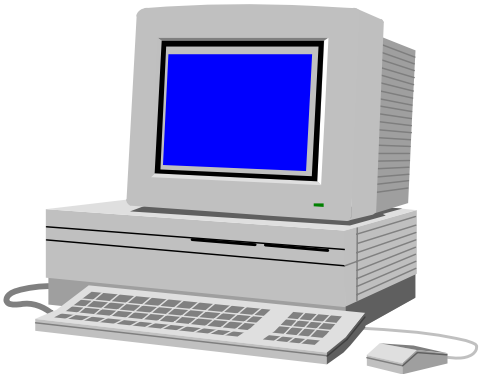
Средняя школа № 53

Основы информатики
и вычислительной техники



Основы алгоритмизации и программирования на языке Pascal

Большинство современных пользователей ПК для своей повседневной работы используют прикладное программное обеспечение (Word, Excel и т.п.), которое работает под управлением операционной системы (Windows). Все вышеперечисленные программы создаются профессиональными программистами с помощью **систем программирования** - совокупности программ, предназначенных для разработки новых программ. Для нормальной работы программист должен уметь создавать алгоритмы решения задач, используя широкий набор математических методов; а также знать язык программирования, на котором алгоритм записывается для компьютера. Этим вопросам и посвящено данное пособие.

АЛГОРИТМ. ПРОГРАММА

1. Алгоритм. Исполнитель

Термин "алгоритм" произошел от имени узбекского ученого Мухаммеда ибн Мусы ***Ал-Хорезми*** *(Магомет, сын Моисея из Хорезма), сформулировавшего кроме прочего алгоритмы четырех основных арифметических действий. В настоящее время теория алгоритмов – самостоятельная научная область, лежащая в основе вычислительных наук, и в частности - программирования.*

Алгоритм – это конечная последовательность точно определенных действий, приводящих к однозначному решению поставленной задачи.

Исполнитель - устройство, способное понимать и исполнять команды алгоритма.

Исполнитель обладает следующими свойствами:

- **Система команд исполнителя** (**СКИ**) - набор команд, который может выполнять данный исполнитель.
- **Среда исполнителя** - совокупность объектов, над которыми исполнитель может совершать действия.

Примеры исполнителей

Исполнитель	*СКИ*	*Среда*
Робот-манипулятор	Перемещение вверх, вниз; повороты вправо, влево; взять, отпустить деталь и т.п.	Конвейер
Человек	Широкий набор умственных и физических операций	Объекты окружающего мира
Система программирования Pascal	Команды языка программирования Pascal	Компьютер

2. Свойства алгоритмов

Любой алгоритм должен отвечать следующим требованиям:

1. **Понятность** - в алгоритме должны содержаться только те команды, которые входят в его СКИ. Чем меньшей СКИ владеет исполнитель, тем сложнее становится описание алгоритма решения некоторой задачи. Например для исполнителя-десятиклассника алгоритм решения квадратного уравнения должен выглядеть так: решить уравнение, потом сообщить результат, - так как в его СКИ должно входить умение решать подобные уравнения. А для исполнителя-семиклассника потребуется подробно расписать все этапы решения уравнений подобного типа, так как его СКИ еще не включает подобные знания.

2. **Дискретность** - дискретное (пошаговое) выполнение команд алгоритма с точной фиксацией выполнения одной команды и начала выполнения следующей.

Алгоритм нельзя описывать в виде сложноподчиненных предложений, которые могут запутать исполнителя. Например, для объяснения человеку алгоритма нахождения какого-либо здания в незнакомом городе лучше пошагово расписать последовательность его действий (две остановки на автобусе, подземный переход, 150 метров прямо), нежели путано расписывать эти операции, упоминая при этом красоты любимого города.

3. **Детерминированность** - алгоритм должен предусматривать определенный порядок выполнения действий. После исполнении одной команды исполнитель должен четко знать, какая следующая команда будет выполняться. Порядок выполнения действий очень важен: попробуйте, например, переставить местами действия в алгоритме перехода улицы: 1. Дождаться зеленого света. 2. Перейти улицу, - и Вы рискуете попасть под машину.

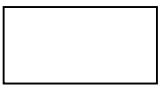
4. **Результативность** - алгоритм должен привести к необходимому конечному результату. Например, вышеупомянутый алгоритм перехода улицы может не привести к конечному результату, так как не предусматривает возможности отсутствия или поломки светофора.

5. **Массовость** - возможность применения данного алгоритма для решения многих задач одного типа. Какой смысл был бы в изучении алгоритма решения квадратного уравнения, если с его помощью решалось бы только одно уравнение?

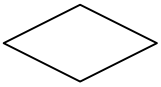
3. Описание алгоритмов

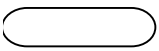
Алгоритмами пронизана вся наша жизнь: человек, часто не задумываясь об этом, выполняет сотни и тысячи разнообразных алгоритмов, некоторые из которых доведены до автоматизма (например, алгоритм завязывания шнурков на ботинках). Решая ту или иную задачу (по физике, химии или по жизни), мы всегда составляем логическую последовательность действий, приводящую нас к поставленной цели, т.е. разрабатываем алгоритм решения этой задачи. Объясняя другому человеку решение некоторой задачи, мы описываем его алгоритм на естественном человеческом языке (русском, английском и т.п.). Но человеческий язык малопригоден для точного описания алгоритмов: каждый человек описывает решение задачи по-своему, поэтому один и тот же алгоритм решения может быть описан разными людьми по-разному, т.е. мала возможность стандартизации записи алгоритма на человечеком языке.

Компьютер же является формальным исполнителем алгоритмов, поэтому алгоритм для него должен быть записан строго по определенным правилам. Для единообразия описания алгоритмов при решении задач на компьютере принято использовать **структурную блок-схему** - способ записи алгоритма с помощью графических блоков, основными из которых являются:


- операционный блок


- блок ввода и вывода информации

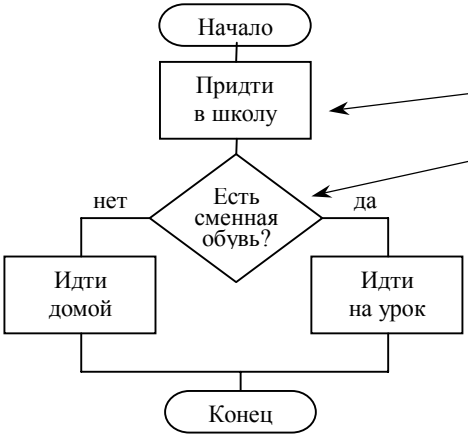

- блок проверки условия


- блок начала и конца алгоритма

4. Виды алгоритмов

1. **Линейный алгоритм** - алгоритм, который состоит из команд, выполняемых последовательно друг за другом.
Пример: алгоритм "Как перейти улицу"

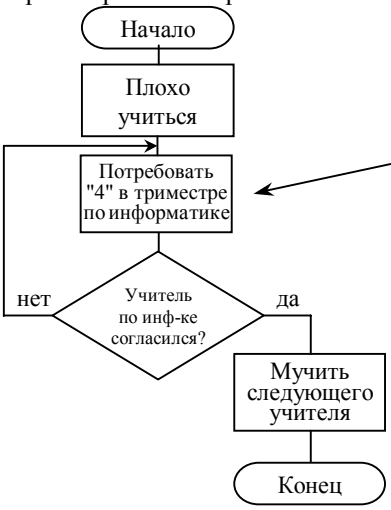
2. **Разветвленный алгоритм** - алгоритм, содержащий хотя бы одно условие, в результате проверки которого возможен переход на одну из двух ветвей.
Пример: алгоритм "Сменная обувь"



Утвердительное предложение

Вопросительное предложение, на которое может быть дан один из двух возможных вариантов ответа: «да» или «нет»

3. **Циклический алгоритм** - это разветвленный алгоритм, который содержит многократное повторение одного и того же действия.
Пример: алгоритм "Выпрашивание отметки".



Цикл – многократное повторение некоторого действия

5. Программа. Язык программирования

Программа – это запись алгоритма на языке, понятном исполнителю (компьютеру).

Программирование – процесс записи алгоритма на языке исполнителя.

Система программирования – это совокупность программ, предназначенных для создания программ на одном из языков программирования.

Язык программирования (ЯП) - это искусственно созданный язык для общения человека с компьютером.

Таким образом, алгоритм и соответствующая ему программа отражают одну и ту же последовательность действий, разница лишь в их представлении:

- Алгоритм - это формальная запись последовательности действий, некоторый план действий для исполнителя, отражающий логику решения задачи.
- Программа - это запись алгоритма для конкретного исполнителя (в частности компьютера), под управлением которой исполнитель решает поставленную задачу.

Например, алгоритм для робота-манипулятора – это запись на бумаге: повернуться вправо, взять деталь. Но формальная запись не заставит робота выполнить эти действия – нужна программа, которая в электронном виде отразит данные действия и приведет к движению механизма.

Применительно к компьютеру:

- сначала создается алгоритм решения задачи (в голове или переносится на бумагу);
- затем по алгоритму пишется программа, которая зависит не только от алгоритма, но и от ЯП, выбранного в качестве СКИ; по одному алгоритму может быть написано несколько соответствующих ему программ на разных ЯП;
- готовая программа вводится в компьютер, который по ней осуществляет заданные действия.

6. История развития языков программирования

1. **Язык машинных кодов** (1946).

Для полноценного общения с компьютером необходимо знать его язык - язык машинных кодов, алфавит которого состоит всего из двух символов: 0 и 1. На первом этапе развития ЭВМ такой способ общения с компьютером был единственно возможным: инженер-программист с помощью специального устройства набивал отверстия на перфокарте или перфоленте; эти отверстия и являлись цифровыми кодами команд, которые должен выполнить центральный процессор. Машина отвечала взаимностью - выдавала перфокарту с другими отверстиями.

Работа с машинными кодами, при которой вся программа состояла из огромной последовательности нулей и единиц, требовала не только высочайшей квалификации пользователя (знание структуры ЭВМ и кодировок всех команд), но и незаурядных усидчивости и аккуратности. Поэтому работа на ЭВМ того времени была уделом немногих высококвалифицированных инженеров.

2. **Ассемблеры** (1949) - ЯП, в которых двоичные коды машинных команд заменяются соответствующими им 2-3 буквенными обозначениями. Например, вместо записи длинной последовательности 0 и 1, отражающей операцию сложения, достаточно записать ADD. Очевидно, что ассемблеры жестко привязаны к конкретному типу ЭВМ, так как отражают характерные только для данных машин коды команд. Поэтому для разных типов ЭВМ существуют разные ассемблеры.

3. **ЯП высокого уровня** - ЯП, позволяющие программисту оперировать обычными математическими конструкциями и определенными сокращениями английских слов. Например, вместо команды ассемблера ADD языки высокого уровня позволяют написать нормальное математическое выражение со знаком "+", например, a := b + c. ЯП высокого уровня близки к естественному человеческому (английскому) языку, что позволяет намного повысить скорость разработки программ и снизить вероятность появления ошибок.

Приведем примеры ЯП высокого уровня в их исторической ретроспективе:

- 1952 – Автокод. Первый ЯП высокого уровня; однако его автор (Глен), являясь участником секретного ядерного проекта, не смог сделать его всеобщим достоянием.
- 1956 - FORTRAN (FORmula TRANslator). Первый завоевавший всемирную популярность ЯП высокого уровня; разработан специалистами IBM и до сих пор является одним из самых удобных ЯП для научно-технических расчетов. Большинство математических расчетов в советском авиа- и ракетостроении было выполнено на этом языке.
- 1958 - ALGOL (ALGOrithmic Language). Не дошел до нашего времени, но его идеи были перенесены в более современные языки программирования.
- 1960 - COBOL (COmmon Business Oriented Language).

Используется для разработки приложений, связанных с экономикой и финансами.

- 1964 - BASIC (Beginners All-Purpose Symbolic Code).

Создан в Дартмутском колледже для обучения программированию начинающих.

- 1970 - Pascal (Никлаус Вирт). Используется как для обучения методам современного программирования, так и для профессионального программирования.
- 1972 - C (Деннис Ричи) обрел широкую популярность как ЯП "среднего уровня", в котором удобство ЯП высокого уровня сочетается с возможностью ассемлера по непосредственному доступу к аппаратным средствам компьютера. Это обеспечило ему лидерство в индустрии разработки программного обеспечения. Дальнейшее развитие этого ЯП (C++) на сегодня является лидером среди систем программирования

Вышеперечисленные ЯП являются вехами развития систем программирования. А всего за 60 лет существования ЭВМ появилось более 1000 различных ЯП. Сегодня наиболее перспективными являются созданные на основе классических ЯП Visual-среды, максимально использующие современные достижения в области графического интерфейса: Visual Basic, Visual C++, C++Builder, Visual J++, Delphi и т.д.

Выводы: пользователь знаком с одним из естественных человеческих языков, компьютер работает на основе машинного языка - языка двоичных кодов (0 и 1) Для организации общения между пользователем и компьютером создаются ЯП, которые являются компромиссом между языком человека и языком машины. По степени близости ЯП к человеку или машине выделяют:

1. **Языки программирования низкого уровня (машиноориентированные):** ассемблеры.
2. **Языки программирования высокого уровня (машинонезависимые)**: Basic, Pascal, C и другие.

Чем ниже уровень языка (ассемблеры), т.е. чем ближе

он к машине, тем эффективнее создаваемые в нем программы, так как в них полностью учитывается специфика данного компьютера. Однако работать с ассемблерами весьма затруднительно, нерационально расходуется время программиста. Поэтому наибольшей эффективностью обладают ЯП высокого уровня, в которых с одной стороны хорошо видна логика программы, с другой - есть возможность непосредственного обращения к аппаратной части компьютера.

7. Структура системы программирования

Для создания программ и их нормальной работы любая система программирования включает в себя следующие программы:

1. **Текстовый (экранный) редактор** – программа, предназначенная для набора и редактирования текста создаваемой программы на основе данного ЯП.
2. **Транслятор** – программа, которая переводит исходную программу, набранную в текстовом редакторе в соответствии с правилами данного ЯП, в машинную программу – программу в машинных кодах. Существует два типа трансляторов:
   - **Компилятор** - транслятор, который сначала полностью переводит исходную программу в машинную программу, которая затем выполняется (Pascal, C).
   - **Интерпретатор** - транслятор, который переводит и выполняет исходную программу покомандно (Basic).
3. **Отладчик** – программа, предназначенная для выявления ошибок в исходном тексте программы. Выделяют следующие типы ошибок:
   - ***Синтаксическая ошибка*** - нарушение правил написания предложений данного языка программирования; выявляется при компиляции программы.
   - ***Семантическая ошибка*** - недопустимые значения и действия над параметрами программы; выявляется уже во время работы программы (на этапе ее отладки).
   - ***Логическая ошибка*** - неправильная программная реализация данного алгоритма; наиболее трудна для обнаружения, так как нарушений в работе программы не вызывает, но приводит к неправильным результатам; выявляется в процессе тестирования программы.

Для исправления ошибки необходимо выявить факт наличия ошибки, определить ее местонахождение и исправить. Синтаксические ошибки выявляются с помощью компилятора, который указывает тип и место ошибки. Для исправления семантических и логических ошибок, найти которые в тексте программы бывает очень сложно, используется специальная программа – отладчик, позволяющий программисту быстрее найти и устранить ошибку. Отладчик позволяет выполнять следующие действия: пошаговое исполнение программы (***трассировка***), получение значений любых параметров программы и их изменение и т.п.

4. **Компоновщик** – программа, объединяющая отдельные части программы путем установления между ними необходимых связей и добавления стандартных подпрограмм.

Система программирования TurboPascal

8. Основные правила работы с TurboPascal

- **File / Open / osnova.pas** - открытие “скелета” программы в начале работы.
- Запуск программы на выполнение - **Run / Run** (**Ctrl + F9**).

➢ Для быстрого перемещения курсора бывают полезны следующие клавиши:
- **Home, End** - в начало или конец строки.
- **Ctrl + ←**, **Ctrl + →** - на одно "слово" влево или вправо.

➢ Смена режимов вставки-замены (надписи внизу окна Insert / Overwrite) – **Insert**.

➢ Чаще всего при завершении работы будет необходимость закрыть программу без сохранения: **File / Exit (Alt + F4)**; на вопрос "<имя программы> has been modified. Save?" щелкнуть по кнопке **No**.

➢ Для копирования, перемещения и удаления фрагмента программы используется меню **Edit**: Вырезать - **Cut** (**Shift + Delete**); Копировать - **Copy** (**Ctrl + Insert**); Вставить - **Paste** (**Shift + Insert**); Удалить - **Clear** (**Ctrl + Delete**). До данных операций необходимо выделить фрагмента текста программы: мышью или при помощи клавиатуры (стрелками при нажатой клавише **Shift**).

➢ Отмена последних операций: **Edit / Undo** (**Alt + Backspace**).

➢ **Исправление синтаксических ошибок**

При выдаче сигнала об ошибке, чаще всего, курсор находится около места ошибки. Наиболее часто встречающиеся синтаксические ошибки:
- **";" expected**. - отсутствует точка с запятой, разделяющая команды программы;
- **Unknown identifier** - неизвестный идентификатор: неправильно записана команда;
- **THEN expected; TO or DOWNTO expected; DO expected** и т.п. - отсутствует необходимое слово в операторе.

9. Алфавит языка

Алфавит - совокупность допустимых в языке символов и групп символов, рассматриваемых как единое целое. В алфавит языка Pascal входят:

1. **Латинские (английские) буквы** A - Z: запись всех команд программы (строчные и прописные буквы при этом не различаются).
2. **Цифры 0 - 9**. Математическая запись чисел (запись чисел на Pascal): 11 (11); -9 (-9); 3,5 (3.5); 0,000617 (0.000617 или .000617 или 6.17E-4); 310000 (310000 или 3.1E+5).
3. **Знаки операций**: + - * /
   mod - остаток от деления целых чисел: 14 mod 5 = 4
   div - целая часть от деления целых чисел 14 div 5 = 2
4. **Знаки отношения**: = > < >= <= <>
5. **Служебные символы**, в т.ч. **;** - разделение операторов;
   , - разделение элементов списка;
   { } - скобки комментария и т.д.

➢ **Зарезервированное слово** – слово языка, которые можно использовать только по своему прямому назначению, например begin, end, const, var, label, while, repeat, for и т.д. Его, в частности, нельзя использовать в качестве имени переменной.

➢ **Русские буквы** А – Я можно использовать только в пояснительных текстах.

10. Структура программы

➢ Заголовок программы должен отражать ее суть, но для экономии времени при наборе программы может быть опущен.

➢ До тела программы все обрабатываемые в нем объекты должны быть описаны (объявлены) в разделе описаний.

Структура программы	Пример
• Заголовок программы	**program password ;**
1. Раздел описаний: - описание библиотеки crt – подключение дополнит. возможностей языка; - описание используемых в программе - меток; - констант; - типов; - переменных; - процедур и функций.	**uses crt ;** **label 1 ;** **var x: string;**
2. Тело программы Начинается с оператора begin Последовательность операторов Заканчивается оператором end.	**begin** **clrscr ;** **1: write ('Введите пароль');** **read (x) ;** **if x='школа' then write('Проходи') else goto 1;** **repeat until keypressed;** **end.**

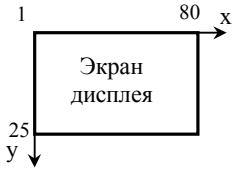
➢ Алгоритм решения задачи записывается в теле программы посредством **операторов** - зарезервированных слов, отражающих те или иные алгоритмические конструкции.

➢ Оформление программы. Программа может быть записана в произвольном виде – компилятор не обращает внимания на оформление. Однако, для наглядности и читабельности программы необходимо следовать следующим правилам:

- на одной строке должна находиться некоторая законченная конструкция языка;
- в теле программы операторы, входящие в состав других операторов, должны быть смещены при написании вправо (~ двух пробелов).

11. Вывод информации

Существует два режима работы: графический и символьный. В символьном режиме экран монитора разбивается на знакоместа, в каждом из которых может находиться один символ. Каждое знакоместо можно задать координатами: по горизонтали (1 - 80) и по вертикали (1 - 25).



Операторы вывода информации на экран дисплея

➢ **WRITE** и **WRITELN** – вывод информации, начиная с текущего знакоместа (writeln кроме этого дополнительно переводит курсор в начало следующей строки). Выводимая информация заключается в круглые скобки, а текст должен дополнительно браться в штрих-кавычки (апострофы).

➢ **GOTOXY (x, y)** - установка текущего знакоместа с координатами (x, y). Если оператор gotoxy отсутствует, но в начале программы находится оператор **CLRSCR** (очистка экрана), то координаты текущего знакоместа (1,1), т.е. информация начинает выводиться с верхнего левого угла экрана.

Вывести по центру чистого экрана красным цветом на синем фоне слово "Паскаль".

```
program write1;
  uses crt;
begin  clrscr;
  gotoxy (36,12);
  textcolor (4); textbackground (1);
  write ('Паскаль');
end.
```

12. Запись арифметических выражений

Запись арифметических выражений осуществляется в соответствие с правилами арифметики. Например, для вычисления значения выражения $\frac{1{,}15 \cdot 3{,}3 + 3}{230}$ необходимо набрать (1.15*3.3+3) / 230.

При получении нецелого числа компьютер выдает результат в нормальном виде (для вышеуказанного примера: 2.9543478261E-02, что означает 0,029543478261). Обычно такая точность вычисления является излишней. Более наглядный способ вывода нецелых чисел с точностью до N знаков после запятой:

WRITE (<арифметическое выражение>:0:N).

Например, WRITE ((1.15*3.3+3)/230 : 0 : 2) даст результат 0.03.

При записи арифметических выражений могут быть использованы следующие **стандартные арифметические функции**:

Функция	Назначение	Пример	
		запись	значение
PI	3.14159...	2*PI	6.28
SQR	квадрат	SQR (9)	81
SQRT	квадратный корень	SQRT (9)	3
SIN	синус	SIN (PI/4)	0.71
COS	косинус	COS (PI/3)	0.5
ABS	модуль	ABS (-5.3)	5.3
FRAC	дробная часть числа	FRAC (13.79)	0.79
INT, TRUNC	целая часть числа	INT (13.79)	13
ROUND	округление числа	ROUND (13.79)	14

При записи арифметических выражений надо помнить, что:

- Отрицательные числа необходимо заключать в скобки.
- Аргумент тригонометрических функций должен выражаться не в градусной, а в радианной мере. Например, для нахождения cos 60^o нужно набирать не ~~cos (60)~~, а cos (pi/3). Общая формула для записи α^o: PI * α / 180.

Например, cos 46^o должен быть записан как cos (pi*46/180).

- Пример записи сложного арифметического выражения: $21 + \frac{35{,}8^2 - \sqrt{7}}{1 + \sin 34^0} \cdot 2{,}3$

write (21 + (sqr(35.8) - sqrt(7)) / (1 + sin(pi*34/180))*2.3 :0:2)

13. Величины и их типы

Величина - элемент данных, хранящийся в памяти под определенным именем. В процессе выполнения программы величины могут менять свои значения (переменные) или быть постоянными (константы).

Константа – величина, не меняющая своего значения в процессе работы программы. Используемые в программе константы описываются в разделе описаний после зарезервированного слова **CONST**. Например, const g = 9.81; e = 1.6E-19.

Pascal предоставляет возможность обращаться и к стандартным константам, например: maxint = 32767, maxlongint = 2147483647.

Переменная – величина, которая может изменяться в процессе работы программы. Используемые в программе переменные описываются в разделе описаний после зарезервированного слова **VAR**, где также указывается тип, к которому принадлежит переменная. Например, var x, y: integer; z: real; s: string;

Каждая величина имеет 3 основных характеристики: имя, тип и значение.

Имя (идентификатор) величины может состоять не только из одной английской буквы, как принято в математике, а содержать до 63-х английских букв и цифр.

При создании сложных программ весьма вероятна путаница в именах величин (например, какая переменная за что отвечает?), что часто приводит к ошибкам даже у опытных программистов. Поэтому имена величинам стараются давать в соответствие с хранимой в них информацией. Например, переменной для указания возраста человека можно дать имя age, что соответствует английскому слову "возраст". При наборе небольших программ, с которыми Вы столкнетесь в школе, данный фактор "логичного именования" величин не является существенным. Однако, если в дальнейшем Вы планируете заниматься программированием - надо соблюдать это правило. Правда это связано с потерей времени при наборе текста программы: например, легче набрать имя переменной s, нежели summa.

Тип величины – характеристика величины, определяющая множество ее допустимых значений и операций над нею.

Стандартный тип – тип, предопределенный (не требующий описания) в данной системе программирования.

Основные стандартные типы величин

Тип	Обозначение	Множество допустимых операций	Диапазон допустимых значений	Объем памяти для хранения величины (байт)
Целый	**INTEGER**	Арифметические операции, операции сравнения, арифметические функции	-32768 .. 32767	2
	LONGINT		-2147483648 .. 2147483647	4
Вещественный	**REAL**		$\pm 2.9 \cdot 10^{-39}$ .. $1.7 \cdot 10^{38}$	6
Символьный	**CHAR**	Операции сравнения	1 символ	1
Строковый	**STRING**	Строковые функции	n (0 .. 126) символов	n
Логический	**BOOLEAN**	Логические операции	true, false	1

Кроме вышеупомянутых основных стандартных типов величин в ЯП Pascal существуют и другие. Так например, к целым типам величин также относятся: BYTE (0 .. 255; 1 байт), SHORTINT (-128 .. 127; 1 байт), WORD (0 .. 65535; 2 байта). Выбор типа величины определяется компромиссом между требуемым диапазоном ее значений и затратами памяти на ее хранение. Самым "мощным" вещественным типом является EXTENDED, с помощью которого могут быть представлены числа в диапазоне $\pm 3.4 \cdot 10^{-4932}$ .. $1.1 \cdot 10^{4932}$.

Правильное определение и использование типов величин крайне важно в программе:

- Если в разделе описаний объявляется переменная одного типа, а используется она в разделе операторов как переменная другого типа, компьютер прекратит выполнение

программы и выдаст сообщение об ошибке типов (семантическая ошибка): Type mismatch или Invalid numerical format.

➢ Особое внимание обратим на работу с числовыми типами:

- К типам, работающим с числами, относятся целые типы (integer и longint) и вещественный тип (real). Если в условии задачи не оговорен тип вводимых числовых данных, то необходимо относить их к более общему вещественному типу: он позволяет работать как с нецелыми, так и с целыми числами.
- Если значение переменной выходит за пределы допустимого диапазона, сообщение об ошибке не появится, но результат будет неверным.
- При выполнение арифметических операций следует помнить, что результат целого типа дают только следующие операции: сложение, вычитание и умножение переменных целого типа, а также операции div и mod. Результат деления целых чисел, даже заведомо целый, является величиной вещественного типа.
- При работе с арифметическими функциями следует помнить, что их аргументы могут относиться к любому числовому типу, а результат почти всегда относится к вещественному типу, кроме abs и sqr (тип результата совпадает с типом аргумента), а также trunc и round (результат целого типа).

14. Оператор ввода информации

READ (< имена вводимых переменных >)

Используемые в программе переменные должны быть объявлены в разделе описаний: **VAR** <имя переменной> **:** <тип переменной>.

Для более наглядного ввода информации в программе перед оператором read ставится оператор write с подсказывающим текстом.

Найти класс, в котором учится школьник по вводимому его возрасту.

```
program in_out;
  uses crt;
  const age0 = 6;
  var age: integer;
begin clrscr;
  write ('Возраст = '); read (age);
  write(Ты учишься в', age-age0, ' классе!');
end.
```

15. Оператор присваивания значения переменной

<имя переменной> **: =** <выражение, значение которого присваивается переменной>

Например:

a:=21; b:=5+a	{a = 21; b = 26}
a:=5; b:=2; a:=b	{a = 2; b = 2}
a:=5; b:=2; b:=a	{a = 5; b = 5}
a:=a+1	{a ↑ на 1}

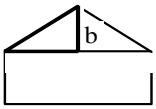
Найти целую и дробную часть среднего арифметического двух целых чисел.

```
program let_sa;
  uses crt;
  var a, b, c: integer;  sa, os: real;
begin  clrscr;
  write ('1 число = '); read (a);
  write ('2 число = '); read (b);
  sa := (a + b)/2;
  c := trunc(sa); os := frac(sa);
  write ('целая часть =',c,' дробная=',os:0:2);
end.
```

➢ Оператор присваивания не аналогичен простому арифметическому равенству. Например, математическая запись a = a + 1 является ложной при любых значениях a. А в программах запись типа a := a + 1 встречается достаточно часто; ее необходимо читать следующим образом: переменной с именем a присвоить значение суммы числа, которое было значением этой переменной до этого, и единицы.

➢ Слева от := находится имя переменной, а справа – значение, которое должно соответствовать типу, к которому данная переменная была отнесена в разделе описаний.

16. Этапы решения задачи на компьютере



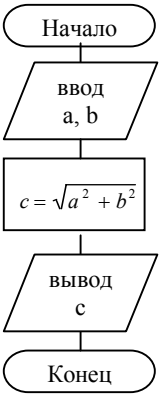
1. **Постановка задачи**.

Необходимо измерить длину стропила крыши (c), считая что прямое измерение затруднительно по техническим причинам.

2. **Определение метода решения задачи** - создание математической модели.

Очевидно, что измерение величин отрезков a и b не составляет большого труда. А длину стропила можно найти как гипотенузу прямоугольного треугольника по теореме Пифагора: $c = \sqrt{a^2 + b^2}$

3. Разработка алгоритма решения задачи



4. Создание программы по алгоритму

```
program gipot;
  uses crt;
  var a,b,c: real;

begin clrscr;

  write ('a = '); read (a);
  write ('b = '); read (b);


  c := sqrt ((sqr(a) + sqr(b));


  write ('результат = ', c :0:2);


end.
```

5. **Ввод программы в компьютер**: набор с клавиатуры или загрузка готовой программы.
6. **Отладка программы:** исправление синтаксических и семантических ошибок.
7. **Тестирование программы** - составление теста, с помощью которого можно было бы убедиться в правильности работы программы. Например, при вводе a = 3, b = 4 гипотенуза должна получиться $\sqrt{3^2 + 4^2} = 5$. Если программа выдала такой же ответ, ее можно считать правильной. Таким образом, на этапе тестирования выявляются и устраняются логические ошибки.
8. **Ввод данных и получение конечного результата**.

17. Оператор безусловного перехода

GOTO < метка >: - переход к выполнению оператора, на котором находится метка. Метки могут быть целым числом в пределах от 0 до 9999 или обычным идентификатором. Все используемые в программе метки должны быть перечислены в разделе описаний: **LABEL** < имена меток >

Вывести на экран бесконечный числовой ряд, состоящий из натуральных чисел с временной задержкой 0,5 секунды.

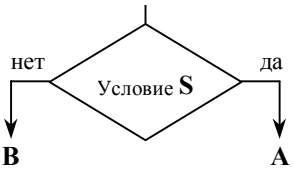
```
program goto_metka;
  uses crt;
  label 1;
  var i: integer;
begin clrscr;
  i:=0;
  1: i:=i+1;
  write (i, ' '); delay (500);
  goto 1;
end.
```

18. Оператор условия

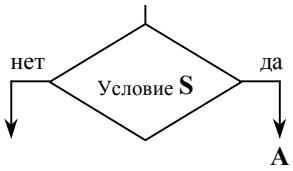
Для программирования разветвленных алгоритмов используется оператор условия (оператор условного перехода или условный оператор), который осуществляет алгоритмический блок проверки условия, определяя порядок выполнения операторов в зависимости от истинности некоторого условия.

Существует два формата оператора условия: полный и неполный.

Полный формат оператора условия
IF S **THEN** A **ELSE** B



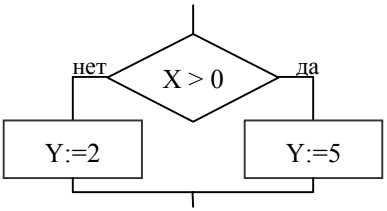
Неполный формат оператора условия
IF S **THEN** A



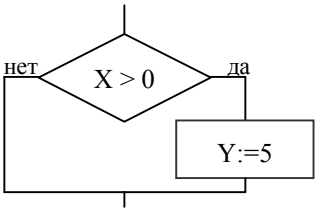
Здесь S - логическое выражение, истинность которого проверяется;
A - оператор, который выполняется, если выражение S истинно;
B - оператор, который выполняется, если выражение S ложно.

Примеры алгоритмических блок-схем и реализующих их операторов условия

IF X>0 **THEN** Y=5 **ELSE** Y=2



IF X>0 **THEN** Y=5



19. Программирование разветвленных алгоритмов

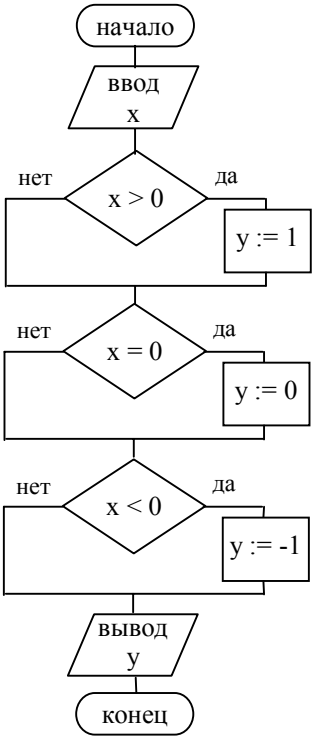
19.1 Вложенный и составной операторы условия

➢ Один оператор условия позволяет осуществить переход на одну из двух ветвей алгоритма. Реально, даже в несложных задачах, приходится программировать большее количество ветвей – в этом случае надо использовать несколько операторов условия. В том числе часто приходится применять **вложенный оператор условия** - оператор условия, размещенный внутри другого оператора условия.

➢ **Составной оператор** – линейная последовательность операторов, заключенная в операторные скобки: между словами begin и end;

Если к одной из ветвей алгоритма, между которыми осуществляет выбор оператор условия (т.е. к слову THEN или слову ELSE), относится более одного оператора, то всю эту группу операторов необходимо преобразовать в составной оператор, т.е. заключить между begin и end; .

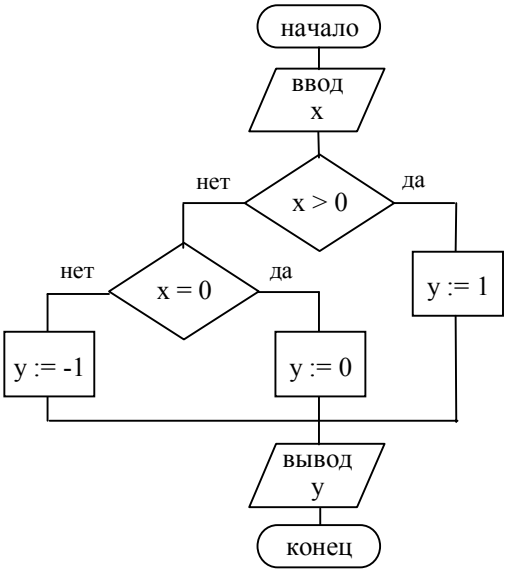
Пример. По вводимому значению x найти $y = \begin{cases} 1, \text{если } x > 0 \\ 0, \text{если } x = 0 \\ -1, \text{если } x < 0 \end{cases}$



Неполный формат условного оператора

```
program if_then;
  uses crt;
  var x: real; y: integer;
begin clrscr
  write ('x = '); read (x);
  if x>0 then y:=1;
  if x=0 then y:=0;
  if x<0 then y:=-1;
  write ('y = ', y);
end.
```

Обратите внимание, что разные форматы оператора условия соответствуют разным алгоритмам (блок-схемам).



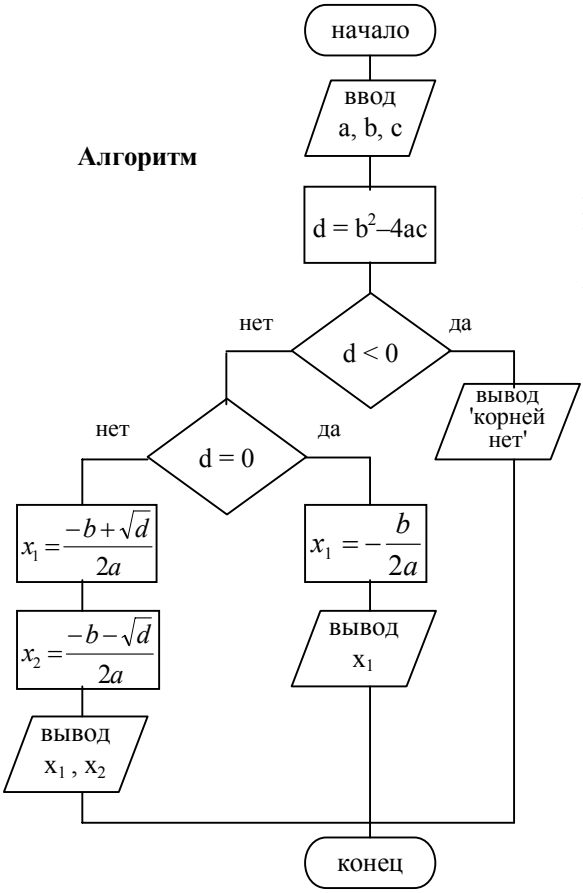
Полный формат условного оператора

```
program if_then_else;
  uses crt;
  var x: real; y: integer;
begin clrscr
  write ('x = '); read (x);
  if x>0 then y:=1 else
    if x=0 then y:=0  else y:=-1;
  write ('y = ', y);
end.
```

Нет необходимости проверять третье условие так, как это делается при использовании неполного формата условного оператора: очевидно, что если на вопросы x > 0 и x = 0 получен ответ "нет", то x < 0, что соответствует y = -1.

Пример.
Найти корни квадратного уравнения вида $ax^2+bx+c=0$ по вводимым a, b и c.

Алгоритм



Программа

```
program kw_ur;
uses crt;
 var a,b,c,d,x1,x2: real;
begin clrscr;
 write ('a = '); read (a);
 write ('b = '); read (b);
 write ('c = '); read (c);
 d:=sqr(b) - 4*a*c;
 if d<0 then write ('корней нет') else
   if d=0 then
    begin
      x1:=-b/(2*a); write ('x1=',x1:0:2)
    end
          else
    begin
      x1:=(-b+sqrt(d))/(2*a);
      x2:=(-b-sqrt(d))/(2*a);
      writeln ('x1 = ',x1:0:2);
      write ('x2 = ',x2:0:2)
    end;
end.
```

19.2 Оператор выбора

Часто при программировании разветвленных алгоритмов в случае наличия множества условий удобно использовать оператор выбора:

CASE S **OF**
C1: операторы;
C2: операторы;
.....
ELSE операторы;
END;

Здесь S - величина, значение которой предварительно задается или вычисляется.

C1, C2, и т.д - константы, с которыми

```
program case_of;
 uses wincrt;
 label 1;
 var a:integer;
begin clrscr;
 1: write ('Введите оценку ');
 readln(a);
case a of
 1: begin writeln ('так не бывает'); goto 1; end;
 2: writeln ('неудовлетворительно');
 3: writeln ('удовлетворительно');
 4: writeln ('хорошо');
 5: writeln ('отлично');
 else writeln ('это фантастика');
 end;
end.
```

сравнивается значение S. В случае совпадения с одной из них выполняются соответствующие ей операторы. Если совпадения отсутствуют, то выполняются операторы, следующие за ELSE (необязательный параметр).

19.3 Логический тип. Логические операции

Иногда при создании программ для перехода на некоторую алгоритмическую ветвь необходимо проверить не одно, а несколько условий. Чтобы не писать для каждого из условий свой условный оператор, их объединяют внутри одного составного условия, все операнды (составляющие) которого связываются при помощи логических операций not, and и or.

В условном операторе IF S THEN A ELSE B условие S – это логическое выражение, и как все **величины логического типа** (***boolean***):

- может принимать всего два значения: true (истина) и false (ложь);
- в нем могут применяться **логические операции**, принцип работы которых изложен ниже:

- Операция NOT (НЕ): изменяет значение логической величины на противоположное (true на false, false на true). Например: not (9>7) = false; not (7>9) = true.
- Операция AND (И): логическое выражение истинно, если истинны все его операнды (И 1-ый, И 2-ой, И ...). Например: (2>5) and (1>7) = false; (5>2) and (1>7) = false; (5>2) and (7>1) = true.
- Операция OR (ИЛИ): логическое выражение истинно, если истинен хотя бы один его операнд (ИЛИ 1-ый, ИЛИ 2-ой, ИЛИ ...). Например: (2>5) or (1>7) = false; (5>2) or (1>7) = true (5>2) or (7>1) = true.

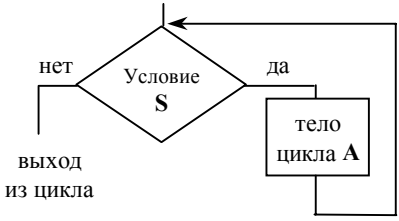
Найти максимальное из трех вводимых неравных целых чисел.

```
program if_boolean;
  uses crt;
  var a, b, c, max: integer;
begin clrscr;
  write ('1-е число = '); read (a);
  write ('2-е число = '); read (b);
  write ('3-е число = '); read (c);
  if (a>b) and (a>c) then max := a ;
  if (b>a) and (b>c) then max := b ;
  if (c>a) and (c>b) then max := c ;
  write ('максимальное = ', max);
end.
```

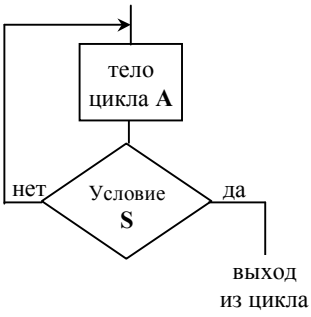
20. Операторы цикла

Цикл – это многократное повторение группы операторов (тела цикла) до выполнения некоторого условия.

1. **Цикл с предусловием**: условие выполнения цикла проверяется до тела цикла.
Реализуется оператором **WHILE** S **DO** A
A – тело цикла, которое выполняется, пока истинно логическое выражение S.



2. **Цикл с постусловием:** условие выхода из цикла проверяется после тела цикла.
Реализуется оператором **REPEAT** A **UNTIL** S
A – тело цикла, которое выполняются до тех пор, пока ложно логическое выражение S.



При решении большинства задач приемлемо использование любой из предложенных алгоритмических конструкций. При этом следует помнить, что:

- Условие выполнения тела цикла является логическим выражением, поэтому в нем можно использовать логические операции (and, or, not).

➢ При записи условия цикла особое внимание следует уделять крайним границам выполнения тела цикла: например от того, какой знак в условии ≥ или >, может зависеть правильность решения задачи.

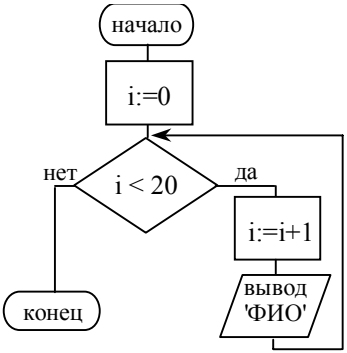
➢ Существенным отличием между циклами с пред- и постусловием является то, что последний выполняется хотя бы один раз. Кроме того, предусловие – это условие выполнения цикла, а постусловие – это условие выхода из цикла.

3. **Оператор цикла с параметром**

FOR I := X1 **TO** X2 **DO** A; **FOR** I := X1 **DOWNTO** X2 **DO** A;

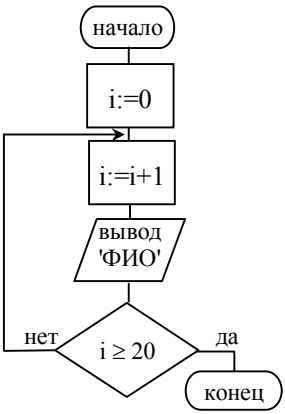
I - имя переменной (параметр цикла); X1, X2 – арифметические выражения, определяющие начальное и конечное значения параметра цикла; A – тело цикла.

Пример. Вывести на экран в столбик свою фамилию, имя и отчество 20 раз.



Цикл с предусловием

```
program cycle_while;
  uses crt;
  var i: integer;
begin clrscr;
  i := 0;
  while i < 20  do
     begin
       i := i+1;
       writeln ('Иванов Иван Иванович')
     end;
end.
```



Цикл с постусловием

```
program cycle_repeat;
  uses crt;
  var i: integer;
begin clrscr;
  i := 0;
  repeat
     i := i+1;
     writeln ('Иванов Иван Иванович')
   until  i>=20;
end.
```

Цикл с параметром

```
program cycle_for;
  uses crt;
  var i: integer;
begin clrscr;
  for  i:=1 to 20 do
     writeln ('Иванов Иван Иванович')
end.
```

Оператор цикла может быть заменен операторами условного и безусловного перехода

```
program cycle_if;
  uses crt;  label 1;
  var i: integer;
begin clrscr;
  i := 0;
  1:  i := i+1;
  writeln ('Иванов Иван Иванович');
  if  i < 20  then goto 1;
end.
```

Оператор цикла с параметром может реализовывать любую циклическую алгоритмическую конструкцию (как с пред-, так и с постусловием), при этом являясь наиболее удобным из операторов цикла. Однако использовать оператор цикла с параметром можно только при выполнении следующих условий:

- параметр цикла должен быть переменной целого типа;
- шаг изменения параметра может быть только +1 (to) или - 1 (downto).

Поэтому оператор цикла с параметром применяется в задачах, где точно определено количество **итераций** - повторений тела цикла.

21. Программирование циклических алгоритмов

При безмашинном решении задач часто возникают проблемы с проверкой правильности созданной программы. Для этого можно использовать ***трассировку*** – пошаговое выполнение программы с прослеживанием изменений значений переменных.

Пример (для приведенной правее задачи): пусть $a_1 = 1$, $d = 5$, $n = 12$.

Тогда на экране должны появиться:

числовой ряд: 1 6 11 и сумма = 18.

Проведем трассировку программы.

d	n	a	s	экран
5	12	1	0	
		6	1	1
		11	7	6
		16	18	11
				сумма = 18

Оператор цикла с предусловием

По данным a_1 и d - первому члену и разности арифметической прогрессии - вывести все ее элементы, не превышающие заданное число n и просуммировать их значения.

```
program arifm_prog;
  uses crt;
  var a, n, d, s: integer;
begin clrscr;
  write ('a1 = '); read (a);
  write ('d = '); read (d);
  write ('n = '); read (n);
  s:=0;
  while a<=n do
    begin  s:=s+a; write (a,' '); a:=a+d; end;
  writeln;
  write ('сумма = ',s);
  end.
```

Таким образом, на основании безмашинной трассировки программы можно сделать вывод о корректности ее работы.

Оператор цикла с постусловием

Последовательность Фибоначчи – это числовой ряд, первые два элемента которого = 1, а каждый следующий элемент равен сумме двух предыдущих: 1 1 2 3 5 8 13 ...

Вывести на экран все элементы, не превышающие 1000 и подсчитать их количество.

```
program fibonachi;
  uses crt;
  var a, b, c, i : integer;
begin clrscr;
  a:=1; b:=1; i:=2; write ('1  1  ');
  repeat
    c := a + b;  write (c,'  ');
    i := i + 1; a:=b; b:=c;
  until c > 1000-a;
  write ('количество  = ', i );
end.
```

Оператор цикла с параметром

Найти максимальное из n вводимых чисел, а также его порядковый номер.

```
program maximum;
  uses crt;
  var x, i, max, n, k: integer;
begin clrscr;
  write ('количество чисел = '); read (n);
  write ('число ='); read (x); max:=x; k:=1;
  for i:=2 to n do
    begin
      write ('число = '); read (x);
      if x>max then begin max:=x; k:=i end;
    end;
    write ('макс = ',max,' номер = ', k);
end.
```

22. Вложенные циклы

Вложенный цикл – это оператор цикла, размещенный внутри другого оператора цикла. Пример: **for** i: = 1 **to** 10 **do** ***for*** j:=1 ***to*** 4 ***do*** writeln (i, j)

Данный фрагмент программы будет выполняться следующим образом: для i = 1 перебираются в цикле все значения j = 1, 2, 3, 4; затем берется следующее значение i = 2, для которого снова перебираются в цикле все значения j = 1, 2, 3, 4 и т.д.

Решение большинства серьезных задач связано с использованием вложенных циклов. Для примера рассмотрим задачу о дружественных числах.

Дружественные числа – это два натуральных числа, каждое из которых равно сумме делителей другого (кроме самого другого числа). Легенда гласит, что когда Пифагора спросили, что такое дружба, он ответил – это 220 и 284. Эти числа являлись единственной известной до XX века парой дружественных чисел (в средние века считалось, что талисманы с этими числами укрепляют любовь). Действительно,

- делители 220: 1+2+4+5+10+11+20+22+44+55+110 = 284
- делители 284: 1+2+4+71+142 = 220.

Сегодня поиск дружественных чисел можно доверить компьютеру. Обратите внимание, что в предложенной программе несколько вложенных циклов.

```
program friend;
  uses crt;
  const n=1000000;
  var a, b, i, s1, s2: longint;
begin clrscr;
  for a:=2 to n do
    begin
    s1:=0;
     for i:=1 to a-1 do if a mod i = 0 then s1:=s1+i ;
     for b:=1 to a-1 do
      begin
       s2:=0;
       for i:=1 to b-1 do
         if b mod i = 0 then s2:=s2+i;
       if (s1=b) and (s2=a) then  writeln (a,' и ',b);
      end;
    end;
end.
```

Данная программа – классический пример метода перебора вариантов: в цикле перебираются натуральные числа (переменная a), для каждого из которых в цикле перебираются все возможные пары для дружбы (переменная b). Перебор вариантов прост, но требует огромных временных затрат. А для уменьшения времени работы программ используются математические методы: средствами математики разрабатываются специальные аналитические методы, упрощающие решение задач. Таким образом серьезный программист должен владеть в совершенстве не только языками программирования, но и специальным математическим аппаратом.

23. Символьные величины

- Описываются как переменные типа char, например var c: char;
- Значение – любой один символ - берется в штрих-кавычки: c:='s';
- Все используемые символы упорядочены в соответствии с величиной их числового кода (кодировка ASCII: коды от 1 до 256). Поэтому допускаются операции сравнения символьных величин (по величине их кода).

Интервалы кодов некоторых символов:

- цифры: 48 – 57 (0 .. 9);
- латинские прописные буквы: 65 – 90 (A .. Z);
- латинские строчные буквы: 97-122 (a .. z);

- русские прописные буквы: 128-159 (А – Я), 240 (Ё);
- русские строчные буквы: 160-175 (а - п), 224-239 (р – я), 241 (ё).

➢ Среди функций обработки величин символьного типа выделим:

- CHR (x) - функция преобразования числового кода символа x в сам символ, например: s := chr (68) {s = 'D'}

- ORD (s) - функция преобразования символа s в его числовой код, например: k := ord ('Б') {k = 129}

➢ *При последовательном вводе нескольких символов, их необходимо набирать непрерывной последовательностью. Ввод по одному символу через Enter ненагляден, кроме того Enter, как и любая другая управляющая клавиша, также имеет свой числовой код, вносящий погрешность в решение задачи.*

По вводимым трем символам найти символ, имеющий минимальный код.

```
program chr1;
 uses crt;
 var c: char; i, k, min: integer;
begin clrscr; min:=256;
 write ('символы: ');
 for i:=1 to 3 do
  begin
   read (c);
   k:=ord(c); if k < min  then min:=k;
  end;
write ('мин. код у символа ',chr(min));
end.
```

Вывести на экран последовательность вида a b c d . . . x y z.

```
program chr21; {1-ый способ}
 uses crt;
 var i: integer;
begin clrscr;
 for i:=97 to 122 do write (chr(i), '  ');
end.
```

```
program chr22; {2-ой способ}
 uses crt;
 var i: char;
begin clrscr;
 for i:='a' to 'z' do write (i, '  ');
end.
```

24. Строковые величины

➢ Описываются как переменные типа string , например var s: string

➢ Значение – совокупность до 126 символов, которая берется в штрих-кавычки, например s := ' Петров Петр Петрович '

В заданном тексте вычислить количество слов, начинающихся с русской 'а'

```
program string1;
 uses crt;
 var txt, bukvi : string;  i, k : integer;
begin clrscr; k:=0;
 write (Введите текст '); read (txt);
 if copy (txt, 1, 1) = 'а' then k := 1;
 for i:=2 to length(txt)  do
  begin
   bukvi := copy (txt, i, 2);
   if bukvi = '  а' then k := k + 1
  end;
 write ('Слов, начинающихся на а = ' ,k);
end.
```

Вывести на экран заданное слово так, чтобы каждая буква слова, начиная с первой, перемещалась с правого края экрана на левый край

```
program string2;
  uses crt;
  var txt, b: string;  i, j: integer;
begin clrscr;
  write ('Введите текст '); read (txt);
  for i:=1 to length(txt) do
    begin
      b:=copy(s,i,1);
      for j:=80 downto i do
        begin
          gotoxy (j,2); write (b);
          gotoxy (j+1,2); write (' '); delay (10);
        end;
    end;
end.
```

Среди функций обработки величин строкового типа (строк) выделим:
- Length (txt) - определение длины строки txt (при этом считаются все символы, включая пробелы). Пример: a:= 'информатика' ; x:= length (a) {x=11}
- Copy (txt, x, y) - выделение из строки txt подстроки: y символов, начиная с символа с номером x. Пример: a:= ' железо ' ; b:= copy (a, 1, 4) { b = ' желе ' }
- Str (x, txt) - преобразование числа x в строку txt. Пример: x:=53; str(x, s) {s='53'}
- Val (txt, x, c) - преобразование строки txt в число x (c: integer).
Пример: s:= ' 345.32 ' ; val (s, x, c) {x = 345.32}

Кодирование информации

Кодирование (шифрование) информации - одна из важнейших задач, которые решаются при помощи компьютера. Существует множество алгоритмов шифровки данных, которые изучаются специальной наукой – криптографией.

Здесь предложена несложная программа шифровки информации, в основе которой заложен следующий принцип: каждый символ шифруемого текста кодируется числом – порядковым номером его расположения в тексте-ключе (предполагается что этот текст содержит все символы алфавита и известен ограниченному кругу лиц).

```
program shifrovka;
  uses crt;
  const key = '<текст-ключ придумайте сами>';
  var  txt, s: string; i, j: integer;
begin clrscr;
  write (' Введите текст ');  read (txt);
  for i := 1 to length (txt) do
    begin
      s := copy (txt, i, 1);
      for j := 1 to length (key) do
        if s = copy (key, j, 1) then break;
      write (j, ' ');
    end;
end.
```

25. Массивы

Массив – это совокупность однотипных данных, каждый элемент которых характеризуется своим набором индексов, однозначно определяющих положение элемента в массиве. Массивы - наиболее удобный способ хранения и обработки информации в программах, поэтому они являются неизменной составляющей всех серьезных программ.

Основные характеристики массива: ***размерность*** (количество индексов, характеризующих местоположение элемента данных) и ***тип*** находящихся в нем данных. Мы ограничимся рассмотрением одно- и двумерных массивов, хотя Pascal позволяет работать с массивами и большей размерности.

Одномерный массив (вектор): данные записываются в строчку, каждый элемент характеризуется одним индексом: A [1] = 9; A [3] = 7.

индексы	1	2	3	4	5
	9	**0**	**7**	**1**	**12**

Двумерный массив (матрица): данные записываются в таблицу, каждый элемент характеризуется двумя индексами (номерами строки и столбца, в которых он находится): A [1,1] = 3.2; A [2,3] = 5.8; A [3,4] = 9.9

индексы	1	2	3	4
1	3.2	14.5	-8.65	7
2	2.45	5.23	5.8	12.5
3	-34	-1.6	6.2	9.9

Массив, являясь нестандартным типом, должен быть описан в разделе описаний после зарезервированного слова type: **TYPE** <имя типа> = **ARRAY** [<имена индексов>] **OF** <стандартный тип элементов массива>.

Для приведенных выше примеров описание массивов будет выглядеть:
- type t = array [1..5] of integer; var a: t; - описан тип с именем t, который является одномерным массивом, состоящим из 5 целых чисел; переменная а имеет тип t (массив);
- type t = array [1..3,1..4] of real; var a: t; - описан тип с именем t, который является двумерным массивом, состоящим из 12 вещественных чисел; переменная а имеет тип t (массив);
- тип-массив можно вводить и непосредственно при описании переменных:
var a: array [1..4, 1..3] of real.

26. Работа с одномерными массивами

Ввод, обработка и вывод массивов любых размерностей связаны с использованием оператора цикла с параметром. При этом параметр цикла, как правило, соотносится с индексами элементов массива.

Способы задания массива:

1. С клавиатуры во время выполнения программы.
2. Случайными числами при помощи функции **RANDOM**:

- random – случайное вещественное число в интервале [0; 1];
- random (x) – случайное целое число в интервале [0; x-1].

Данная функция позволяет задать любой интервал случайных чисел:

- a + (b-a)*random – вещественные числа в интервале [a; b];
- a + random(b-a+1) – целые числа в интервале [a; b];

Например, -5 + 12*random - случайные вещественные числа в интервале [-5;7];
3 + random (45) – случайные целые числа в интервале [3;47].

При работе со случайными числами (а они, кроме всего прочего, лежат в основе всех игр) в начале программы необходимо использовать процедуру **RANDOMIZE**, которая инициализирует генератор случайных числе - привязывает выработку случайного числа к системному времени (время в каждый момент разное, соответственно получаются различные последовательности случайных чисел).

Задать с клавиатуры массив B[15] и найти сумму элементов массива.

```
program mas_sum;
  uses crt;
  type t=array [1..15] of integer;
  var b: t; i, sum: integer;
begin clrscr; sum:=0;
  for i:=1 to 15 do
    begin  write ('b[',i,'] = '); read (b[i]) end;
  for i:=1 to 15 do sum:=sum + b[i];
  write('сумма элементов  = ', sum);
end.
```

Задать массив A[9], элементы которого случайные целые числа в интервале от 8 до 29. Вывести на экран все элементы массива, кратные 3.

```
program mas_rnd;
  uses crt;
  type t = array [1..9] of integer;
  var a: t;  i: integer;
begin clrscr; randomize;
  for i:=1 to 9 do
    begin a[i]:=8+random(22); write (c[i],' '); end;
  writeln;
  for i:=1 to 9 do if a[i] mod 3 = 0 then write(c[i],' ');
end.
```

Задача упорядочивания одномерного массива по возрастанию

Задача об упорядочивании элементов массива по некоторому признаку является одной из частовстречающихся в прикладном программировании.

Логика программы:

1. Ввод массива.
2. Обработка массива, которая происходит по следующему алгоритму: находится мини-

мальный элемент массива, который записывается под индексом 1, затем ищется следующий минимальный элемент (первый уже не считается), который записывается под индексом 2 и т.д.

Проблема здесь заключается в том, что когда минимальный элемент (min) записывается под индексом j , не должен пропасть элемент массива, который находился под этим индексом. Для этого его записывают под бывшим индексом минимального элемента (i_min). Таким образом за одну итерацию два элемента массива меняются местами.

3. Вывод упорядоченного по возрастанию массива.

Переставить элементы массива A[10] (случайные числа в интервале от 1 до 99) так, чтобы они образовали возрастающую последовательность.

```
program mas_upor;
  uses crt;
  type t = array [1..10] of real;
  var a: t;  i, j, min, i_min, k: integer;
begin clrscr; randomize;
  for i:=1 to 10 do                                          1
   begin
     a[i]:=1+98*random; write (a[i]:0:2,' ');
   end;
  writeln;
  for j:=1 to 9 do                                           2
   begin  min := a[j];
     for i := j to 10 do
       if a[i] <= min then begin min:=a[i]; i_min:=i; end;
     a[i_min] := a[j];  a[j] := min;
   end;
  writeln;
  for i:=1 to 10 do  write(a[i]:0:2,' ');                    3
end.
```

27. Работа с двумерными массивами

Работа с двумерными массивами связана с использованием вложенных циклов. Действительно, например для задания массива A[6, 7] необходимо записать в программе: for i:=1 to 6 do for j:=1 to 7 do read (a[i, j]).

Особый интерес среди двумерных массивов представляют квадратные матрицы – двумерные массивы, в которых количество строк и столбцов совпадает.

В квадратной матрице выделяют:

- главную диагональ – последовательность элементов от левого верхнего до правого нижнего элемента;
- побочную диагональ – последовательность элементов от правого верхнего до левого нижнего элемента.

Соответственно эти диагонали делят квадратную матрицу на 4 четверти: верхнюю, нижнюю, правую и левую.

Задать массив A[10,10], элементы которого: на главной диагонали = 1; выше нее – случайные целые числа [0; 8]; ниже равны разности своих индексов.

```
program mas2_kw;
  uses crt;
  type t = array [1..10,1..10] of integer;
  var a: t; i, j: integer;
begin clrscr; randomize;
  for i:=1 to 10 do
   begin
    for j:=1 to 10 do
     begin
      if i = j then a[i, j] := 1;
      if i < j then a[i, j] := random(9);
      if i > j then a[i, j] := i - j;
      write (a[i, j],' ');
     end;
    writeln;
   end;
end.
```

28. Структурное программирование

Профессиональный программист вынужден писать программы, сложные по сути и большие по объему, что может привести его к серьезной путанице в логике программы, а следовательно к ошибкам в ней. Здесь ему на помощь и приходят методы структурного программирования, основная идея которого: если целое является очень сложным для восприятия, то его необходимо структурировать – разделить на более простые части, с которыми уже и работать. Поэтому в реальном (а не учебном) программировании первым шагом программиста при решении задачи является ***составление плана решения (его также можно назвать алгоритмом), в котором надо правильно разбить единую программу на составляющие ее части***.

Структурное программирование - это способ организации программ, основными отличительными особенностями которого являются:

1. ***Программа должна состоять из блок-программы (основной программы) и блоков (подпрограмм), к которым блок-программа обращается.***
2. ***Основная программа*** должна быть короткой и содержать только логику решения задачи. Она связывает между собой части (подпрограммы) в целое и состоит, в основном, из операторов обращения к подпрограммам.
3. Каждый блок оформляется в виде подпрограммы. ***Подпрограмма*** – это часть программы, реализующая некоторую законченную операцию и допускающая многократное обращение к ней из любого места основной программы. Основная программа обращается к подпрограмме по ее имени; после отработки подпрограммы осуществляется переход на оператор основной программы, следующий за тем, который обращался к подпрограмме.
4. Программа пишется сверху вниз, то есть начиная с основной программы. Т.о. сначала определяется логика решения, оформляемая в основной программе в виде некоторой последовательности блоков, каждый из которых затем записывается в виде подпрограммы.

На сегодня структурное программирование является общепризнанным стандартом: большая неструктурированная программа рассматривается как неграмотно написанная. Данный метод, кроме облегчения чтения логики программы и возможности исправления ошибок, позволяет разделить работу между исполнителями: каждый программист может работать над своим блоком.

29. Подпрограммы: процедуры и функции

В Pascal существуют две разновидности подпрограмм: процедуры и функции, структура которых в целом аналогична и подобна структуре основной программы:

1. ***Заголовок*** (в отличие от основной программы в подпрограмме он обязателен, так как именно по нему основная программа обращается к подпрограмме):

- **PROCEDURE** < имя процедуры > (< список формальных параметров >).
- **FUNCTION** <имя функции> (<список формальных параметров>): <тип результата функции>.

Формальные параметры - параметры, которыми оперирует подпрограмма. (*Параметры – это метки, константы, типы, переменные*). Каждый формальный параметр соотносится с **фактическим параметром** – параметром, используемым в основной программе. Среди типов формальных параметров выделим:

- Параметр-значение: при его изменении соответствующий ему фактический параметр не меняется. Пример: function max (**mas: t**): integer
- Параметр-переменная: при его изменении соответствующий ему фактический параметр также меняется. Описание параметра переменной в отличие от параметра-значения начинается с зарезервированного слова var.

Пример: procedure vvod_mas (**var mas: t**);

Обращение к подпрограмме в основной программе осуществляется по имени подпрограммы с указанием списка фактических параметров, которые должны совпадать по количеству и типу с формальными параметрами подпрограммы.

2. ***Раздел описаний* локальных параметров** – параметров, доступных только данной подпрограмме. Параметры, описываемые в разделе описаний основной программы и доступные по всей программе, называются **глобальными параметрами**. Очевидно, что обмен информацией между основной программой и ее подпрограммами может осуществляться только при помощи глобальных параметров.

3. ***Раздел операторов*** - тело подпрограммы.

- ***Тело подпрограммы-процедуры – это последовательность каких-либо действий.***
- ***Тело подпрограммы-функции – это вычисление какого-либо параметра.*** (поэтому в теле функции ее имени обязательно должно быть присвоено значение).

Пример использования метода структурного программирования

Задача. Ввести с клавиатуры два массива, каждый из которых состоит из 10 целых чисел. Найти максимальные элементы каждого массива.

План действий по решению задачи:

1. Создать основную программу, в которой отражена логика решения поставленной задачи:
   1. Ввод массива A.
   2. Нахождение максимального элемента массива A.
   3. Ввод массива B.
   4. Нахождение максимального элемента массива B.
2. В соответствие с логикой решения задачи очевидна необходимость создания двух блоков: ввод массива (процедура vvod_mas) и нахождение максимального элемента массива (функция max).

```
program str_prog;
  uses crt;
  type t = array [1..10] of integer;
  var a, b: t;
  procedure vvod_mas (var mas: t);
    var i: integer;
     begin
      for i:=1 to 10 do
        begin write (i,' элемент = '); read (mas[i]) end;
      writeln;
     end;
  function max (mas: t): integer;
    var i, m: integer;
      begin
        m := mas[1];
        for i:=2 to 10 do if mas [i] > m then m:=mas [i];
        max := m;
      end;
begin clrscr;
  vvod_mas (a);
  writeln ('максимум массива A = ',max(a));
  vvod_mas (b);
  writeln ('максимум массива B = ',max(b));
end.
```

Обратите внимание, что по способу применения задаваемые программистом процедуры и функции полностью идентичны стандартным процедурам и функциям ЯП. Действительно: есть стандартная процедура read для ввода значения переменной, а мы создали процедуру vvod_mas для ввода значений массива; есть стандартная функция sqr , вычисляющая квадрат переменной, а мы создали функцию max, вычисляющую максимальный элемент массива. Только одни процедуры и функции являются стандартными (их можно использовать "по умолчанию", например процедуры read, write и т.п., функции sin, abs и т.п.), а другие - определяются пользователями (в нашем примере процедура vvod_mas и функция max). Более того, определяемые пользователем подпрограммы могут быть сохранены. Очевидно, что за длительное время работы с тем или иным ЯП у каждого программиста накапливается солидная библиотека подпрограмм (процедур и функций), которые он может использовать в своей дальнейшей работе. Это также является одним из важнейших достоинств структурного программирования.

ЗАДАЧИ

Процедура write

5. Вычислить значения следующих арифметических выражений:

A. $581 \cdot 1241$

B. $415 \cdot (-311) + 123$

C. $\frac{1848}{31} + 35$

D. $\sin 25^0$

E. $\cos 16^0$

F. $\frac{15 + 41{,}1^2}{35 \cdot 3{,}5^2}$

G. $\frac{3{,}2 \cdot 7{,}1^2}{1{,}3 \cdot 4{,}5} + 2{,}23$

H. $71 - \frac{\sqrt{23} - 3}{3{,}13^2} \cdot 9{,}56$

I. $\left(\frac{24\sqrt{8} + 31}{12 + 1{,}8^2} \right)^2 + 18{,}5$

J. $\left(\frac{4\sqrt{18 - 3{,}1^2} + 3{,}1}{2{,}18^2} \right)^2 - 3{,}5$

K. $\sqrt{\frac{9{,}1^2 + 11}{14 \cos 64^0} + 2{,}1}$

L. $3{,}14^4 - 1{,}48^8$

M. $\frac{7{,}33^4 + \sqrt{2956 \cdot \sin 11^0}}{987{,}3 - 15{,}2^2}$

Процедура read

6. Вывести на экран фразу "Сидоров получил 5 по информатике", если перед этим вводится число - отметка Сидорова (5):

A. ввод осуществляется без подсказывающего текста;

B. ввод с подсказывающим текстом;

C. выводимый текст размещается в центре экрана зеленым цветом.

7. Вывести в центре экрана фразу "Сидоров получил 5, а Иванов - 4", если перед этим последовательно вводятся числа - отметки Сидорова (5) и Иванова (4):

A. ввод без подсказывающего текста, вывод - по центру экрана;

B. ввод с подсказывающим текстом, вывод - по центру экрана;

C. без центрирования, сообщения об отметках Сидорова и Иванова - на разных строках разным цветом (синим и красным).

Оператор присваивания

8. Для двух целых чисел найти:

A. сумму, без использования оператора присваивания;

B. сумму, с использование оператора присваивания;

C. частное с точностью до четырех знаков после запятой;

D. целую часть и остаток при делении чисел;

E. среднее арифметическое с точностью до трех знаков после запятой;

F. среднее геометрическое с точностью до двух знаков после запятой.

9. Для двух вещественных чисел найти:

A. частное;

B. целую и дробную часть частного;

C. первое число умножается на 3, второе – на 4; в центре экрана вывести сумму полученных элементов.

10. По вводимым натуральным a и b найти $y = 4a^2 + \frac{\sqrt{b^3 + 1}}{a - b}$

11. Найти сумму целой и умноженной на 10 дробной части заданного числа.

12. Найти периметр и площадь прямоугольника по двум заданным сторонам.

13. А. Вычислить по заданному радиусу длину окружности и площадь круга;
В. * использовать одну переменную для длины окружности и площади круга.
14. * Найти сумму цифр заданного натурального трехзначного числа.
15. * Определить число, полученное выписыванием в обратном порядке цифр заданного натурального трехзначного числа.
16. * Поменять местами значения вводимых переменных x и y:
А. с использованием дополнительной переменной;
В. ** без использования дополнительной переменной.

Операторы безусловного и условного перехода

17. Вывести на экран, используя временную задержку в 1 секунду бесконечный ряд, состоящий из своих фамилии, имени и отчества.
18. Составить программу многократного нахождения суммы двух целых чисел:
ввод чисел – их сумма; ввод новых чисел – их сумма и т.д.
19. По вводимому значению x найти $y = \begin{cases} 1, \text{если } x \geq 0 \\ 0, \text{если } x < 0 \end{cases}$
А. использовать краткий формат оператора условия;
В. использовать полный формат оператора условия.
20. Определить, является ли заданное натуральное число n:
А. четным или нечетным числом;
В. кратным данному числу k.
21. По вводимым a и b найти $c = \begin{cases} a + b, \text{если } a - \text{целое} \\ a - b, \text{если } a - \text{нецелое} \end{cases}$
22. Найти частное двух целых чисел, предусмотрев защиту от деления на 0.
23. Дано однозначное натуральное число. Найти:
А. является ли оно делителем заданного числа m;
В. предусмотреть защиту от ввода числа с другим количеством цифр.
24. Определить, равна ли сумма двух первых цифр заданного натурального четырехзначного числа сумме двух его последних цифр. Предусмотреть защиту от ввода числа с другим количеством цифр.
25. Найти номер той четверти координатной плоскости, в которой находится точка с координатами x и y ($x \neq 0, y \neq 0$).
26. По вводимым значениям целых чисел x и y найти $z = \begin{cases} \max(x, y), \text{если } x < 0 \\ \min(x, y), \text{если } x \geq 0 \end{cases}$
27. Определить, одинаковы ли цифры заданного двузначного числа.
Предусмотреть защиту от ввода числа с другим количеством цифр.
28. А. Вводятся последовательно 3 целых числа, два из которых равны между собой. Найти отличное от других число;
В. предусмотреть защиту от неправильного ввода чисел;
С. найти порядковый номер отличного числа.
29. Найти среднее по величине из трех вводимых целых чисел.
30. * Даны целые числа a, b и c. Определить, возможно ли построение треугольника с заданными сторонами. Если возможно, то указать тип треугольника: равносторонний, равнобедренный или прямоугольный, если он таковым является.
31. * Перераспределить значения переменных x и y так, чтобы в x оказалось большее из этих значений, а в y - меньшее.

32. * Вывести на экран первые три цифры из дробной части вещественного числа. Определить, есть ли среди них цифра 0.
33. * Значения переменных a, b и c поменять местами так, чтобы было a > b > c.
34. * Составить программу для угадывания натурального числа, не превосходящего 100, задуманного компьютером (задаете сами). По ходу угадывания компьютер должен давать подсказки: "Больше" или "Меньше", а в конце выдать количество сделанных попыток.

Операторы цикла

35. Вывести на экран свое имя в столбик:
A. 20 раз;
B. n раз.
36. A. Вывести на экран ряд натуральных чисел, не превышающих заданного n;
B. вычислить сумму полученных элементов.
37. A. Вывести на экран элементы арифметической прогрессии (a_1=3, d=2): 3 5 7 9... не превышающие заданного m;
B. найти количество выведенных элементов;
C. найти сумму выведенных элементов.
38. Распечатать с шагом $\pi/4$ значения функции sin (x) на отрезке [0, 2π].
39. Вывести элементы последовательности Фибоначчи, не превышающие:
A. 1000;
B. заданного числа n;
C. вычислить сумму выведенных элементов.
40. Подсчитать количество цифр в заданном натуральном числе.
41. Определить, является ли заданное натуральное число степенью числа 3.
Если является степенью, то определить какой.
42. Для пяти вводимых целых чисел найти:
A. максимальное;
B. минимальное;
C. разность между максимальным и минимальным;
D. номера максимального и минимального элементов по порядку ввода.
43. Для натуральных a и b вывести на экран из интервала [a , b]:
A. четные числа;
B. нечетные числа, выводить в порядке убывания;
C. числа, кратные заданному числу k.
44. Для заданного натурального n найти:
A. все делители;
B. количество делителей;
C. определить, является ли оно простым (простое число - это натуральное число, которое не имеет делителей, кроме 1 и самого себя);
D. определить, является ли оно совершенным (совершенное число – это натуральное число, которое равно сумме всех своих делителей за исключением самого себя).
45. Проверить, являются ли два данных числа дружественными.
46. По данным b_1 и q - первому члену и знаменателю геометрической прогрессии вывести на экран:
A. первые 8 элементов, просуммировав их значения;

B. элементы, не превышающие заданного n;
C. первый элемент, превышающий заданное n.

47. Для заданного натурального числа n вычислить его факториал.
Факториал числа вычисляется по формуле $n! = 1 \cdot 2 \cdot 3 \cdot 4 \cdot \ldots \cdot n$.

48. Для последовательности Фибоначчи найти:
A. первые n элементов и их сумму;
B. сумму всех элементов с номера n по номер m.

49. Вывести все натуральные числа, квадраты которых находятся в интервале [a, b].

50. Выполнить умножение целых положительных чисел a и b, заменив действие умножения на действие сложения.

51. * Составить программу деления двух натуральных чисел с выделением целой части и остатка от деления (без использования операций деления).

52. * Построить из звездочек в центре экрана прямоугольник размером $n \times m$.

53. * По заданному n вывести на экран таблицу Пифагора, имеющую n строк и n столбцов. Каждое число, входящее в эту таблицу, равно произведению номеров строки и столбца, определяющих положение данного числа.

54. * Найти все простые числа, не превосходящие заданного числа n.

55. * Найти все совершенные числа, не превосходящие заданного числа n.

56. * A. Вывести возрастающую последовательность, составленную из трехзначных чисел, сумма цифр которых равна заданному числу n, и определить их количество;
B. определить при каком значении n получается самая длинная последовательность таких чисел.

57. * Найти сумму цифр заданного натурального числа.

Символьные величины

58. По вводимым трем символам вывести:
A. их числовые коды;
B. просуммировать эти коды.

59. По трем вводимым числовым кодам вывести последовательность соответствующих им символов (ввод кодов через пробел).

60. Организовать ввод символов до появления символа '.'
Найти количество вводимых символов.

61. Вывести на экран постранично (с использованием временной задержки) весь набор символов вместе с их кодами.

62. Вывести на экран последовательности символов вида:
A. z y x . . . c b a
B. а б в . . . э ю я

63. По вводимому символу получить:
A. его числовой код;
B. предшествующий ему и следующий за ним символы.

64. По вводимым пяти прописным латинским буквам найти их порядковый номер в алфавите.

65. * Вывести на экран последовательности символов вида:
A. a b b c c c . . . z z . . z z
B. z y y x x x . . . a a . . a a
C. a a b a b c . . . . . a b c . . . x y z

Строковые величины

66. В заданном тексте подсчитать:
A. количество символов 'd' ;
B. количество слов;
C. количество слов, оканчивающихся на букву 'q';
D. количество слов 'dog'.
67. В заданном тексте определить каких букв больше: русских 'а' или 'о'.
68. Вывести на экран заданное слово (до 10 символов) так, чтобы каждая буква слова, начиная с первой, перемещалась:
A. с правого края экрана на левый край;
B. с верхней строки экрана на нижнюю строку;
C. с нижней строки экрана на верхнюю строку;
D. начиная с последней буквы, с левого края экрана на правый край.
69. Организовать "бегущую строку": вводимый текст должен перемещаться по горизонтали экрана справа налево.
70. * Проверить, является ли вводимое слово палиндромом ?
(палиндром – слово, одинаково читающееся в обоих направлениях).
71. * Найти сумму цифр заданного натурального числа.

Массивы

72. Задать с клавиатуры массив A[4] и найти:
A. сумму элементов массива;
B. сумму четных элементов;
C. сумму элементов с четными номерами;
D. среднее арифметическое элементов;
E. среднее арифметическое положительных элементов.
73. Задать с клавиатуры массив B[3] и проверить:
A. сумма элементов – четное число?
B. сумма квадратов элементов > 100 ?
C. среднее арифметическое из квадратных корней элементов > 5 ?
74. Для массива C[8], элементы которого случайные целые числа [5; 53] вывести на экран:
A. все элементы;
B. все элементы в обратном порядке;
C. все элементы, оканчивающиеся на 0.
75. Преобразовать массив A[6], элементы которого – случайные вещественные числа [2; 20] (точность – один знак после запятой):
A. уменьшить все элементы в 2 раза;
B. увеличить все элементы на заданное число n;
C. умножить все элементы на последний элемент массива;
D. заменить все нечетные элементы на их квадратные корни.
76. Задать массив A[9], элементы которого случайные целые числа [0; 10].
Заполнить массив B элементами массива A:
A. увеличенными в 3 раза;
B. уменьшенными на величину первого элемента массива A;
C. внести только элементы с четными индексами;

D. внести только элементы с нечетными индексами.

77. Задать массив A[10], элементы которого случайные целые числа [–15; 15], и найти:

A. максимальный элемент массива;

B. минимальный элемент и его индекс;

C. максимальным и минимальный элементы, а также их разность;

D. ко всем элементам прибавить максимальный элемент.

78. Задать массив A[6], элементы которого случайных целые числа [1; 99], причем каждый следующий элемент должен быть не меньше предыдущего.

79. * Задать массив A[8], элементы которого случайные целые числа [0; 50].
Заполнить массив B элементами массива A, упорядоченными по возрастанию.

80. Массив A[5,9] вывести в виде матрицы. Элементы массива:

A. случайные целые числа [1; 9];

B. прописные латинские буквы;

C. строчные латинские буквы.

81. Для массива A[4,7], элементы которого случайные целые числа [1; 9], найти:

A. индексы всех элементов равных 4;

B. номера строк, в которых присутствует число 5;

C. номера столбцов, в которых присутствует число 2;

D. максимальный элемент и его индексы.

82. A. Задать массив A[10,10], элементы которого: на главной диагонали = 1; выше нее – случайные целые числа [0; 8]; ниже - равны разности своих индексов;

B. выделить эти три группы элементов массива разным цветом.

83. Задать массив A[23,23], элементы которого: на побочной диагонали = 1 (выделить желтым цветом); выше нее = 2; ниже = 0.

84. Задать массив A[5,5], элементы которого случайные целые числа [0; 9].
Выделить цветом:

A. элементы верхней четверти;

B. элементы правой четверти и найти их сумму.

85. Задать массив A[7,7], элементы которого случайные целые числа [0; 9].
Проверить равенство сумм элементов на главной и побочной диагоналях.

Подпрограммы: процедуры и функции

96. Найти сумму четырех чисел (процедура - ввод значения переменной).

97. Найти среднее арифметическое элементов в каждом из трех массивов A[5], B[5] и C[5] (процедура - ввод массива, функция - нахождение среднего арифметического).

98.Найти количество делителей четырех натуральных чисел (процедура - ввод значения переменной, функция - нахождение количества делителей).

99. Найти все совершенные числа, не превосходящие заданного n (функция – нахождение суммы делителей числа).

100. Числа-близнецы - это простые числа, отличающиеся друг от друга на 2.
Найти все пары чисел - близнецов, не превышающих n.
(функция – определение, простое ли число? /логический тип/).