Государственное бюджетное общеобразовательное учреждение
школа № 403 Пушкинского района
Санкт-Петербурга

Учебно-методическое пособие
«Основы алгоритмизации и программирования»

Никитенко Наталия Леонидовна
учитель информатики и ИКТ, ГБОУ школы №403 Пушкинского района
Санкт-Петербурга

Оглавление

Введение

Учебно-методическое пособие "**Основы алгоритмизации и программирования** " предназначено для учителей и обучающихся 8 - 9 классов общеобразовательных школ, изучающих разделы информатики: основы теории алгоритмов и программный способ записи алгоритмов. Эти материалы могут быть полезны при подготовке к государственному экзамену по информатике. Пособие содержит 10 тем, на изучение которых отводится 10 теоретических и 10 практических занятий.

В пособие включен цикл лабораторных работ, основными целями для проведения которых являются:

- Углубление, обобщение и систематизация знаний по блок-схемам;
- Развитие алгоритмического мышления через составление блок-схем;
- Закрепление навыков составления всевозможных алгоритмов фиксированной длины на алгоритмическом языке для формального исполнителя с заданной системой команд;
- Закрепление уже полученных и приобретенных навыков работы на персональном компьютере в среде программирования КУМИР;
- Знакомство с языком программирования Pascal ABC;
- Закрепление навыков создания и выполнения программ для решения алгоритмических задач в среде Pascal ABC, содержащих базовые алгоритмические конструкции
- Развитие информационно-коммуникационной компетентности обучающихся.

Лекция 1. Алгоритмизация

Известно множество областей применения компьютера: обработка текстов и графики, передача и получение информации, создание справочников, произведение расчетов. Еще одно из важнейших направлений применения компьютеров – управление.

Управление – это целенаправленное воздействие одних объектов, которые являются управляющими, на другие объекты – управляемые. Все управляющие воздействия производятся с определенной целью с помощью команд. Таким образом, алгоритмом управления - это последовательность команд по управлению объектом, приводящую к достижению заранее поставленной цели. Объект управления – исполнитель алгоритма, в данном случае исполнитель алгоритма – устройство. Все исполнители, которые относятся к такому типу, называются формальные. Формальный исполнитель не понимает смысл команд, в информатике рассматривают только формальных исполнителей.

Алгоритм – это точное описание порядка действий, которые должен выполнить исполнитель алгоритма для решения задачи за конечное время.

Система команд исполнителя (СКИ) алгоритмического языка:

алг Название алгоритма
нач Начало тела алгоритма
кон Конец тела алгоритма

Алгоритм применительно к вычислительной технике должен обладать свойствами, которые обеспечивали бы его автоматическое выполнение:

1) Процесс решения задачи должен быть разбит на последовательность отдельно выполняемых шагов – дискретность (прерывность).
2) Алгоритм для данного исполнителя содержит только те команды, которые входят в систему его команд – понятность.
3) Каждое правило алгоритма должно быть четким и однозначным – определенность (точность).
4) Исполнение алгоритма должно завершиться за определенное количество шагов конечность (результативность).
5) Алгоритм должен выполняться для любого набора исходных данных из некоторой области, удовлетворяющих условию задачи (массовость).

Наиболее часто применяемые формы записи алгоритмов:

Обычный язык
Таблица
Псевдокод
Блок-схема

строго не определена форма представления алгоритма

Формальный алгоритмический язык (программный – строгая форма записи алгоритма в виде текста на каком-либо языке программирования.

Обычный язык - словесный способ записи - заключается в описании алгоритма средствами естественного языка (написание инструкций, руководств эксплуатации, рецептов приготовления и применения лекарств)

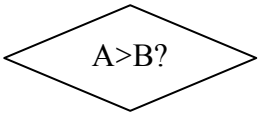
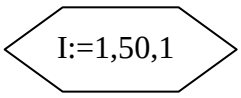
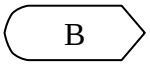
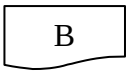
Таблица - показывает зависимость результата от исходных данных (применяется в физике, химии, математике)

Псевдокод – синтез алгоритмического и обычного языков (например, школьный алгоритмический язык).

Блок-схема - графический способ записи – последовательность блоков, соединенных линиями перехода (ветвями).

Блок-схема

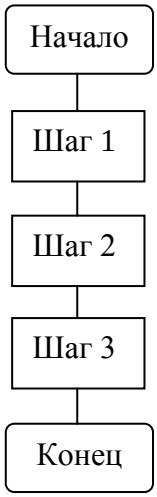
Название блока	Вид блока	Действие
Начало, остановка	Начало	Начало и конец блок-схемы
Данные	A, B	Ввод-вывод данных
Процесс	A: = 10	Вычислительное действие

Решение		Проверка условия
Подготовка		Начало цикла
Дисплей		Вывод результата на экран
Документ		Вывод результата на печать

Лекция 2. Представление алгоритмов в виде блок-схем

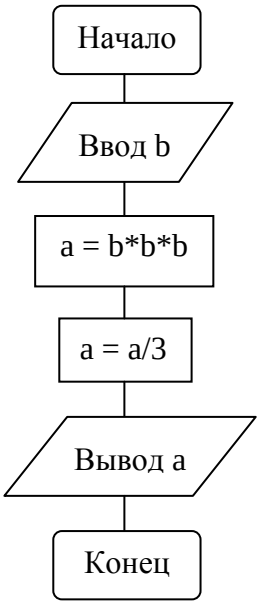
Основные (базовые) структуры алгоритмов: следование, ветвление и цикл.

Следование - линейная структура, которая описывает алгоритм действий, следующих одно за другим в линейной последовательности.



Пример: Найти a = (b*b*b)/3

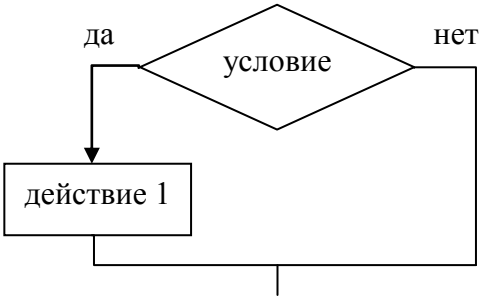
Алгоритм решения поставленной задачи:



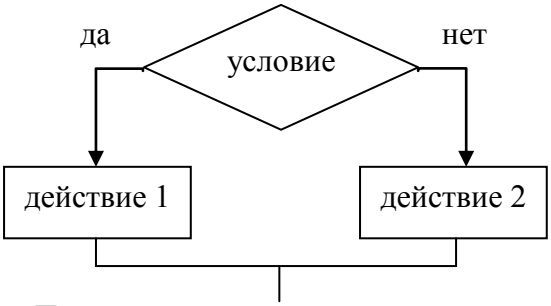
Ветвление (развилка) – используется в случае, если выполнение программы может пойти двумя разными путями в зависимости от результата проверки.

Ветвление может быть полным и неполным.

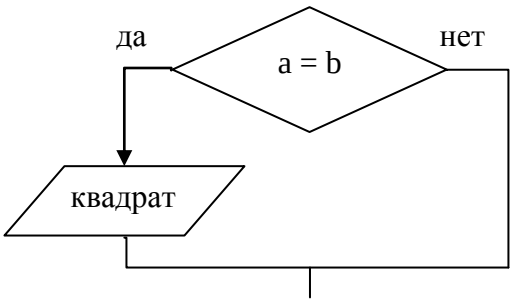
Для неполной структуры характерно в случае справедливости условия выполнять действие 1, иначе – ничего не выполнять.



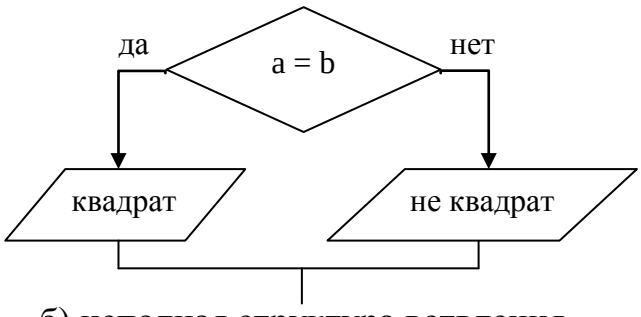
Для полной структуры характерно в случае справедливости условия выполнять действие 1, иначе – действие 2.



Пример: Дан прямоугольник со сторонами a и b. Определить, является ли эта фигура квадратом. Решение представим в виде следующего алгоритма:



а) полная структура ветвления

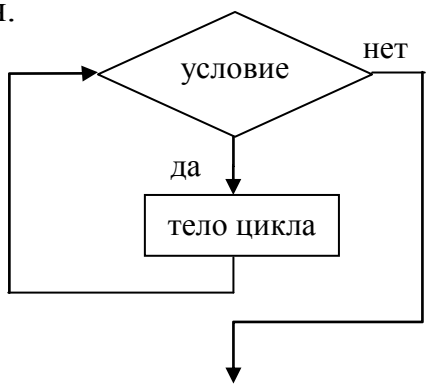


б) неполная структура ветвления

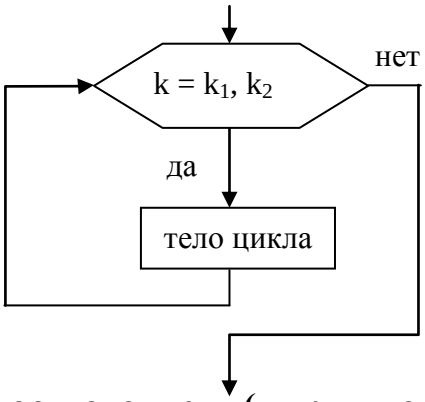
Структура цикл служит для записи алгоритмов, в которых некоторая их часть многократно должна повторяться. Различают три вида циклов, в зависимости от способа определения количества повторений.

Цикл с предусловием (цикл «пока») – проверяется условие в начале цикла, и если условие не выполняется, то действия в теле цикла не будут выполнены ни разу. Пока выполняется условие при реализации цикла, будет повторяться тело цикла. В некоторых случаях может происходить «зацикливание», то есть цикл работает бесконечно долго; чтобы избежать

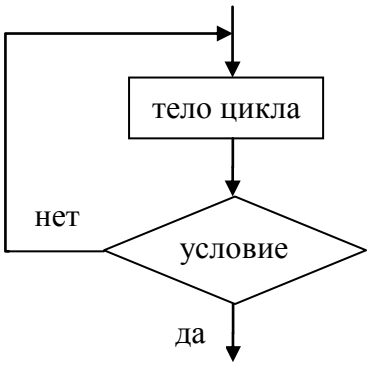
этого, необходимо правильно записывать условие – параметры условия должны изменяться.



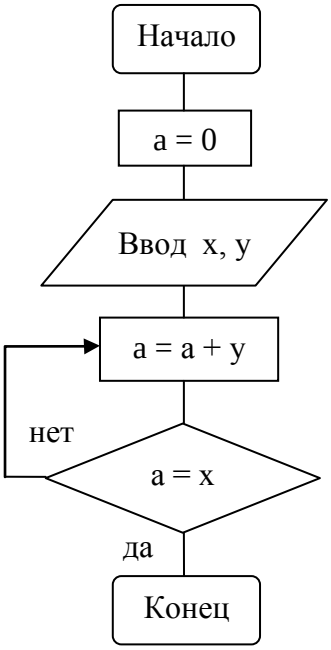
Цикл с параметром (цикл с переменной) - используется в случаях, когда заранее известно количество повторений цикла. В заголовке цикла задаются начальное и конечное значение счетчика цикла (переменной), который по умолчанию считается равным или 1, или - 1. Такая переменная может быть только целым числом. С каждым шагом цикла переменная цикла может увеличиваться или уменьшаться.



Цикл с постусловием (цикл «до») – используется в случае, если необходимо выполнить цикл хотя бы один раз. Цикл будет выполняться до тех пор, пока не выполнится условие. Условие проверяется в конце цикла, таким образом, условием окончания выполнения цикла будет выполнение условия.



Пример циклического алгоритма с постусловием:



В данном алгоритме показано, как решается задача наполнения емкости (большой) с помощью другой (маленькой) через переменную **a**, которая обозначает объем сахара в большей емкости, **x** – объем большой емкости, **y** – объем маленькой емкости.

Заметим, что в начале выполнения алгоритма a = 0, то есть большая емкость первоначально пуста.

Словесно опишем данный алгоритм:

Шаг 1. Насыпать сахар в маленькую емкость **y**.

Шаг 2. Высыпать сахар из емкости **y** в большую емкость: **a = a + y**

Шаг 3. Делаем проверку: полна вторая емкость: **a = x?** Если условие выполнено, задача решена, если **a < x**, выполнить Шаг 1, Шаг 2 до тех пор, пока не выполнится условие **a = x.**

Лекция 3. Разработка алгоритма с помощью учебного исполнителя Робот

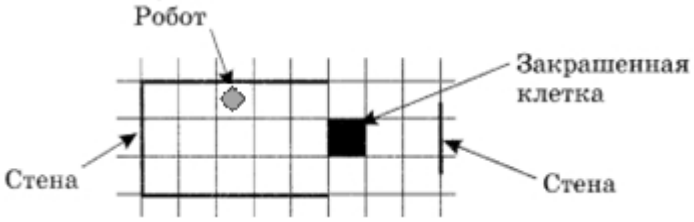
Прежде чем говорить о реализации алгоритмов для исполнителя, нужно понимать, что такое программа и чем она отличается от алгоритма.

Программа – алгоритм, записанный на языке программирования.

Как мы помним из первой лекции, алгоритмы могут иметь различные формы представления, а программа должна быть записана на языке исполнителя.

Любому учебному исполнителю свойственна **среда деятельности, система команд управления и режимы работы.**

Среда исполнителя Робот – прямоугольное клетчатое поле, между клетками которого могут быть стены (в среде КУМИР).



Система команд исполнителя Робот:

Простые команды: вверх, вниз, влево, вправо, закрасить.

Команды логические: (проверки условия)

сверху свободно	снизу свободно
слева свободно	справа свободно.

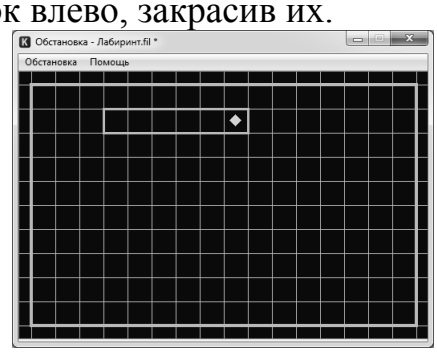
Логические связки: И, НЕ, ИЛИ (сложные условия)

Пример: (не слева свободно) или (не справа свободно)

команда ветвления:	**команда цикла:**
если условие **то**	**нц пока** условие
серия команд	серия команд
всё	**кц**

Линейные программы для исполнителя

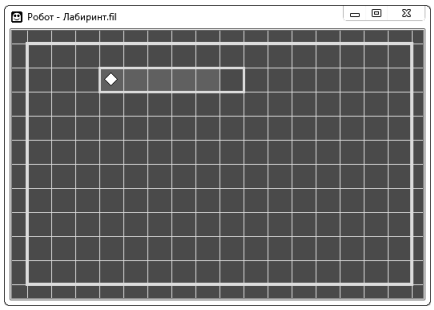
Задача: Робот в произвольной точке поля. Передвинуть Робота на 5 клеток влево, закрасив их.



Напишем программу для Робота:

```
использовать Робот
алг лабиринт
нач
    влево; закрасить
    влево; закрасить
    влево; закрасить
    влево; закрасить
    влево; закрасить
кон
```

Как видно, клетки, первоначально находящиеся слева от Робота закрашены:

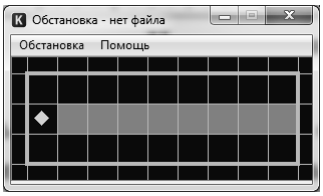


В данном случае показан пример создания линейной программы для исполнителя.

Циклические алгоритмы

Цикл «пока»

Задача: закрасить все клетки справа от Робота, при условии, сто справа на неизвестном расстоянии есть стена и количество клеток, которые нужно закрасить неизвестно.



Очевидно, что пока будет выполняться условие **справа свободно**, нужно выполнять команды:

вправо; закрасить

Для оформления таких последовательностей действий используется специальная конструкция алгоритмического языка — цикл «пока».

ПОКА справа свободно ДЕЛАТЬ
 вправо
 закрасить
КОНЕЦ

В общем виде цикл «пока» записывается так:

ПОКА <условие> ДЕЛАТЬ
 <тело цикла (последовательность команд)>
КОНЕЦ

Напишем программу для решения задачи:

использовать Робот
алг линия
нач
вправо
нц пока справа свободно
если не справа стена
то

закрасить
все
кц
кон

Цикл «раз»

Задача: нарисовать букву «П» размер высотой 6 клеток, шириной 4 клетки с помощью исполнителя Робот. Исходное положение Робота показано на рисунке ниже:

В этом алгоритме известно количество шагов исполнителя, поэтому используем цикл «раз». Его конструкция в общем виде выглядит так:

НЦ количество РАЗ
КЦ

По аналогии с предыдущим примером Робот должен закрашивать очередную клетку, а затем переходить на следующую – сначала вверх, затем вправо и вниз.

Программа для Робота:

```
использовать Робот
алг буква
нач
  нц 5 раз
    закрасить; вверх
  кц
  закрасить
  нц 3 раз
    вправо; закрасить
  кц
  нц 5 раз
    вниз; закрасить
  кц
кон
```

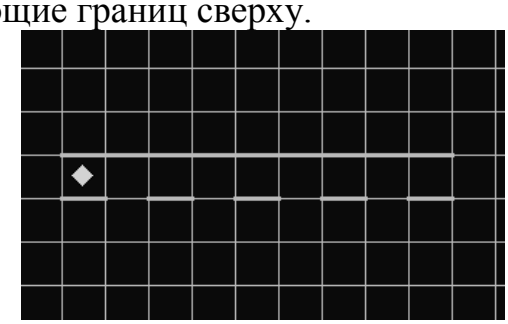
Лекция 4. Ветвление и последовательная детализация алгоритма для исполнителя Робот

Вспомним, что форма организации действий, при которой в зависимости от выполнения или невыполнения некоторого условия совершается либо одна, либо другая последовательность действий, называется **ветвлением.**

Для организации ветвлений в СКИ Робота предусмотрена специальная команда ЕСЛИ. Ее общий вид:

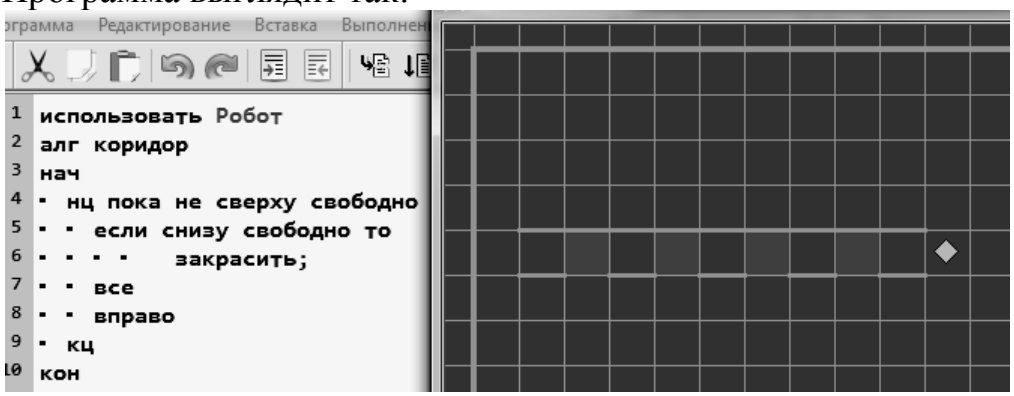
ЕСЛИ <условие> ТО <серия действий 1>
ИНАЧЕ <серия действий 2>
КОНЕЦ

Задача: Робот находится в горизонтальном коридоре, нижняя граница которого сплошная, а в нижней имеются выходы (рис. ниже). Требуется провести Робота через весь коридор и закрасить клетки коридора, не имеющие границ сверху.



Единственным признаком коридора является наличие границы сверху, значит, условие **НЕ сверху свободно** должно выполняться. Если при этом выполняется условие **снизу свободно**, то клетку нужно закрасить, иначе — закрашивать не надо. Известно, что слева и справа от горизонтального коридора есть клетки.

Программа выглядит так:

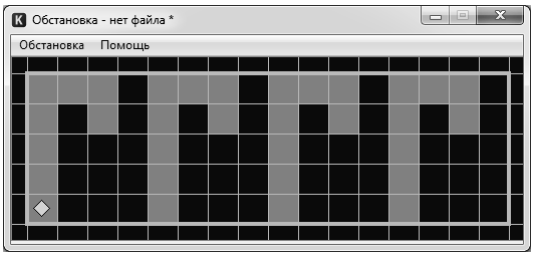


При решении больших «сложных» задач применяют метод последовательного уточнения: одна «сложная» задача разбивается на более мелкие подзадачи, каждую из которых можно оформить как самостоятельный алгоритм. Составляется так называемый основной алгоритм, в котором для решения подзадач используются вызовы вспомогательных алгоритмов. Этот метод дает возможность работать над проектом группе программистов, каждый специалист решает свою подзадачу.

Процедура (вспомогательный алгоритм)— это алгоритм, который можно использовать в других алгоритмах, указав только его имя.

Один и тот же алгоритм может рассматриваться и как основной, и как вспомогательный. Так, если в записи алгоритма **АВС** встречается команда вызова алгоритма Х, то говорят, что алгоритм **Х** является **вспомогательным** для **АВС**, и наоборот, алгоритм **АВС** – **основной** для процедуры **Х**. В алгоритмическом языке сначала записывается основной алгоритм, сразу же под ним записывается вспомогательный. Вспомогательных алгоритмов может быть записано несколько.

Задача: нарисовать орнамент, состоящий из четырех элементов. Исходное положение Робота указано на рисунке:



Рассмотрим один элемент орнамента, для его получения мы можем написать следующую программу:

```
алг зигзаг
нач
  нц 4 раз
    закрасить; вверх
  кц
  закрасить
  нц 2 раз
    вправо; закрасить
  кц
  вниз; закрасить; вправо
кон
```

Таких «зигзагов» нам нужно нарисовать четыре, при этом Робот не должен разбиться. В данной ситуации было бы нецелесообразно алгоритм **зигзаг** четыре раза записывать в основной алгоритм.

```
использовать Робот
алг орнамент
нач
   зигзаг
   нц 3 раз
     вниз
   кц
   если справа свободно
     то
       вправо
   все
```

```
    зигзаг
    нц 3 раз
      вниз
    кц
    если справа свободно
      то
        вправо
    все
    зигзаг
    нц 3 раз
      вниз
    кц
    если справа свободно
      то
        вправо
    все
    зигзаг
    нц 3 раз
      вниз
    кц
    если справа свободно
      то
        вправо
    все
кон
```

```
алг зигзаг
нач
  нц 4 раз
    закрасить; вверх
  кц
  закрасить
  нц 2 раз
    вправо; закрасить
  кц
вниз; закрасить; вправо
кон
```

Как видно из примера, программа получается очень длинной. Обратите внимание, что алгоритм **зигзаг** записан после основного алгоритма (выделен в рамке). Здесь было бы удобнее использовать вспомогательный алгоритм **зигзаг** в **цикле «раз»**, так как нам необходимо создать 4 одинаковых элемента, а также применить цикл для перемещения Робота в клетку, с которой начинается следующий элемент. Необходимо также учесть, чтобы исполнитель завершил программу без аварии; для этого применим условие

ЕСЛИ не справа стена **ТО** вправо. Итак, окончательно программа для исполнителя выглядит так:

```
использовать Робот
алг орнамент
нач
        нц 4 раз
                зигзаг
                нц 3 раз
                вниз
                кц
                если справа свободно
                то
                вправо
                все
        кц
        кон
алг зигзаг
нач
        нц 4 раз
        закрасить; вверх
        кц
        закрасить
        нц 2 раз
        вправо; закрасить
        кц
        вниз; закрасить; вправо
кон
```

Сделаем выводы:

Сложные алгоритмы удобно строить путем пошаговой детализации, применяя отдельные алгоритмы, которые называют вспомогательные.

Вспомогательные алгоритмы применяют в случаях, когда в разных местах алгоритма необходимо выполнение одной и той же последовательности шагов.

Вспомогательный алгоритм (процедура) - инструмент, который позволяет:

1) Сокращать текст основного алгоритма (повторяющиеся действия не нужно многократно описывать);
2) Облегчать понимание текста программы, делая его похожим на текст на естественном языке, если использовать содержательные имена процедур;
3) переделывать программы и исправлять в них ошибки, которые относятся к основному алгоритму;
4) решать "трудную" задачу, разбивая ее на более мелкие и "легкие" подзадачи.

Лекция 5. Знакомство с языком Паскаль

Для разработки программ управления исполнителем компьютер с целью решения различных информационных задач используется **программирование.**

Система программирования – это программное обеспечение компьютера, предназначенное для разработки программ, записанных на определенном языке программирования.

В настоящее время используются языки высокого уровня. Команды языка высокого уровня – **операторы** – это слова естественного языка, которые значительно упрощают работу программиста. Существует большое количество языков программирования, наиболее распространенные из них: BASIC, Pascal, C, C++.

Язык программирования Pascal (Паскаль) был разработан в 1971 году швейцарским профессором Никлаусом Виртом. Свое название язык получил в честь французского ученого, изобретателя механической вычислительной машины Блеза Паскаля.

Структура программы на Паскале

Язык программирования **Паскаль** базируется на трех составляющих: алфавите, синтаксисе (жестких правил построения элементов языка) и семантике (смысле и правилах использования тех элементов языка, для которых были даны синтаксические определения).

PROGRAM - имя программы (английскими буквами, одно слово);
USES - подключаемые библиотеки (модули);
LABEL - список меток;
CONST - раздел описания констант (постоянные величины, их нельзя изменять);
TYPE - описание типов переменных;
VAR - определение глобальных переменных (описание всех переменных величин, которые в программе могут изменяться) ;
ОПРЕДЕЛЕНИЕ ПРОЦЕДУР;
ОПРЕДЕЛЕНИЕ ФУНКЦИЙ;
BEGIN
основной блок программы
END.

Не все приведенные выше блоки обязательно использовать.

Пример программы:

```
Program primer;
  Begin
  End.
```

Данная программа ничего не делает, так как в ней нет ни одного оператора.

Пунктуация Паскаля

Почти после каждой строчки в программе ставится знак "**;**". Этот знак говорит о том, что строка закончена. Знак "**;**" не ставится после служебного слова **BEGIN** и последнего **END.** Перед словом **END** точку можно не ставить.

Запятая (**,**) является разделителем элементов в списках переменных, а также списках вводимых и выводимых величин.

Текст программы всегда заканчивается точкой.

Для удобства программисты включают комментарии, которые не влияют на работу программы. В комментариях можно использовать русские буквы. Комментарии оформляются либо фигурными скобками, либо знаком «//».

Особенностью языка является то, что различия между прописными и строчными буквами нет. Варианты записи **BEGIN, END, begin, end, Begin, End** тождественны.

Алфавит языка

Алфавит - это совокупность допустимых в языке символов. Алфавит Паскаль АВС включает следующий набор основных символов:

- строчные и прописные латинские буквы:
  A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
  a b c d e f g h i j k l m n o p q r s t u v w x y z
- подчеркивание: _
- арабские цифры: 0 1 2 3 4 5 6 7 8 9
- знаки операций: **+ - * / = <> < > <= >= :=**
- ограничители: **. , ' ( ) [ ] { }.. : ; //**

Переменные

Переменная – это объект программы, имеющий имя, тип и значение. В памяти компьютера для хранения переменной выделяется определенное место.

Каждая переменная имеет свое уникальное имя. Имя переменной указывает, в каком месте памяти компьютера она хранится. Имена составляются из букв латинского алфавита и цифр, первой должна стоять буква или знак «_». Не используются в качестве имен зарезервированные слова языка программирования (DIV, CASE, CONST, DOWNTO, DO, ELSE и др.)

Тип переменной – это диапазон всех значений, которые может принимать данная переменная.

Тип переменной определяет:

1) Необходимый размер памяти
2) Диапазон значений, которые может принимать величина
3) Возможные операции над величиной
4) Формат представления величин

Стандартные типы переменной: числовой, литерный, логический.

Операторы

Рассмотрим пример программы, вычисляющей произведение двух целых чисел.

program proizvedenie; //название программы
var x, y, z: integer**; //** объявление переменных целого типа
begin // начало программы
writeln ('Введите два целых числа'); //на экране появится надпись начала диалога с пользователем
read (x, y); // программа ждет ввода двух чисел целого типа
z **:=** x * y; // в переменную z записывается результат произведения введенных чисел
writeln (x, ' * ', y, '=', z); // на экране дисплея видим запись 10*20=200 – числа, введенные пользователем и результат их умножения
end. // окончание программы

В комментариях видно, какую операцию в данной строке совершает программа.

Оператор присваивания – это команда для записи нового значения в переменную. При записи нового значения старое стирается.

Пример: z **:=** x * y;

Оператор ввода данных – предназначен для ввода данных пользователем.

Если в операторе несколько переменных, то данные вводятся через пробел или *Enter*.

Пример: read (x, y) или readln (x, y).

Отличие первого от второго варианта в том, что с помощью оператора readln курсор перемещается на новую строку (от англ. read line).

Оператор ввода данных – предназначен для вывода компьютером результата работы программы на экран.

Пример: **write** (x, '=', y)
writeln ('Введите два целых числа')
writeln;

Во втором случае, аналогично оператору ввода данных, курсор переводится на новую строку.

В третьем варианте оператор делает пропуск строки.

```
Program urok1;
  Begin
  Writeln ('Урок окончен. Вы успешно изучили первый урок!');
  End.
```

Лекция 6. Линейные вычислительные алгоритмы

Прежде чем переходить к линейным алгоритмам, рассмотрим подробнее типы данных и стандартные функции, используемые в Паскале.

Типы данных

Integer – целый тип (в памяти компьютера от 2 до 4 байтов)

Real – вещественный тип (в памяти компьютера 6 байтов)

Для написания программ необходимо познакомиться еще с двумя операторами:

Div и **Mod**.

Div – используется, чтобы получить целый результат деления целых чисел.

Пример: 15 Div 4 =3

Mod - используется, чтобы получить целый результат остатка от деления целых чисел.

Пример: 15 Mod 3 = 0

Стандартные функции

Функция - это такая организация преобразования переданного ей значения, при которой это измененное значение передается обратно.

Арифметические функции:

Abs(x) - вычисляет модуль (абсолютную величину) числа x;

Cos(x) - вычисляет косинус угла x, заданный в радианах;

Sin(x) - вычисляет синус угла x, заданного в радианах;

Frac(x) - выделяет дробную часть числа x;

Int(x) - выделяет целую часть числа x;

Sqr(x) - вычисляет x^2;

Sqrt(x) - вычисляет $\sqrt{x}$.

Функции преобразования типов:

Round(x) - округляет вещественное число x до ближайшего целого;

Trunc(x) - выделяет целую часть числа x, отбрасывая дробную часть

Линейные алгоритмы - это такие алгоритмы, в которых действия выполняются последовательно одно за другим.

Как правило, в них есть ввод данных, вычисление и вывод результата.

Этапы решения задач на компьютере

1) Постановка задачи.

2) Построение математической модели.

3) Алгоритмизация.

4) Составление сценария работы на компьютере (этот этап мы пока будем опускать).

5) Написание задачи на языке программирования.

6) Отладка и тестирование программы.

7) Анализ полученных результатов.

Задача «Путешествие по городам Золотого кольца России»

Турист выехал из Москвы на автомобиле и посетил сначала Сергиев Посад - от Москвы **A** км, затем Переславль-Залесский - от Сергиева Посада **B** км. Определите расстояние, которое ему осталось проехать до Владимира, если он посетил сначала Ростов Великий, Ярославль, Кострому, Иваново и Суздаль – от Переславля-Залесского до Суздаля всего **C** км. Все исходные данные задаются с клавиатуры.

1. Постановка задачи

Исходные данные

Переменная	**Смысловое значение**	**Тип**
A	Расстояние от Москвы до Сергиева Посада	вещественный
B	Расстояние от Сергиева Посада до Переславля-Залесского	вещественный
C	Расстояние от Переславля-Залесского до Суздаля	вещественный
rasstoyanie	Расстояние от Москвы до Владимира по Золотому кольцу	вещественный

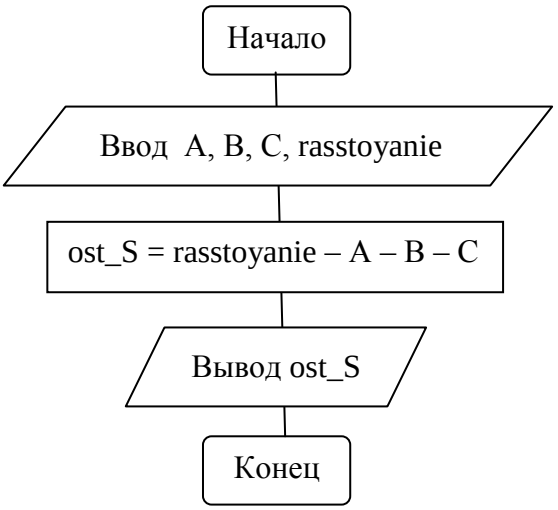
Результат

ost_S **–** расстояние от Суздаля до Владимира - вещественный

2. Построение математической модели.

ost_S = rasstoyanie – A – B – C

3. Алгоритм решения



5. Программа

```
Program puteshestvie;
Uses crt;
Var  A, B, C, rasstoyanie, ost_S : real;
Begin
Write (’Какое расстояние до Сергиева Посада, Переславля –Залесского, Суздаля?’);
Readln (A, B, C);
Write (’Какое расстояние от Москвы до Владимира?’);
Readln ( rasstoyanie );
ost_S: = rasstoyanie – A – B – C;
Writeln (’ Расстояние от Суздаля до Владимира ’  , ost_S :5:2 , ’ км’);
End.
```

Обратите внимание на формат переменной **ost_S:5:2 -** эта запись обозначает, что будет выведено на экран вещественное число не более чем из 5 цифр, две из которых после целой части (например: 152.12; _15.05).

6. Тест

Задаем значения
A = 75.3
B = 75.5
C = 351.0
Rasstoyanie = 537.0
Вычисляем
ost_S = 537 - 75.3 - 75.5 – 351 = 35.20

7. Анализ полученного результата

Проверяем полученный на экране результат с нашим тестом. Если результат совпадает, то программа работает правильно. Если – нет, то надо искать ошибку.

Вывод: в линейных программах операторы выполняются последовательно друг за другом, порядок их выполнения не зависит от входных данных.

Лекция 7. Программирование ветвлений на Паскале

В линейных алгоритмах команды выполняются одна за другой. Часто при решении некоторых задач возникают ситуации, когда порядок действий может изменяться, в зависимости от того, какие данные поступили. Например, находим решение линейного уравнения ax = b при условии, что коэффициент a не равен нулю. В таких случаях используется конструкция ветвление. В Паскале конструкция реализуется с помощью условного оператора.

Алгоритмы, в которых последовательность шагов зависит от выполнения некоторых условий, называются **разветвляющимися.**

if <условие> **then begin** //что делать, если условие верно
end
else begin //что делать, если условие неверно
end;

Запомните:

1. Перед ***else*** **не** ставится точка с запятой;
2. Вторая часть (***else*** …) может отсутствовать - неполная форма ветвления;
3. Если в блоке один оператор, можно не использовать слова ***begin*** и ***end.***

Задача: написать программу нахождения меньшего из двух значений целых чисел и определить, является найденное число четным или нечетным.

```
program menshee;
        var a, b, min: integer;
begin
  writeln ('Введите два целых числа');
  readln ( a, b );
    if  a>b  then
    min:= b
    else min:= a; // полная форма ветвления
  writeln ('Наименьшее число ', min);
    if  a mod 2=0  then // остаток от деления числа на 2 равен нулю
    write (min, ' - число четное')
    else   write (min, ' - число нечетное ')
end.
```

Можно программу написать, используя неполную форму ветвления. Для этого нужно одно из значений обозначить как минимальное. Первая часть программы будет выглядеть так:

```
program menshee;
var a, b, min: integer;
begin
  writeln ('Введите два целых числа');
  readln ( a, b );
  min: = a
    if  a>b  then
    min:= b // неполная форма ветвления
  writeln ('Наименьшее число ', min);
…
```

Сложное условие – это условие, состоящее из нескольких простых условий (отношений), связанных с помощью **логических операций**:

- **not** – НЕ (отрицание, инверсия)
- **and** – И (одновременное выполнение условий)
- **or** – ИЛИ (выполнение хотя бы одного из условий)

Каждое из простых условий обязательно заключается в скобки.

***Пример*:**

```
if not (a < b) or (c = d) and (b <> c)
then begin
  ...
end
```

Задача: a, b, c - стороны предполагаемого треугольника. Требуется написать программу, которая сравнивает длину каждого отрезка с суммой двух других. Если хотя бы в одном случае отрезок окажется больше суммы двух других, то треугольника с такими сторонами не существует.

Пример:

```
program treugol;
uses crt;
var a, b, c: integer;
begin
  write ('Длины сторон треугольника:  ');
  readln ( a, b, c );
    if  (a<b+c) and (b<a+c) and (c<a+b)  then
    writeln ('Треугольник существует. ')
    else  writeln ('Треугольник не существует. ')
end.
```

Лекция 8. Программирование структуры «выбор» на Паскале

На прошлом занятии вы познакомились с условным оператором If, с помощью которого программа может выполнять ту или иную операцию по значению логического условия. Используя несколько операторов If, можно производить ветвление по последовательности условий, можно применять составные условия.

Пример фрагмента программы:

```
if cifra = 0 then
   write ('Нуль');
if cifra = 1 then
   write ('Единица');
if cifra = 2 then
   write ('Два');
…
```

В этом фрагменте показано, как преобразовать целое число (в диапазоне 0-9) к его словесному представлению при помощи ряда операторов If.

Как вы видите, этот подход к решению задачи очень однообразен и утомителен. В языке программирования Паскаль, кроме оператора **if**, предусмотрен оператор ветвления по ряду условий в форме, более удобной для чтения программ - это оператор выбора **case**.

Оператор выбора позволяет выбрать одно из нескольких возможных вариантов исхода программы. Параметром, по которому осуществляется выбор, служит селектор - выражение любого типа (кроме типов **REAL** и **STRING**).

Общая форма записи следующая:

```
case порядковая_переменная of
значение1 : оператор (группа операторов);
значение2 : оператор (группа операторов);
```

. . . . . .

значениеN : оператор (группа операторов)

else оператор (группа операторов);

end;

Строка **else** может отсутствовать. Если в списке выбора не окажется константы, совпадающей со значением порядковой переменной, то управление передается оператору, стоящему за словом **END**.

В программе при использовании оператора выбора сначала вычисляется значение выражения, стоящего после слова **case**, а затем выполняется оператор (или составной оператор), соответствующий результату вычисления выражения.

Пример 1:

```
case chislo mod 2 of
  0 : writeln (chislo, ' - число четное')
else writeln (chislo, ' - число нечетное');
end;
```

Пример 2:

```
case n of
1 : writeln (' желтый');
2: writeln (' красный');
3: writeln (' зеленый');
end;
```

В случае, если один оператор выполняется при нескольких значениях, то их перечисляют через запятую.

Пример:

```
case sezon of
  1, 2, 12: writeln ('Зима');
  3, 4, 5 : writeln ('Весна');
  6, 7, 8 : writeln ('Лето');
  9, 10, 11 : writeln ('Осень');
end;
```

Если оператор должен выполняться при нескольких последовательных значениях селектора, образующих диапазон, можно использовать запись типа ‘a’..’z’, которая включает промежуток от a до z .

Пример:

```
case Chislo of
  0..9: write ('Это число является цифрой');
```

Задача: Напишите программу, по которой выводится тип группы детского сада по введенному числу из промежутка [1,7].

```
program gruppa;
```

```
uses crt;
var data: integer;
begin
  write ('Введите целое число лет:  ');
  readln (data);
    case data of
   1..2: writeln ('ясли ') ;
   3,4: writeln ('младшая ') ;
   5: writeln ('старшая ');
   6,7: writeln ('подготовительная ')
   else  writeln ('число должно входить в интервал от 1 до 7')
   end;
 end.
```

Лекция 9. Программирование циклов

В процессе решения задач часто появляется необходимость повторять одни и те же действия несколько раз. Рассмотрим следующие ситуации:

1. Известно количество повторений до выполнения тела цикла
2. Не известно до выполнения тела цикла количество повторений
3. Необходимость выполнить тело цикла хотя бы один раз

В языке программирования Паскаль существует три вида циклических конструкций в соответствии с приведенными выше ситуациями.

Цикл FOR

Цикл for называют часто циклом со счетчиком. Этот цикл используется, когда число повторений не связано с тем, что происходит в теле цикла. Нужно понимать, что количество итераций цикла for заранее известно до его выполнения, то есть количество повторений может быть вычислено заранее (хотя оно не вычисляется).

Счетчик – это переменная типа integer (чаще всего). Если между начальным и конечным выражением указано служебное слово **to**, то отсчет будет идти с шагом, увеличивающим на единицу. Если же указано **downto**, то значение параметра будет уменьшаться на единицу.

В заголовке цикла указываются два значения. Первое значение присваивается так называемой переменной - счетчику, и от этого значения начинается отсчет количества повторений (итераций). Второе значение указывает, при каком значении счетчика цикл должен остановиться. Таким образом, количество итераций цикла определяется разностью между вторым и первым значением плюс единица. В Pascal тип переменной счетчик не может быть real.

Цикл **for** существует в двух формах:

```
for счетчик:=A to B do
    тело_цикла;
for счетчик:=B downto A do
    тело_цикла;
```

Количество итераций цикла **for** известно до его выполнения, но не до выполнения всей программы.

Пример:

```
Var k, m: integer;
 begin
   write ('Количество +:  ');
   readln (m);
    for  k:= 1 to m do
      write ('(+) ');
 end.
```

Как видим, количество выполнений данного цикла определяется пользователем, при вводе числа значение присваивается переменной, а затем используется в заголовке цикла. Но когда это значение используется, циклу уже точно известно, сколько раз надо выполниться.

Цикл WHILE

Цикл **while** - циклом с предусловием. В заголовке цикла находится логическое выражение. Если оно истинно, то тело цикла будет выполняться, если оно ложно – то не выполнится ни разу. Тело цикла выполнится столько раз, сколько раз логическое выражение будет принимать значение истина. Важно предусмотреть изменение переменной, фигурирующей в заголовке цикла, так, чтобы когда-нибудь обязательно наступала ситуация **false**, в случае, если условие никогда не станет ложным, программа **зацикливается.**

Пример:

```
Var k, m: integer;
 begin
   write ('Количество +:  ');
   k := 1;
    while k<= m do begin
      write ('(+) ');
      k := k + 1
    end;
 end.
```

Здесь видим, что программа с циклом FOR (рассмотренная выше) легко может быть преобразована в цикл WHILE.

Цикл REPEAT

Цикл **while** может не выполниться ни разу, если логическое условие изначально ложно. Иногда возникает необходимость, чтобы тело цикла было выполнено хотя бы один раз. В таком случае используется цикл Repeat – цикл с постусловием.

В цикле **Repeat** логическое выражение стоит после тела цикла. В отличие от цикла **while**, в этом цикле в случае выполнения логического

условия происходит выход из цикла, в случае невыполнения – его повторение.

Примеры:

```
Var k, m: integer;
begin
    write ('Количество +: ');
    readln (m);
    k:= 1;
       repeat
       write ('(+) ');
       k := k + 1
       until k > m;
end.
```

```
program summa;
var number, sum: integer;
 begin
  sum:=0;
    repeat
    read  (number);
    sum:= sum + number
    until  number = - 1;
   writeln (sum)
  end.
```

Вывод: наиболее универсальным можно считать оператор цикла Паскаля с предусловием – с помощью таких операторов можно задать циклические процессы, определяемые операторами цикла с параметром и постусловием.

Лекция 10. Одномерные массивы в Паскале

При обработке большого количества однотипных данных, находящихся в памяти компьютера, возникают затруднения, так как приходится давать имя каждой ячейке памяти. В этом случае группе ячеек дают одно имя, при этом каждая ячейка получает собственный номер. Выделенную таким образом область памяти называют массивом. С понятием «массив» чаще всего приходится сталкиваться при решении научно-технических и экономических задач.

Массив – это именованная группа однотипных данных, состоящая из фиксированного числа элементов. Каждый элемент массива имеет уникальный номер. Элементы нумеруются по порядку, чаще всего начиная с единицы, но необязательно. Порядковый номер элемента массива называется **индексом** этого элемента, индекс определяет положение

элемента в массиве. Количество элементов массива называется его размерностью.

Примеры: квартиры в доме; список учеников в классе; школы в городе; данные о температуре воздуха за год.

Добыча нефти в России и странах СНГ за период 1950 по 2000 г.г

№	1	2	3	4	5	6
Год	1950	1960	1970	1980	1990	2000
Количество нефти в млн т	40	150	350	605	570	395

Такую таблицу называют линейной. В программировании линейная таблица называется одномерным массивом. Годы в таблице можно пронумеровать как 1, 2, 3, 4, 5, 6. Например, через H [1], обозначено количество нефти, добытой в 1950 году, а через H [6] – количество нефти в 2000 году.

Подчеркнем, что все элементы определенного массива должны иметь один и тот же тип. У разных массивов типы данных могут различаться. Например, этот массив может состоять из чисел типа **real** (вещественный). Если массив состоит только из целых чисел, то используется тип **integer**.

Объявление массива позволяет правильно оперировать с данными и резервировать место для хранения данных в памяти.

Описание массива

Объявление массива целых чисел:

var A **: array** [1..5] **of** integer; //описан массив A из 5 целочисленных значений

Описание через константу:

Const N=12**;**

var A**: array**[1.. N**] of** integer; //описан массив A из 12 целочисленных значений

Массивы других типов:

C**: array [**1..20] **of char**;

Индексы других типов:

var A**: array** ['A'..'Z'] **of real**;

D**: array** ['a'..'z',w2..w4] **of** string;

Заполнение массива

Ввод массивов осуществляется поэлементно.

Введем одномерный массив **A**, состоящий из **15** элементов, то есть необходимо ввести некую последовательность элементов $\mathbf{A_1, A_2, \ldots, A_{15}}$.

Пусть **i** – индекс (порядковый номер) элемента в массиве **A**.

Тогда $\mathbf{A_i}$ – **i**-й элемент массива A, где **i = 1, 2, …, 15**.

Для ввода массива можно использовать любой цикл.

1. ввод массива с использованием цикла с предусловием***:***

Program massiv_1;

const N=15;

```
Var i: integer;
    A: array [1..N] of Integer;
Begin
    i := 1;
    While i <= N Do
       Begin
         Read (A[i]);
         i := i + 1
        End;
End.
```

2. Ввод данных с клавиатуры

Цикл FOR:

```
for i:=1 to N do begin
write ('a[', i, ']=');
read ( a[i] );
end;
```

Массив будет заполнен следующим образом (красный цвет чисел – данные, введенные пользователем):

a[1] = 10
a[2] = 50
a[3] = 40
a[4] = 35
a[5] = 18

Обращение к определенному элементу массива осуществляется путем указания имени переменной массива и в квадратных скобках индекса элемента.

3. Применяя цикл FOR с помощью оператора присваивания:

```
program massiv_7;
uses crt;
const N=7;
var a: array[1..N] of integer;
i:  integer;
  begin
       for i:=1 to N do    begin
       a[i]:= i;
       writeln ('a[', i, ']=', i)
       end
  end.
```

Вот, как будет заполнен массив:

a[1] = 1
a[2] = 2
a[3] = 3
a[4] = 4
a[5] = 5
a[6] = 6
a[7] = 7

Вывод массива

Вывод одномерного массива осуществляется также поэлементно. Для вывода массива можно использовать любой цикл.

Например, выведем одномерный массив $X_1, X_2, \ldots, X_n$, состоящий из элементов вещественного типа.

Приведем два возможных способа вывода массива:

1) **For** i := 1 **To** n **Do** Write (X[i],' ')
2) **For** i := 1 **To** n **Do** Writeln (X[i])

Второй способ может показаться более простым и удобным, но это не всегда так: результат работы такой программы зачастую неудобно, а то и просто невозможно анализировать. Так как *каждый элемент массива будет располагаться в отдельной строке*, следовательно, более 25 элементов одновременно увидеть будет нельзя.

Очень часто массив требуется распечатать дважды, чтобы сравнить состояние массива до обработки и после его обработки.

В этом случае сравнение состояний массива гораздо удобнее проводить, если они распечатаны в двух соседних строках, а элементы выровнены по столбцам, то есть к варианту 1 должна быть добавлена еще и ***форматная печать*** (указано количество позиций, которое должно отводиться на печать одного элемента).

```
Program massiv_4;
Const n = 15;
Var i: Integer;
   X: Array [1..n] Of Real;
Begin
   For i:= 1 To n Do Write (X[i] : 6 : 2,  ' '); // для вывода указано 6
                                                  позиций, из них 2 позиции –
                                                  цифры дробной части числа
   Writeln; //курсор переводит на новую строку
End.
```

Задача: Дан целочисленный одномерный массив, состоящий из n элементов. Найти сумму и произведение четных элементов, кратных 4.

Введем обозначения:

n – количество элементов в массиве;

A – имя массива;

i – индекс элемента массива;

A_i – ***i***-й элемент массива ***A***;
S– сумма четных элементов массива, кратных 4;
Pr – произведение четных элементов массива, кратных 4.
Входные данные: n, A.
Выходные данные: S, Pr.

```
Program massiv_summa_pr;
Var A: Array[1..20] Of Integer;
    i, n, S, Pr: Integer;
Begin
 Write (‘n  =  ’); Readln (n); //пользователь указывает, какое количество
                чисел        нужно ввести и обработать программе
   For i:=1 To n Do Readln (A[i]);  //ввод массива
   S:= 0; Pr:=1;
   For i:=1 To n Do  //обработка массива
    If (A[i] mod 2=0) and (A[i] mod 4 = 0) Then //определение четности
                                                числа и кратности его 4
    Begin
      S:=S+A[i];
     Pr:= Pr*A[i]
    End;
    Writeln (‘S= ’, S, ‘  Pr =  ’, Pr);
    End.
```

Суммирование нужных элементов массива осуществляется по тому же принципу, что и суммирование значений простых переменных.

Определяется ячейка памяти (переменная S), в которой будет сохраняться результат последовательного суммирования значений;

В переменной S первоначально помещается 0, это число не влияет на результат сложения;

Значение каждого элемента массива поочередно складывается со значением переменной S, и записывается в переменную S.

Нахождение произведения значений элементов массива аналогично суммированию, с тем лишь отличием, что первоначально переменная, хранящая результат последовательного умножения (Pr) должна иметь значение 1, так как именно 1 не влияет на результат умножения.

Лабораторная работа 1

Цель: Формирование умения представлять алгоритмы в виде блок-схем для решения задач.

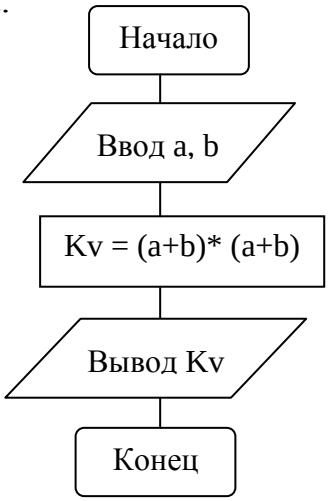
Рассмотрим задачу:

Задача: Вычислить квадрат суммы двух чисел по введенным значениям a и b. Составить алгоритм решения задачи с помощью блок-схемы.

Пример решения задачи средствами MS Word 2007:

В программе MS Word для составления блок-схемы воспользуемся вкладкой **Вставка → Фигуры → Блок-схема**. Выбираем нужный элемент и, удерживая левую кнопку мыши, рисуем нужную фигуру. Когда все фигуры готовы, соединяем их с помощью линий (**Вставка → Фигуры → Линия**). В каждый элемент блок-схемы можно вписывать данные. Для этого щелкнуть по фигуре и в контекстном меню выбрать **Изменить текст**. Если необходимо добавить надпись **Да** или **Нет** воспользуемся вкладкой **Вставка → Надпись**). Когда блок-схема будет полностью готова, ее необходимо сгруппировать: во вкладке **Выделить → Выбор объектов**, выделяем левой кнопкой мыши всю блок-схему и в контекстном меню выбираем **Группировать.**

Блок-схема готова:



Задание:

Составить алгоритмы для решения задач с помощью блок-схем средствами MS Word:

1. Вычислить площадь прямоугольника.
2. Вычислить сумму трех чисел.
3. Заданы числа a и b. Определить, эти числа одного или разных знаков? (используя операцию умножения).
4. Имеются три дыни различной массы (a, b, c). Как, пользуясь чашечными весами без гирь, путём не более трех взвешиваний расположить их по убыванию веса?

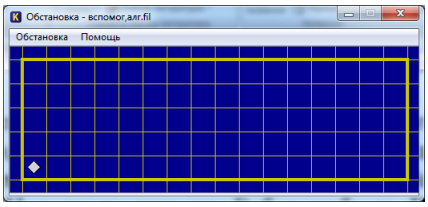
Лабораторная работа 2

Цель: изучение возможностей программирования с помощью учебного исполнителя Робот в среде КУМИР: составление линейных и циклических алгоритмов управления.

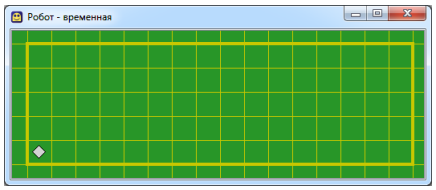
Порядок работы:

1. Задать ***стартовую обстановку*** по условию задачи:

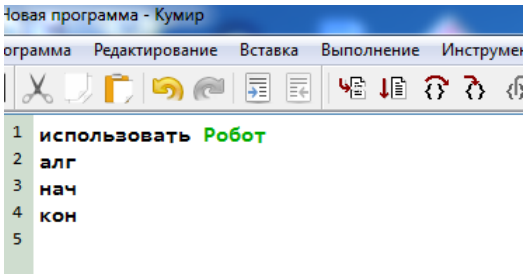
Меню **Инструменты → Редактировать стартовую обстановку Робота** (подготовить обстановку по условию задачи, выбрать в контекстном меню **Сохранить как стартовую**, дать имя файлу, сохранить в личной папке)



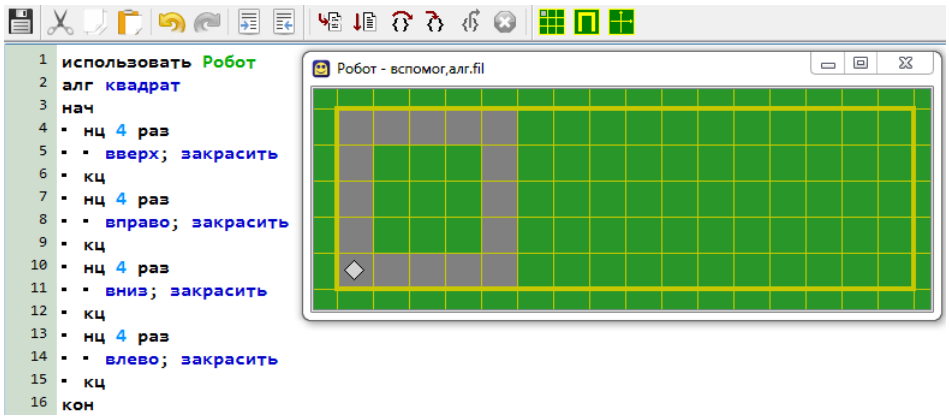
2. Нажать на вкладку Робот выбрать **Показать окно Робота** – появится клетчатое зеленое поле с подготовленной обстановкой.



3. Указать Исполнителя:
   Меню **Вставка → Использовать Робот** (подсвечивается зеленым цветом)

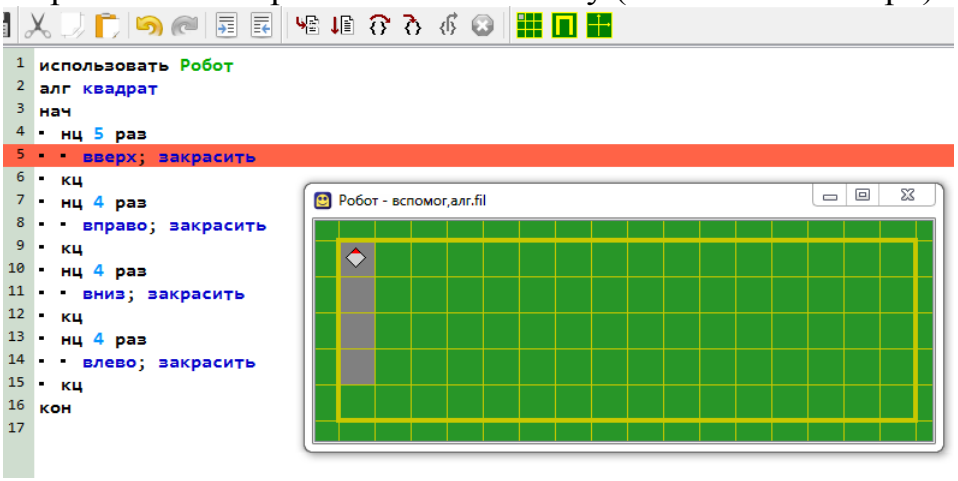


4. Написать алгоритм решения задачи.
5. Выполнить алгоритм (Меню **Выполнение→Выполнить непрерывно /F9** (или по шагам).

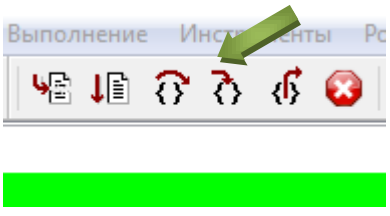


6. Если программа написана без ошибок, то сохранить программу: Меню Программа → Сохранить программу как… (указать папку, дать имя файлу, расширение файла .kum).

Ниже приведен пример алгоритма с ошибкой: программа завершается аварийно: Робот разбивается об стену (лишний шаг вверх).

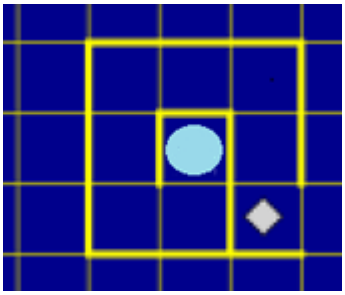


В этом случае нажать на пиктограмму **Шаг** и далее нажать на красную пиктограмму. Внести исправления в программу и проверить еще раз.

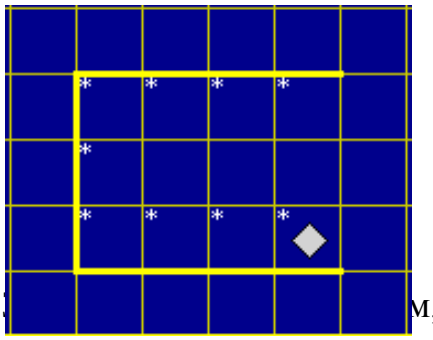


Задание:

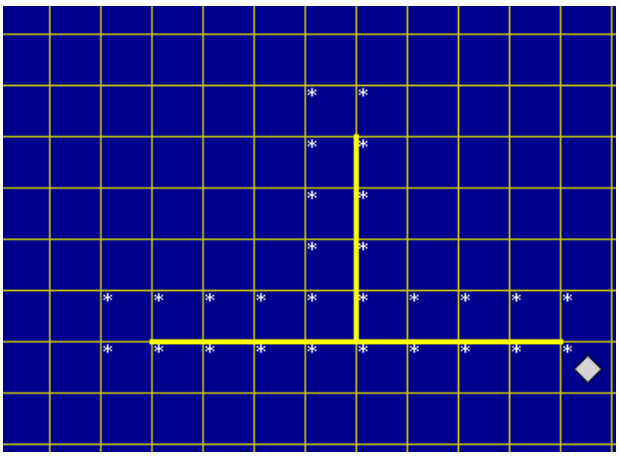
1. Необходимо перевести Робота из начального положения (◊) в центр лабиринта за минимальное число шагов любым из возможных способов.



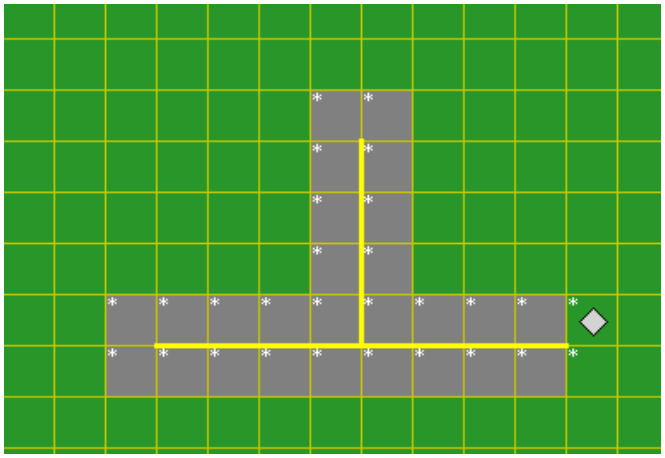
2. Составьте алгоритм, закрашивающий все внутренние клетки, прилегающие к стене (используйте циклический алгоритм).



м, закрашивающий все клетки вокруг стен (количество клеток, ограниченных стенами заранее неизвестно).



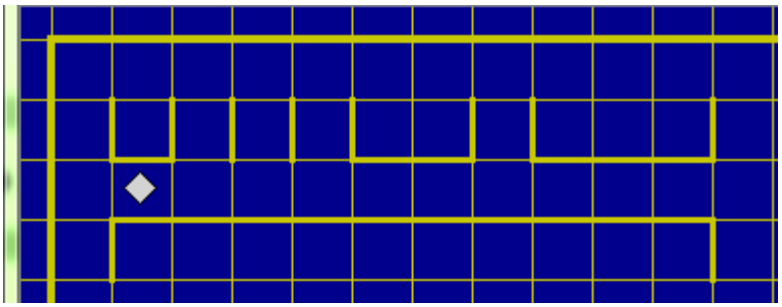
Вот, должно получиться:



Лабораторная работа 3

Цель: изучение возможностей программирования с помощью учебного исполнителя Робот в среде КУМИР: составление ветвящихся алгоритмов.

Задача: Закрасить все клетки коридора, из которых есть выход вверх, длина коридоров и количество выходов заранее точно не известны.

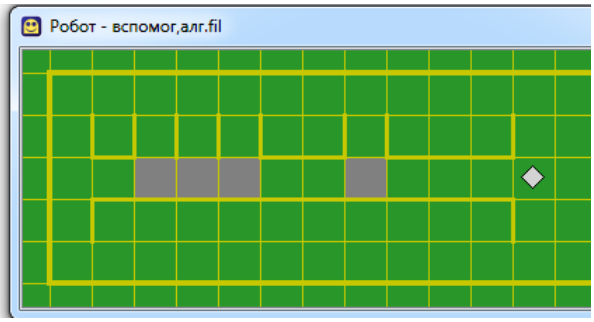


Решение данной задачи только с помощью цикла «пока» не представляется возможным (нужно закрасить только некоторые клетки), поэтому вспомним конструкцию «если» и пропишем, при каком условии Робот должен закрашивать клетки:

если сверху свободно
    то закрасить
все

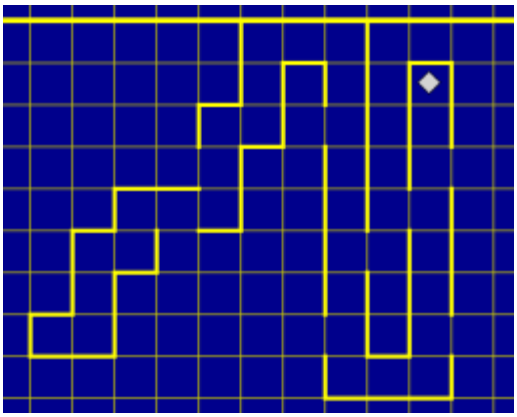
При данном условии Робот закрасит все клетки, имеющие выход вверх. Осталось с помощью цикла «пока» переместить исполнителя Робот вправо. Программа будет выглядеть так:

```
использовать Робот
алг коридор
нач
· нц пока снизу стена
· ·
· · если сверху свободно
· · · то закрасить
· · все
· · вправо
· кц
кон
```

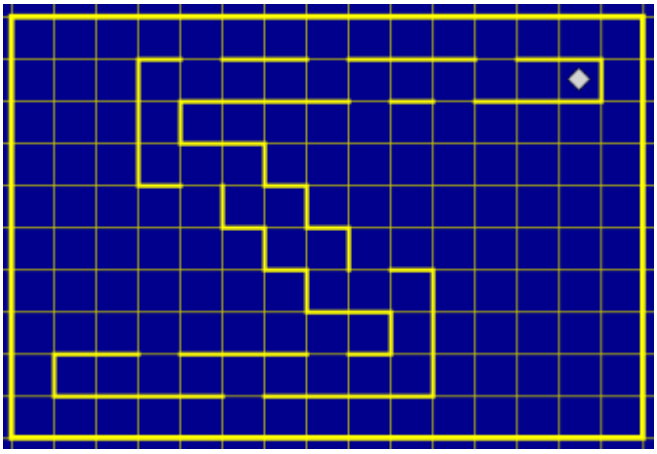


Задание:

1. Необходимо провести Робота по коридору шириной в одну клетку из начального положения (◊) до конца коридора, закрашивая при этом все клетки коридора, которые имеют выход. Выходы размером в одну клетку располагаются произвольно по всей длине коридора. Коридор заканчивается тупиком. Коридор имеет два вертикальных и диагональный участки в форме  . Пример коридора показан на рисунке:



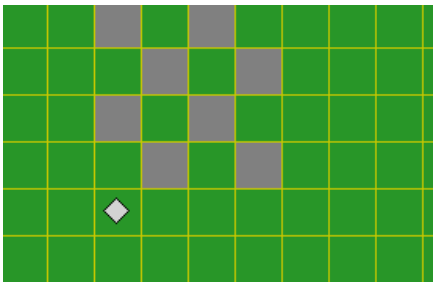
2. Необходимо провести Робота по коридору шириной в одну клетку из начального положения (◊) до конца коридора, закрашивая при этом все клетки коридора, которые имеют выход. Выходы размером в одну клетку располагаются произвольно по всей длине коридора. Коридор заканчивается тупиком. Коридор имеет два горизонтальных и диагональный участки в форме Σ. Пример коридора показан на рисунке:



Лабораторная работа 4

Цель: изучение возможностей программирования с помощью учебного исполнителя Робот в среде КУМИР: построение алгоритмов методом последовательного уточнения через составление вспомогательных алгоритмов.

Задача: Робот находится в верхнем левом углу поля. Стен и закрашенных клеток нет. Составьте алгоритм, который закрашивает в шахматном порядке квадрат 4 х 4. Конечное положение Робота может быть произвольным.



Рассмотрим более детально то, что должно получиться: можно выделить две линии: одна начинается с первой закрашенной клетки, а вторая - со второй:

Составим алгоритм для получения первой группы закрашенных клеток:

алг **первая**
нач
нц 2 раз
закрасить; вправо; вправо
кц
кон

Составим алгоритм для получения второй группы закрашенных клеток:

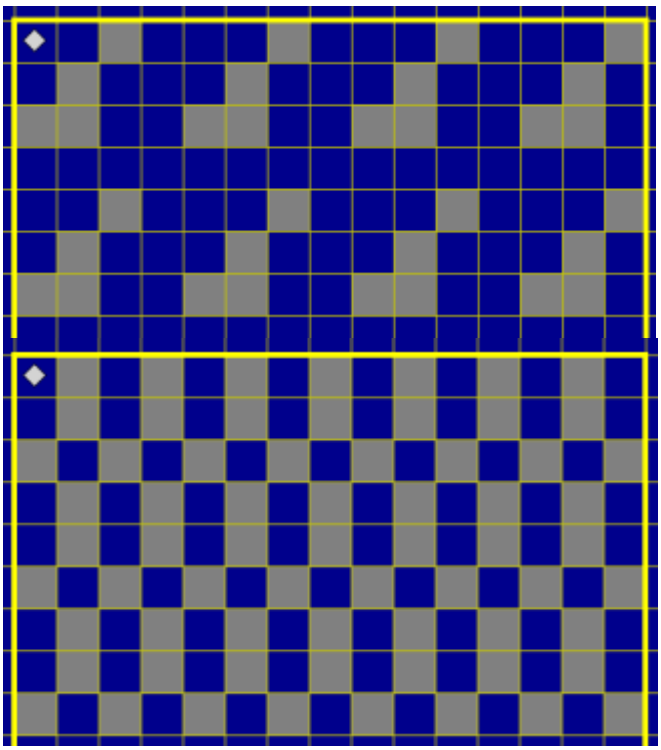
алг **вторая**
нач
нц 2 раз
закрасить; влево; влево
кц
кон

Этот алгоритм отличается от первого только направлением движения исполнителя – первую линию Робот закрашивает, двигаясь вправо, а вторую – двигаясь в обратном направлении. Для решения поставленной задачи нам необходимо использовать алгоритм **первая** 2 раза и алгоритм **вторая** 2 раза. Основной алгоритм будет обращаться к двум вспомогательным алгоритмам, записанным после слова **кон** в цикле «раз». Следует отметить, что после выполнения алг **первая** необходимо исполнителя перевести на клетку вниз и влево для выполнения следующего вспомогательного алг **вторая.** В целом получилась программа:

```
использовать Робот
алг узор
нач
· нц 2 раз
· · первая; вниз;влево;
· · вторая; вниз; вправо
· кц
кон
алг первая
нач
· нц 2 раз
· · закрасить; вправо;вправо
· кц
кон
алг вторая
нач
· нц 2 раз
· · закрасить; влево;влево
· кц
кон
```

Задание:

1. Используя вспомогательные алгоритмы, составьте алгоритм для закрашивания клеток, образующих число 1515.
2. Используя вспомогательный алгоритм (алгоритмы) нарисовать следующие узоры:



Лабораторная работа 5

Цель: изучение возможностей программирования с помощью среды программирования Pascal ABC: составление простых программ.

Программа на языке программирования Паскаль состоит из заголовка, описаний и операторов.

Program primer; //заголовок программы
Var список переменных: тип;
Begin //операторы
End.

Арифметический оператор присваивания:
<перемення>:=<арифметическое выражение>
Пример: z **:=** x * y;

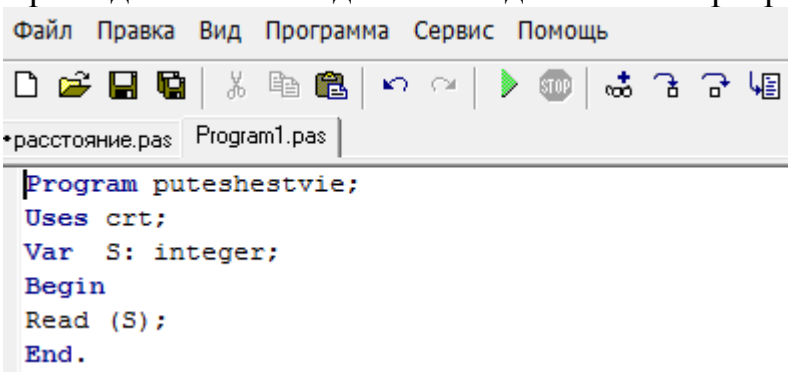
Оператор ввода данных:
Пример: read (x, y) //курсор остается на этой строке
readln (x, y) // курсор перемещается на новую строку (от англ. read line)

Оператор ввода данных:
Пример: **write** (x, '=', y) //на экране числа : 10=10
writeln ('Введите два целых числа') // на экране текст: Введите два целых числа
writeln; // оператор делает пропуск строки

Система программирования **Pascal ABC** - достаточно простая и дружественная среда программирования. Во время работы используются три основных окна: окно редактора кода, окно ввода и окно вывода.

Окно редактора кода

Большую часть рабочей области, её верхнюю часть занимает окно редактора кода. В него вводится исходный текст программы.



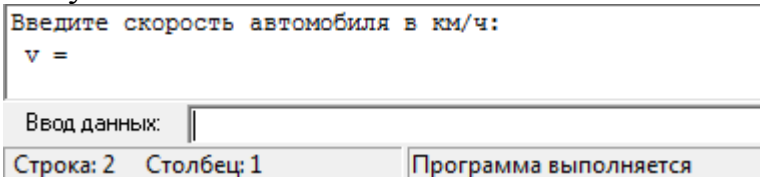
Окно вывода

Под окном редактора расположено окно вывода. Оно предназначено для вывода данных операторами **write** и **writeln**, а также для вывода сообщений об ошибках и предупреждений во время работы программы.

Окно вывода может быть скрыто. Клавиша F5 показывает/скрывает окно вывода. Для скрытия окна вывода используется также клавиша Esc.

Окно вывода обязательно открывается при любом выводе в него.

Для очистки окна вывода следует нажать комбинацию клавиш Ctrl+Del или кнопку ✕.



Не всегда удобно просматривать выводимые данные внизу программы, особенно числовые, в этом случае лучше применять модуль **Uses crt,** который записывается сразу после заголовка программы и позволяет при выполнении программы выводить данные отдельным окном.

Окно ввода

Окно ввода открывается при выполнении операторов **read** и **readln** в ходе работы программы:

Ввод данных в окно ввода сопровождается эхо-выводом в окно вывода. После нажатия клавиши Enter данные из окна ввода попадают в соответствующие переменные, окно ввода закрывается, и программа продолжает работать дальше.

Запуск и остановка программы

Для запуска программы в текущем окне редактора следует нажать клавишу F9 или кнопку  на панели инструментов.

Программа вначале компилируется во внутреннее представление, после этого, если не найдены ошибки, начинает выполняться. Выполнение программы можно в любой момент прервать нажатием комбинации клавиш Ctrl+F2 или кнопки  . При этом в окне вывода появится сообщение: Программа прервана пользователем.

Задача: Вычислите путь, пройденный автомобилем (S), если известно, что средняя скорость его движения была v км/ч, время в пути - t ч. Пример программы:

```
Program put;
Uses crt;
Var  S, v, t: integer;
Begin
    Writeln ('Введите скорость движения автомобиля в км/ч ');
    Write (' v = ');
    Readln (v);
    Writeln ('Введите время нахождения автомобиля в пути в ч ');
    Write (' t = ');
    Readln (t);
    S:= v*t;
    Writeln (Путь, пройденный автомобилем составляет,'  ',S, ' км');
End.
```

Задание:

1. Программу (Вычисление пути автомобиля) скопировать в среду Pascal ABC и выполнить ее.

2. Напишите программу, которая вычисляет площадь квадрата со стороной **a** (через дружественный диалог пользователя с компьютером).

3. Напишите программу, которая вычисляет сумму трех чисел (a, b, c).

Лабораторная работа 6

Цель: изучение возможностей программирования с помощью среды программирования Pascal ABC: составление линейных программ.

Integer – целый тип данных

Real – вещественный тип данных

Div – используется, чтобы получить целый результат деления целых чисел.

Mod - используется, чтобы получить целый результат остатка от деления целых чисел.

Арифметические функции:

Abs(x) - вычисляет модуль (абсолютную величину) числа x;

Cos(x) - вычисляет косинус угла x, заданный в радианах;

Sin(x) - вычисляет синус угла x, заданного в радианах;

Frac(x) - выделяет дробную часть числа x;

Int(x) - выделяет целую часть числа x;

Sqr(x) - вычисляет x^2;

Sqrt(x) - вычисляет $\sqrt{x}$.

Функции преобразования типов:

Round(x) - округляет вещественное число x до ближайшего целого;

Trunc(x) - выделяет целую часть числа x, отбрасывая дробную часть

Вспоминим структуру программы:

```
Program lin;
Uses crt;
Var  S: integer;
      Begin
        writeln ('Я умею программировать!');
      End.
```

Задание:

1. Напишите программу, которая вычисляет среднее арифметическое и среднее геометрическое двух чисел.

2. Напишите программу, которая позволяет пользователю ввести свой рост и вес. Выведите на экран сообщение:

Рост … сантиметров и вес … килограммов вам вполне подходят!

3. Используя оператор DIV, напишите программу, которая по двум введенным целым числам (a, b) выводит на экран сообщение:

*Результат деления числа **a** на число **b** = …*

4. С помощью оператора MOD напишите программу, которая по двум введенным целым числам (a, b) выводит на экран сообщение:

*Остаток от деления числа **a** на число **b** = …*

Лабораторная работа 7

Цель: изучение возможностей программирования с помощью среды программирования Pascal ABC: составление программ с разветвляющимися алгоритмами.

Условный оператор:

if <условие> **then begin** //что делать, если условие верно
end
else begin //что делать, если условие неверно
end;

Правила:

4. Перед **else не** ставится точка с запятой;
5. Вторая часть (**else** …) может отсутствовать;
6. Если в блоке один оператор, можно **не** использовать слова ***begin*** и ***end.***

Пример: Составить программу вычисления функции **y** для заданного значения x.

$$y = \begin{cases} e^x, \text{если } x < 0; \\ \cos x, \text{если } x > 0. \end{cases}$$

Для справки: **экспоне́нта** – показательная функция $f(x) = \exp(x) = e^x$, где e — число Эйлера

```
Program func;
Uses crt;
Var  x, y: real;
 Begin
      Writeln ('Введите x');
      Read (x);
      If x>0 then y:=cos(x)
       else
      If x<0  then y:=exp(x);
      Writeln ('Результат вычисления функции ');
      Writeln (‘ y=’, y:8);
 End.
```

Задание:

1. Составьте программу, входными данными в которой будут возраст Светы (переменная X) и возраст Димы (переменная Z). После выполнения программы на экран выводится одно из трех сообщений:
   Света старше Димы
   Дима старше Светы
   Свет и Дима - ровесники
2. Напишите программу, по которой на экран выводится сообщение: *«Сегодня рабочий день»* или *«Сегодня выходной»* в зависимости от введенного числа в диапазоне от 1 до 7.

3. Составьте программу определения наименьшего из трех введенных чисел.

Лабораторная работа 8

Цель: изучение возможностей программирования с помощью среды программирования Pascal ABC: составление программ с разветвляющимися алгоритмами (оператор множественного выбора).

Общая форма записи оператора выбора:

```
case порядковая_переменная of
значение1 : оператор (группа операторов);
значение2 : оператор (группа операторов);
. . . . . .
значениеN : оператор (группа операторов)
else оператор (группа операторов);
end;
```

Пример фрагмента программы, которая по введенному числу выводит на экран имя числительное:

```
if chislo = 10 then
   write ('Десять');
if chislo = 20 then
   write ('Двадцать');
if chislo = 30 then
   write ('Тридцать');
                …
```

Задание:

1. Вывести на экран, в зависимости от введенного пользователем числа (от 1 до 12), время года.
2. Написать программу, которая выводит на экран название геометрической фигуры в зависимости от введенного числа (например: 3 – треугольник, 4 – четырехугольник и. т. д. до 12). Предусмотрите случай, когда пользователь неверно вводит число.

Лабораторная работа 9

Цель: изучение возможностей программирования с помощью среды программирования Pascal ABC: составление программ с циклическими алгоритмами.

В языке программирования Паскаль существует циклы двух типов: циклы с условием и цикл с параметром (FOR). Циклы с условием делятся на циклы с предусловием (WHILE) и постусловием (REPEAT). Цикл с параметром задает параметр (условие выполнения цикла) сразу, цикл с предусловием проверяет работу цикла в начале, с постусловием - в конце.

Вспомним три вида циклических конструкций:

1. **For … To, For ... Downto**

Оператор **For** вызывает оператор, находящийся после слова Do, по одному разу для каждого значения в диапазоне от начального до конечного значений.

For переменная: = начальное **To** (или **Downto** для уменьшения) конечное **Do** оператор

2. **While** … **Do** оператор

Оператор после **Do** будет выполняться до тех пор, пока условие является истинным (True).

3. **Repeat**

операторы ...

Until // условие выхода из цикла (в случае выполнения логического условия происходит выход из цикла, в случае невыполнения – его повторение)

Задача: вычислить сумму ряда 1+1.5+2+2.5+3+3.5+ .. + 20.

Пример программы с оператором цикла **WHILE:**

```
program summa_riada;
uses crt;
var sum:real;
    n:real;
  BEGIN
    sum:=0;   n:=1;
    while n<=20 do
       begin
         sum:=sum+n;
         n:=n+0.5;
       end;
    writeln (‘Сумма ряда чисел от 1 до 20:  ', sum);
  END.
```

Задание:

1. Найдите сумму всех натуральных чисел из промежутка [1, n] (применяя циклический алгоритм).

2. Начав тренировки, спортсмен в первый день пробежал 10 км. Каждый день он увеличивал дневную норму на 10% нормы предыдущего дня. Какой суммарный путь пробежит спортсмен за 7 дней? Решить с помощью циклического алгоритма.

3. Составить алгоритм решения задачи: сколько можно купить быков, коров и телят, платя за быка 10 руб., за корову — 5 руб., а за теленка — 0,5 руб., если на 100 руб. надо купить 100 голов скота? Написать программу, используя циклический алгоритм.

Лабораторная работа 10

Цель: изучение возможностей программирования с помощью среды программирования Pascal ABC: составление программ с одномерными массивами.

В программировании линейная таблица называется одномерным массивом. **Одномерный массив** – это фиксированное количество однотипных элементов, объединенных одним именем, каждый элемент массива имеет свой уникальный номер, и номера элементов идут подряд.

Описание типа массива:

Type

Имя типа = **Array** [тип индекса (ов)] **Of** тип элементов;

Var

Имя переменной: имя типа;

Переменную типа массив можно описать сразу в разделе описания переменных **Var**:

Var Имя переменной: **array** [тип индекса (ов)] **Of** тип элементов;

Array – служебное слово (в переводе с английского означает «массив»);
Of – служебное слово (в переводе с английского означает «из»).
Тип элементов может быть любым, кроме файлового типа.

Задача: Найти сумму элементов одномерного массива. Размер произвольный. Элементы вводятся с клавиатуры.

```
Program summa;
Var a: array[1..30] of real; //задан массив с вещественным типом элементов
    i, n: integer;
     s: real; // переменная для сохранения суммы элементов вещественного типа
Begin
  Write (‘n=’); Readln (n); //вводим число, которое показывает, сколько элементов нужно будет ввести пользователю
  s:=0; // переменная хранит до начала цикла  0 – сумма пока не подсчитана
       For i:=1 to n do //оператор цикла с параметром
         begin
           write (‘введите число’); readln (a[i]); // в цикле пользователь вводит элементы массива – всего n элементов
           s:=s+a[i]; // в переменную накапливается сумма сначала 1-го элемента, затем 1-го и 2-го элемента и т. д. до n-го элемента, при этом предыдущее сохраненное значение суммы стирается
         end;
  writeln (‘сумма  всех чисел: ‘,s);  // вывод на экран суммы всех элементов
End.
```

Задание:

1. Задан массив A, состоящий из N чисел. Найти среднее арифметическое его элементов. Элементы вводятся с клавиатуры.
2. Найти сумму элементов массива с четными номерами, содержащего N элементов. Элементы вводятся с клавиатуры.

Список литературы

1. Босова Л.Л. Информатика: уч. для 9 кл. – 2-е изд., испр. – М.: БИНОМ. Лаборатория знаний, 2014
2. Камалова Н.А. http://festival.1september.ru/articles/594645/
3. Нурмухамедов Г.М. Информатика. Теоретические основы: учеб. Пос. для подготовки к ЕГЭ. –СПб.: БХВ-Петербург, 2012
4. Поляков К.Ю., Еремин Е.А. Информатика. Углубленный уровень: уч. для 10кл.: в 2 ч. Ч.1. – М.: БИНОМ. Лаборатория знаний, 2013
5. Семакин И.Г., Залогова Л.А., Русаков С.В. Информатика и ИКТ: уч. для 9 кл. – М.: БИНОМ. Лаборатория знаний, 2012
6. Сухих Н.А.Поурочные разработки по информатике: 9 класс. – М.: ВАКО, 2012.