы завершить выполнение программы нажмите кнопку Stop в строке инструментов.

Вот практически коротко и все о конструкции Выбор.

Задание

Усовершенствуйте программу так, чтобы она понимала и русские буквы.

(z4_17.pas)

Повторения. Циклы

В реальной жизни человеку очень часто приходится, выполняя какую-то работу, совершать подряд одни и те же действия, пока не будет достигнута цель. Например, скрепляя две деревянные заготовки между собой с помощью гвоздя, мы ударяем по нему молотком до тех пор, пока гвоздь не сравняется с поверхностью заготовки. Скрепляя металлические детали болтом, опять налицо совершение одних и тех же действий, вращение гаечным ключом головки болта. Можно привести еще много-много примеров в качестве подтверждения выше сказанному.

Такие действия называются повторением. Ученик учит стихотворение, пловец гребет руками совершая движение в воде, день сменяется ночью, стрелки часов совершают повторяющиеся круговые движения и т.д. все это примеры повторяющихся действий или процессов в природе, технике.

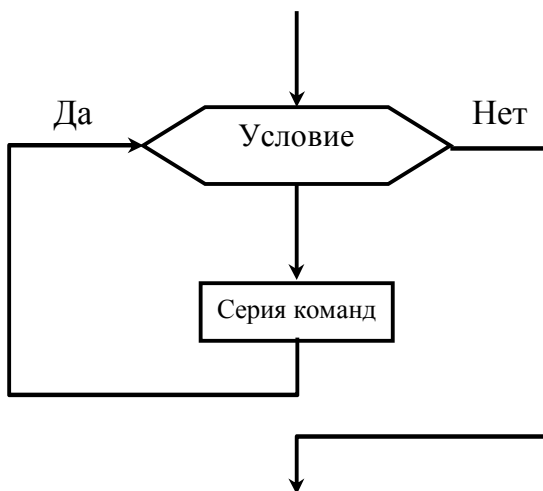
Система программирования тоже заимствована из реальной жизни. Решая задачи, мы часто делаем повторение одних и тех же действий, построений на листе бумаге и чтобы эту монотонную работу автоматизировать, как раз, и используется персональный компьютер, а точнее система программирования.

В информатике **повторение одних и тех же действий или команд называют циклом**. А способ организации цикла на языке программирования называют конструкцией, которая состоит из специальных (служебных) слов.

Вот мы и подошли вплотную к теме циклы. Но прежде чем начать вести разговор о них, а их не один, пред-

ставим это все графически. Ниже приводиться блок-схема цикла вообще, у всех их будут некоторые различные начала и окончания.

И так, блок-схема цикла:



Читать блок-схему очень просто. При выполнении программы, если надо повторить одни и те же действия при каких то условиях, их проверяют, это записано в шести-угольнике «Условие». Если оно выполняется («Да»), далее идет серия одних и тех же команд и снова возврат на проверку. Это будет до тех пор, пока условие не будет выполняться («Нет»), тогда цикл завершает работу и продолжается выполнение оставшейся части программы.

Цикл с предусловием.

Конструкция (оператор) цикла с предусловием содержит в себе логическое выражение, которое управляет

повторным выполнением оператора (который может быть составным оператором).

**While выражение do
оператор**

**Пока выражение выполнять
оператор**

Выражение, с помощью которого осуществляется управление повторением оператора, должно иметь логический тип. Вычисление его проводится до того, как внутренний оператор будет выполнен. Внутренний оператор выполняется повторно до тех пор, пока выражение не примет значение false (ложь). Если выражение с самого начала принимает значение false, то оператор, что находится внутри оператора цикла с предусловием, не выполняется. Блок-схема этого цикла такая же как и приведенная выше.

Пример оператора цикла с предусловием:

**while freeforw do
forward;**

**пока СвободноВперед выполнять
вперед;**

Вот и все. Осталось закрепить знания решением задач.

Задание 1

Составить программу, блок которой был приведен в качестве примера в учебном пособии.

(z4_18.pas)

Задание 2.

Модифицировать задачу из учебного пособия (z4_13.pas), назвав ее «Обход_склада». Завершением программы считать выход из склада, т.е. робот не должен покинуть склад.

(z4_19.pas)

Задание 3.

Используя схемы склада (z4_13, 14, 15.maz) проверить работоспособность задачи из задания 2.

