1. **Алгоритм. Способы записи алгоритмов. Схема алгоритма.**

Алгоритм - это последовательность шагов, представляющая собой инструкцию или описание процесса выполнения определённой задачи.  
Способы записи алгоритмов могут быть различными, включая текстовое описание, псевдокод, языки программирования и схемы алгоритмов.  
Схема алгоритма - это графическое представление последовательности шагов, необходимых для выполнения задачи. Схемы алгоритмов состоят из блоков, представляющих отдельные шаги, соединенных стрелками, обозначающими направление выполнения шагов.

1. **Характеристики языка Pascal. Этапы разработки программы**.

Язык программирования Pascal был разработан Никлаусом Виртом в 1970-х годах и стал одним из первых структурированных языков программирования. Вот несколько основных характеристик языка Pascal:

1.Структурная языковая конструкция:  
2. Сильная типизация  
3. Поддержка процедур и функций  
4. Удобство в написании программ

Этапы разработки программы:

* 1. Понимание проблемы или постановки
  2. Планирование
  3. Написание кода
  4. Тестирование
  5. Отладка

1. **Структура программы на языке Pascal. Организация пользовательского интерфейса.**

Структура программы на языке Pascal обычно состоит из нескольких основных частей:

0. Заголовок программы (program z1;)

1.Объявление переменных ( var i: integer; )

2.Описание пользовательских процедур и функций( procedure z1; ) or ( function z(i: integer); )

3.Тело программы ( begin … end. )

Организация пользовательского интерфейса в программе на Pascal может быть реализована с помощью различных процедур вывода информации на экран (например, writeln) и чтения данных с клавиатуры (например, readln). Или например с помощью case-меню.

1. **Типы данных. Понятие переменная и идентификатор.**

Типы данных в языке программирования Pascal представляют различные категории значений, которые могут быть использованы в программе. Несколько основных типов данных в Pascal включают в себя:

1.Целочисленные типы данных(integer)

2.Вещественные типы данных(real)

3.Символьные типы данных (char, string)

4.Логические типы данных(Boolean)

Переменная в Pascal - это именованное хранилище для данных определенного типа. Переменная может быть использована для хранения различных значений и изменения их в процессе выполнения программы.

Идентификатор в Pascal - это имя, используемое для идентификации переменной, процедуры, функции, модуля, типа данных и других конструкций в программе. Идентификаторы могут быть заданы программистом в соответствии с правилами именования, предусмотренными языком программирования.

1. **Явное и неявное приведение типов. Правила совместимости**

Явное и неявное приведение типов относится к процессу изменения типа данных переменной в программировании.  
1. Неявное приведение типов:  
Неявное приведение типов происходит автоматически компилятором при выполнении операций, когда тип данных переменной автоматически преобразуется в другой тип без явного указания программистом.

2. Явное приведение типов:  
Явное приведение типов происходит, когда программист явным образом указывает компилятору преобразовать тип данных переменной в другой тип. Это может быть выполнено с помощью операторов явного приведения типов.

( b := byte(i);)

Правила совместимости типов:  
В Pascal нельзя смешивать различные типы данных в операциях, например, складывать вещественное число с символьным, или сравнивать целое число с логическим. Однако, существуют правила неявного приведения типов, которые позволяют, например, использовать вещественные числа в арифметических операциях с целыми числами.

1. **Основные операторы языка Pascal. Линейные алгоритмы. Математические функции.**

Основные операторы в языке Pascal

1.Оператор присвоения ( := )

2.Арифмеический оператор (div, mod, +, -)

3.Логический оператор (not, and, or)

4.Оператор ветвления (if … then … else)

Линейные алгоритмы в программировании представляют собой последовательность инструкций, выполняемых последовательно от начала до конца. Это может включать в себя выполнение математических операций, присваиваний, ввода-вывода данных и других операций.

Математические функции в Pascal могут включать в себя стандартные математические операции, такие как:

1.Возведение в степень ( sqt(), power(x,2), exp(2\*ln(x)) )

2.Вычисление кв.корня ( sqrt() )

3.Вычисление модуля числа ( abs() )

4.Тригонометрическая функция ( sin(), cos() )

1. **Основные операторы языка Pascal. Условные операторы и оператор выбора.**

Основные операторы в языке Pascal

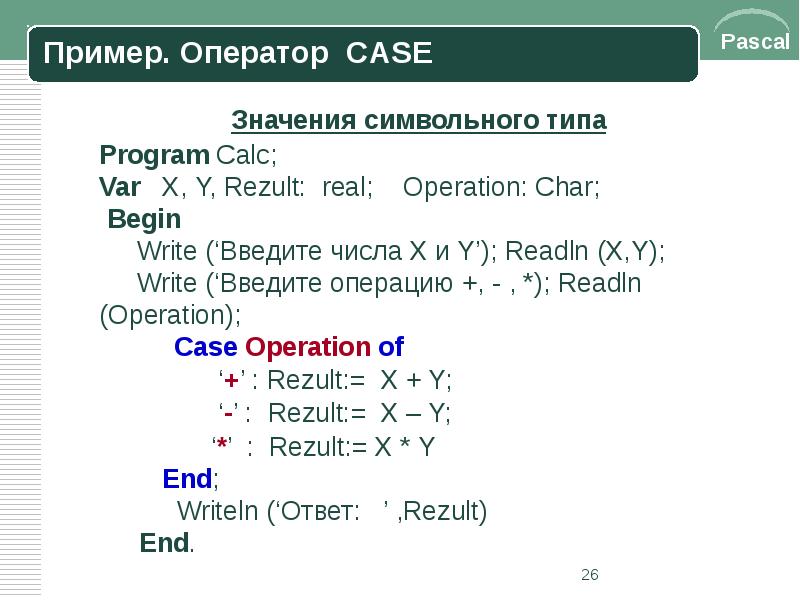
1.Оператор присвоения ( := )

2.Арифмеический оператор (div, mod, +, -)

3.Логический оператор (not, and, or)

4.Оператор ветвления (if … then … else)

Условные операторы (if … then … else) позволяют выполнять определенные блоки кода в зависимости от условия, в то время как оператор выбора(case) позволяет выполнять различные блоки кода в зависимости от значения выражения.



1. **Основные операторы языка Pascal. Организация циклических процессов.**

Основные операторы в языке Pascal

1.Оператор присвоения ( := )

2.Арифмеический оператор (div, mod, +, -)

3.Логический оператор (not, and, or)

4.Оператор ветвления (if … then … else)

Циклические процессы в программах позволяют выполнять одни и те же действия многократно до тех пор, пока не будет выполнено определенное условие. Примеры:

1.while <условие> do …;

2.for i := n to/downto k do …;

3.repeat …; until <условие>;

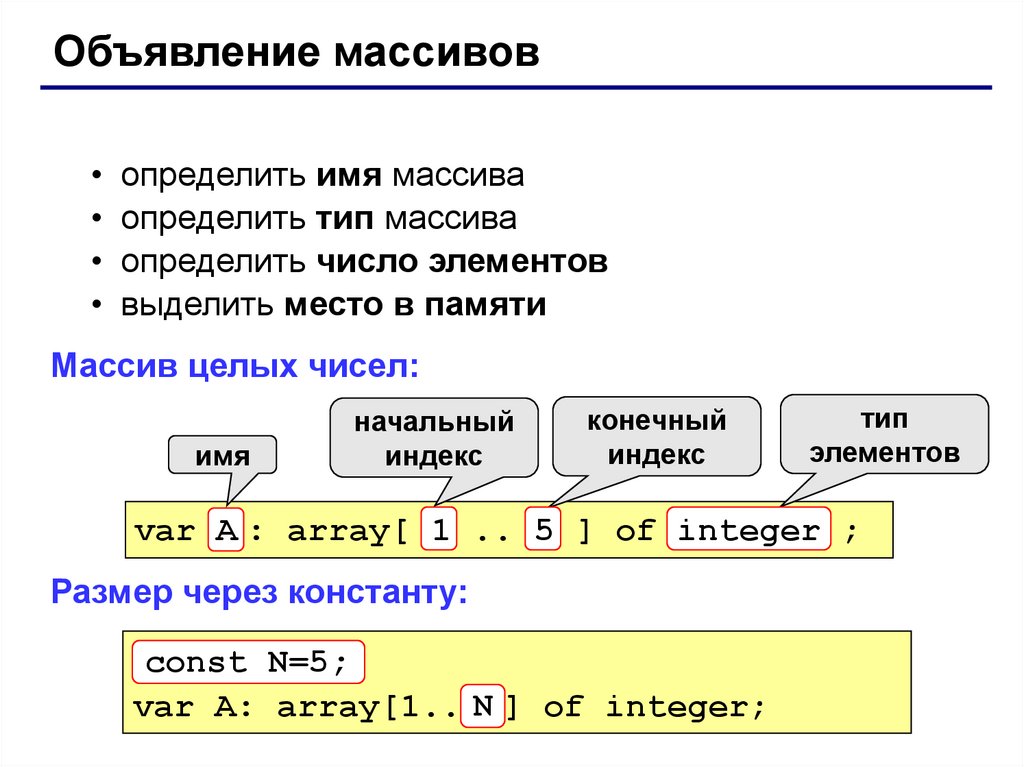
1. **Работа со структурированными типами данных. Понятие массив, размерность массива, индекс элемента массива.**

Работа со структурированными типами данных в языке программирования Pascal включает в себя работу с массивами, которые позволяют хранить однотипные элементы данных.

Массив представляет собой структурированный тип данных, содержащий фиксированное количество элементов одного и того же типа.

Размер массива - это количество элементов, которые он может содержать. Размерность массива определяется при его объявлении и не может быть изменена во время выполнения программы.

Индекс элемента массива - это порядковый номер элемента в массиве. В Pascal индексация элементов массива начинается с 1.



1. **Работа со структурированными типами данных. Размещение массива в памяти. Способы инициализации массивов. Открытый массив.**

Работа со структурированными типами данных в языке программирования Pascal включает в себя работу с массивами, которые позволяют хранить однотипные элементы данных.

Размещение массива в памяти:  
В Pascal массивы размещаются в памяти последовательно, начиная с адреса, соответствующего первому элементу массива. Элементы массива располагаются друг за другом в соответствии с их порядком. Это означает, что для доступа к элементу массива компьютеру известно, как вычислить адрес этого элемента с учетом его индекса и адреса начала массива.

Способы инициализации массива:

1. Присвоение значений элементам массива ( A [ 1 ] := 1; )

2. Цикл инициализации

for i := 1 to 5 do

A [ i ] := 2 \* i;

3. Инициализирование массива случайными значениями

randomize;

for i := 1 to 10 do

A [ i ] := random(100);

4. Константой

const

A: array[1…5] of real = (0, 0.5, 0.4, 0.6, 4.5);

5. FillChar( var V; Count: Word; B: Byte );

Открытый массив в Pascal – это массив переменной длины. Он не имеет фиксированного размера, и его размер может быть изменен в процессе выполнения программы. ( setlength() ) Открытые массивы в Pascal обычно объявляются в качестве параметров процедур. (procedure sum(a:array of integer; n: integer):integer;)

1. **Работа со структурированными типами данных. Вычисление индекса элемента массива. Удаление и вставка элемента в массив.**

Работа со структурированными типами данных в языке программирования Pascal включает в себя работу с массивами, которые позволяют хранить однотипные элементы данных

Вычисление индекса элемента массива можно осуществить с помощью цикла, проходя по каждому элементу массива и сравнивая его с искомым значением. Когда сравниваемый элемент совпадает с искомым, можно вернуть индекс этого элемента.

begin

FindIndex := -1; // Изначально устанавливаем значение -1 (элемент не найден)

for i := 1 to length(array) do

begin

if array[i] = target then

begin

FindIndex := i; // Возвращаем индекс найденного элемента

Exit; // Выходим из функции

end;

end;

end;

Удаление элемента из массива:

Пример: Удалить из массива X(n) отрицательные элементы.

program upor\_massiv;

var

i,n,j:byte;

X: array [1..100] of real;

begin

writeln ('введите размер массива ');

readln (n);

for i:=1 to n do

begin

write('X[',i,']=');

readln (X[i]);

end;

writeln;

i:=1;

while(i<=n)do

if x[i]<0 then

begin

for j:=i to n-1 do

x[j]:=x[j+1];

n:=n-1;

end

Else

i:=i+1;

writeln('Измененный массив:');

for i:=1 to n do

write (X[i]:5:2,' ');

writeln;

end.

Вставка элемента в массив:

var i,n,m:byte; X: array [1..100] of real;

b:real;

begin

write ('N='); readln (n);

for i:=1 to n do

begin

write('X[', i ,']='); readln (X[i]);

end;

writeln ('m='); readln (m);

writeln ('b='); readln(b);

for i:=n downto m+1 do

x[i+1]:=x[i];

x[m+1]:=b;

n:=n+1;

writeln('Измененный массив');

for i:=1 to n do write (X[i]:5:2,' ');

writeln;

end.

1. **Организация работы со строками. Типы строковых переменных в Pascal. Хранение строк в памяти. Строки как частный случай одномерного массива.**

Организация работы со строками в языке Pascal включает в себя использование специальных строковых типов данных и операторов для работы со строками.

Типы строковых переменных – string

Строки в памяти хранятся как последовательность символов, где каждый символ занимает определенное количество байтов в зависимости от используемой кодировки.

В Pascal строки также можно рассматривать как частный случай одномерного массива, где каждый элемент массива представляет отдельный символ строки. Таким образом, мы можем работать со строкой как с массивом и использовать те же процедуры, что облегчает работу.

1. **Организация работы со строками. Функции работы со строками на Pascal.**

**Организация работы со строками в Pascal**предполагает использование стандартных процедур и функций:

1. length(s) Определяет текущую длину строки s.

2. pos(s1, s) Возвращает номер первой позиции подстроки **s1** в исходной строке **s**. Если подстроки нет, то выдает 0.

3. copy(s, p, k) Возвращает подстроку, выделенную из строки **s**, длиной **k** символов, начиная с символа под номером **p**.

4. insert(s1,s,p); Вставляет строку s1 в строку s, начиная с символа под номером p строки s.

5. delete(s,p,k); Удаляет из строки s подстроку длиной k символов, начиная с символа под номером p.

1. **Подпрограммы. Понятие процедуры и функции.**

Подпрограмма — это специальным образом оформленный алгоритм, который может многократно использоваться при решении более общей задачи.

Процедура - это подпрограмма, которая выполняет определенное действие, но не возвращает значение, имеющая любое количество входных и выходных данных. Может быть описана без параметров и с параметрами.

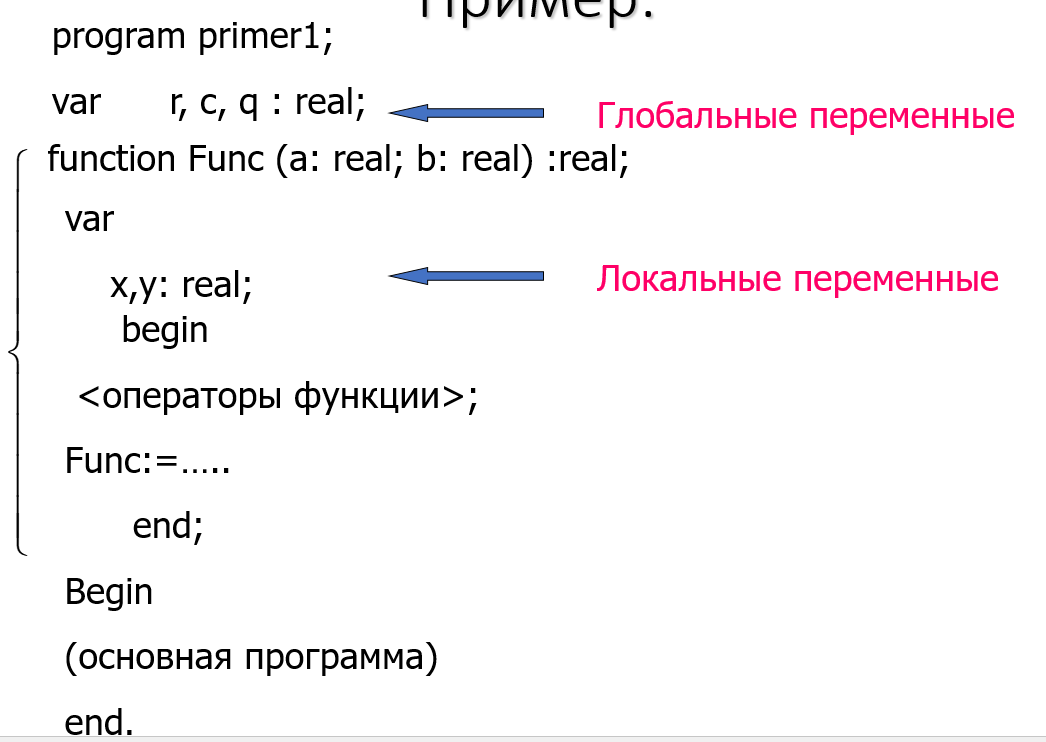
Функция - это последовательность операторов, имеющая имя и результат. Обрабатывает данные, предназначенные ей из главной программы, и затем возвращает полученный результат

1. **Подпрограммы. Локальные и глобальные переменные.**

Подпрограмма — это специальным образом оформленный алгоритм, который может многократно использоваться при решении более общей задачи.

***Глобальные переменные*** - переменные, объявленные в основной программе, доступны всем операторам программы, а так же операторам процедур и функций.

***Локальные переменные*** - переменные, объявленные в процедуре или функции. Они доступны только операторам процедур или функций.



1. **Подпрограммы. Формальные и фактические параметры.**

Подпрограмма — это специальным образом оформленный алгоритм, который может многократно использоваться при решении более общей задачи.

***Формальные параметры*** - это локальные переменные(определялись при изучении функции) программы, которые перечисляются в заголовке процедуры с обязательным указанием типа для каждого из них. Формальные параметры описываются только в заголовке процедуры и больше нигде. Формальные параметры при выполнении процедуры заменяются на конкретные значения

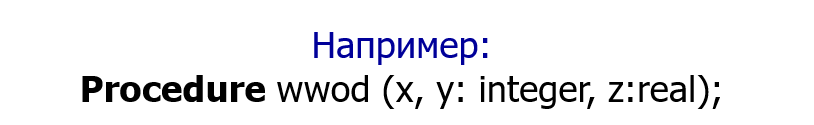
***Фактические параметры-*** это значения переменных или константы, которые передаются подпрограмме при обращении к ней.

1. **Подпрограммы. Механизм передачи параметров. Параметры – значения. Параметры – переменные**

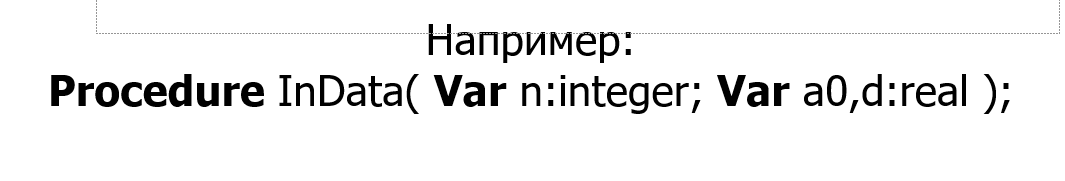
Подпрограмма — это специальным образом оформленный алгоритм, который может многократно использоваться при решении более общей задачи.

Механизм передачи параметров - это способ передачи данных в подпрограмму.

Формальный параметр называют ***параметром – значения (по значению)***, если передача фактического параметра формальному происходит только по его значению (как будто формальный параметр присвоил значение фактического параметра и работает с ним отдельно).



Формальный параметр называют ***параметром – переменным (по ссылке)***, если перед ним стоит оператор описания переменных **VAR** и передача фактического параметра формальному происходит по его адресу (как будто формальный параметр работает как фактический параметр, но с другим именем).



1. **Рекурсия и рекурсивные алгоритмы. Рекурсивная триада. Рекурсивный спуск и рекурсивный подъем**

Рекурсия - это процесс, когда функция вызывает саму себя внутри своего собственного кода.

Рекурсивные алгоритмы - это алгоритм, в определении которого содержится прямой или косвенный вызов этого же алгоритма. Рекурсивные алгоритмы обычно используются для решения задач, которые могут быть разбиты на более мелкие подзадачи того же типа.

Для решения задач рекурсивными методами разрабатывают следующие этапы, образующие **рекурсивную триаду**:

* **параметризация** – выделяют параметры, которые используются для описания условия задачи, а затем в решении;
* **база рекурсии** – определяют тривиальный случай, при котором решение очевидно, то есть не требуется обращение функции к себе;
* **декомпозиция** – выражают общий случай через более простые подзадачи с измененными параметрами.

Рекурсивный спуск - это процесс, когда рекурсивная функция последовательно вызывает саму себя с изменяющимися параметрами, пока не достигнет базового случая.

Рекурсивный подъем - это процесс, когда рекурсивная функция возвращает значения на каждом уровне рекурсии до тех пор, пока не завершит все рекурсивные вызовы и не вернет финальный результат

1. **Работа в графическом режиме. Настройка палитры. Функции Pascal для работы с графическими примитивами.**

Работа в графическом режиме позволяет создавать и отображать графические элементы на экране компьютера. Для работы в графическом режиме необходимо подключение модуля GraphABC, который предоставляет инструменты для создания графики на экране.

Настройка палитры:



**SetPenColor(color)** - устанавливает цвет пера, задаваемый параметром color.

**FloodFill(x,y,color)** - заливает область одного цвета цветом color, начиная с точки (x,y).

**SetBrushColor(color)** – устанавливает цвет кисти. Заливка кистью распространяется на замкнутый контур, описание которого следует за процедурой установки цвета кисти.

**SetBrushPicture(‘fname’)** - устанавливает в качестве образца для закраски кистью образец, хранящийся в файле fname, при этом текущий цвет кисти при закраске игнорируется.

**ClearWindow(color);** - очищает графическое окно указанным цветом.

Функции Pascal для работы с графическими примитивами включают в себя:

**SetPixel(x,y,color)** - Закрашивает один пиксел с координатами (x,y) цветом color.

**LineTo(x,y)** - рисует отрезок от текущего положения пера до точки (x,y); координаты пера при этом также становятся равными (x,y).

**Line(x1,y1,x2,y2)** - рисует отрезок с началом в точке (x1,y1) и концом в точке (x2,y2).

**Rectangle(x1,y1,x2,y2)** - рисует прямоугольник, заданный координатами противоположных вершин (x1,y1) и (x2,y2).

**Circle(x,y,r)** - рисует окружность с центром в точке (x,y) и радиусом r.

**Ellipse(x1,y1,x2,y2)** - рисует эллипс, заданный своим описанным прямоугольником с координатами противоположных вершин.

**Arc(x,y,r,a1,a2)** - Рисует дугу окружности с центром в точке (x,y) и радиусом r, заключенной между двумя лучами, образующими углы a1 и a2 с осью OX (a1 и a2 – вещественные, задаются в градусах и отсчитываются против часовой стрелки).

**Pie(x,y,r,a1,a2)** - рисует сектор окружности, ограниченный дугой (параметры процедуры имеют тот же смысл, что и в процедуре Arc).

**TextOut(x,y,’строка’);** - выводит строку текста в позицию (x,y) (точка (x,y) задает верхний левый угол прямоугольника, который будет содержать текст).

1. **Работа в графическом режиме. Экранная и декартова системы координат**

Работа в графическом режиме позволяет создавать и отображать графические элементы на экране компьютера Для работы в графическом режиме необходимо подключение модуля GraphABC, который предоставляет инструменты для создания графики на экране.

Экранная система координат представляет собой систему координат, где начало координат находится в верхнем левом углу экрана, а ось X увеличивается слева направо, а ось Y увеличивается сверху вниз.

**Для построения декартовой системы координат на языке Pascal** необходимо выполнить следующие шаги:

1. **Задать прямоугольник, в котором будет находиться система координат**. Для этого нужно указать координаты левого верхнего и правого нижнего углов системы координат (например, (xLeft; yLeft) — координаты левого верхнего угла, (xRight; yRight) — правого нижнего угла).
2. **Провести оси координат OX и OY**. Обе оси должны иметь положительные и отрицательные значения.
3. **Поставить центр координат (x0; y0)**. Для этого нужно знать границы изменения аргумента x по оси OX и значения функции f по оси OY.
4. **Построить график функции**. График строят по точкам, используя процедуру SetPixel(x, y, c), где x, y — координаты точки в графическом окне Паскаля, c — цвет точки. Для рисования осей координат OX и OY используют процедуру Line(x1, y1, x2, y2), где (x1; y1) — координаты начальной точки, (x2; y2) — координаты конечной.
5. **Работы в графическом режиме. Построение графиков функций.**

Работа в графическом режиме позволяет создавать и отображать графические элементы на экране компьютера. Для работы в графическом режиме необходимо подключение модуля GraphABC, который предоставляет инструменты для создания графики на экране.

Для построения графиков функций обычно используются математические формулы. Например, чтобы построить график функции y = sin(x), можно использовать функцию синуса и применить преобразования для перевода координат функции в координаты экрана. График строят по точкам, используя процедуру SetPixel(x, y, c), где x, y — координаты точки в графическом окне Паскаля, c — цвет точки.

1. **Организация работы с файлами. Алгоритм работы с типизированным файлом.**

Организация работы с файлами включает в себя создание, открытие, чтение, запись и закрытие файлов. Алгоритм работы с типизированным файлом включает следующие шаги:

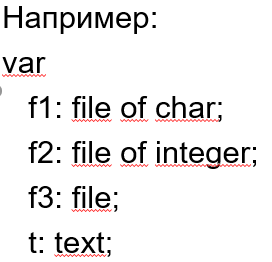
* 1. Объявление файловой переменной. **Var** <имя файла>: **File Of** <тип элементов>;

Переменную файлового типа описывают одним из трех способов:

file of *тип* - типизированный файл (указан тип компоненты);

text - текстовый файл;

file - нетипизированный файл.



* 1. Открытие файла: Используйте функцию Assign, чтобы связать переменную с именем файла.
  2. Чтение файла: Для чтения из файла необходимо открыть файл для чтения с помощью процедуры **RESET**(<имя файловой переменной>); Чтение данных из файла выполняется процедурой **READ** (<имя файловой переменной>,’<имя переменной>);

4. Запись данных в файл: Для записи в файл необходимо открыть файл для записи посредством процедуры

**REWRITE**(<имя файловой переменной>);

Запись данных в файл выполняется процедурой

**WRITE**(<имя файловой переменной>, <значение>);

5. Закрытие файла: В конце работы с файлом используйте процедуру Close, чтобы закрыть его.

1. **Организация работы с файлами. Алгоритм работы с текстовым файлом.**

Организация работы с файлами включает в себя создание, открытие, чтение, запись и закрытие файлов.

Алгоритм работы с текстовым файлом может быть следующим:

0. Объявление файловой переменной.

1. Для связывания файловой переменной с файлом на диске употребляется процедура **Assign**.

2. Для чтения данных применяется процедура **Read**. Если необходимо после чтения данных перейти на следующую строку, то используется процедура **Readln**.

3. Процедура **Write** записывает данные в текущую строку. Если надо записать данные и перейти к следующей строке, то можно использовать процедуру **Writeln**, которая записывает в файл признак конца строки и устанавливает файловый указатель на начало следующей строки.

4.Закрытие файла: когда работа с файлом закончена, необходимо закрыть его, чтобы сохранить изменения и освободить ресурсы. **close**

5.Удаление файла: если файл больше не нужен, его можно удалить с помощью команды удаления файла (**delete**).

Важно следить за правильностью именования файлов и выбором подходящего типа расширения для хранения данных. Также необходимо проверять корректность операций записи и чтения данных из файла, чтобы избежать ошибок и потери информации.

1. **Понятие сортировки. Сортировка пузырьком.**

Сортировка - последовательное расположение или разбиение на группы чего-либо в зависимости от выбранного критерия.

Алгоритм:

1. Каждый элемент сравнивается с последующим и если текущий элемент больше, происходит замена.
2. Повторяем шаг 1.

Отличительная особенность: очень прост в реализации, но неэффективен. Обычно используется в образовательных целях.

1. **Понятие сортировки. Сортировка подсчетом.**

Сортировка - последовательное расположение или разбиение на группы чего-либо в зависимости от выбранного критерия.

Сортировка подсчетом. Находим максимальное значение в массиве для определения размера вспомогательного массива. Создаётся вспомогательный массив и на старте заполняется нулями. Проходится по всему исходному массиву и смотрится очередное значение в ячейке. Содержимое этой ячейки берётся и увеличивается на единицу значение вспомогательного массива под этим номером. После цикла во вспомогательном массиве хранятся данные, сколько раз встречается каждый элемент. Теперь проходится по вспомогательному массиву, и если в очередной ячейке лежит что-то больше нуля, то в исходный массив столько же раз отправляется номер этой ячейки.

1. **Понятие сортировки. Сортировка вставками.**

Сортировка - последовательное расположение или разбиение на группы чего-либо в зависимости от выбранного критерия.

(англ. *Insertion sort*) — алгоритм сортировки, в котором элементы входной последовательности просматриваются по одному, и каждый новый поступивший элемент размещается в подходящее место среди ранее упорядоченных элементов.

То есть, сортировки вставками всегда делят массив на 2 части — отсортированную и неотсортированную. Из неотсортированной части извлекается любой элемент. Поскольку другая часть массива отсортирована, то в ней достаточно быстро можно найти своё место для этого извлечённого элемента. Элемент вставляется куда нужно, в результате чего отсортированная часть массива увеличивается, а неотсортированная уменьшается. Всё. По такому принципу работают все сортировки вставками.

1. **Понятие сортировки. Сортировка выбором**

Сортировка - последовательное расположение или разбиение на группы чего-либо в зависимости от выбранного критерия.

Сортировка выбором – основывается на выборе наименьшего (или наибольшего) элемента из неотсортированной части массива и его перемещении в начало (или конец) отсортированной части.

1. **Алгоритм сортировки Шелла**

Сортировка - последовательное расположение или разбиение на группы чего-либо в зависимости от выбранного критерия.

(англ. Shell sort) — алгоритм сортировки, являющийся усовершенствованным вариантом сортировки вставками.

Идея метода заключается в сравнение разделенных на группы элементов последовательности, находящихся друг от друга на некотором расстоянии. Изначально это расстояние равно d или N/2, где N — общее число элементов. На первом шаге каждая группа включает в себя два элемента расположенных друг от друга на расстоянии N/2; они сравниваются между собой, и, в случае необходимости, меняются местами. На последующих шагах также происходят проверка и обмен, но расстояние d сокращается на d/2, и количество групп, соответственно, уменьшается. Постепенно расстояние между элементами уменьшается, и на d=1 проход по массиву происходит в последний раз.