Программирование

Что это?

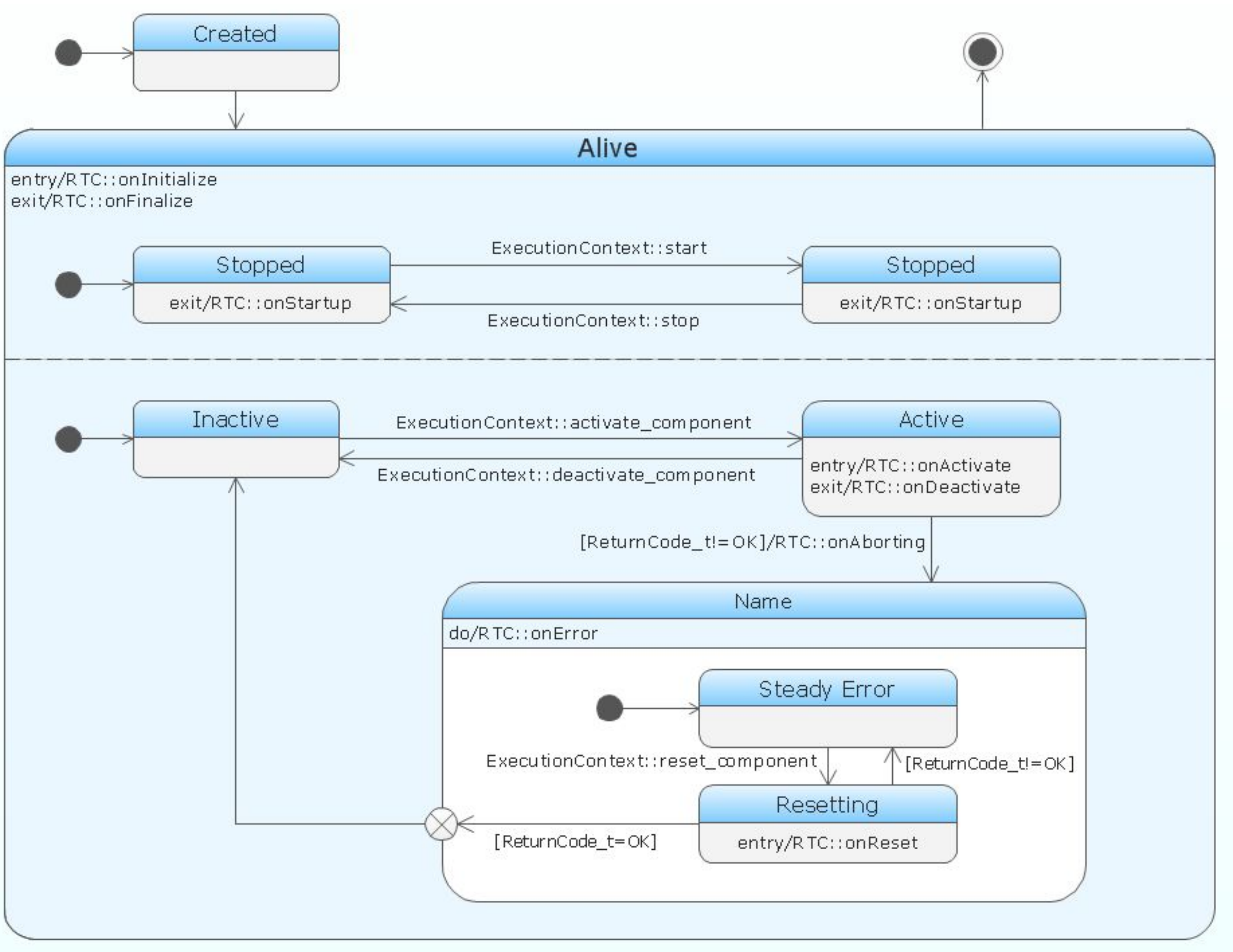
Программирование?

```
private int[] splitArrayParts(){
        System.out.println();
        int[] resultArray = new int[mainArray.length]; //массив с объединенными частями
        int i=0, j=0, k=0;
        int a[]=flows[0].getLocalArray(); //массив с первой отсортированной частью
        for (int n = 1; n < threadsCount; n++) {
                k=0; j=0; i=0;
                int b[]=flows[n].getLocalArray(); //массив со следующей отсортированной частью
                while (k<(a.length+b.length)){ //цикл до конца обоих массивов
                //если массив a уже кончился или если массив b еще не закончился и b[i]<=a[j]
                        if ((j>=a.length) || ((i<b.length) && (b[i]<=a[j]))){
                                //в результирующий массив записываем элемент массива b
                                resultArray[k]=b[i];
                                k++;
                                i++;
                        }
                        else { //иначе в результирующий массив записываем элемент массива a
                                resultArray[k]=a[j];
                                k++;
                                j++;
                        }
                }
                a=new int[k]; //пересоздаем массив a и увеличиваем его размер до k
                if (n!=threadsCount-1)     //если не последняя итерация цикла
                        //переносим все элементы результирующего массива в массив a,
                        for (int m=0; m<k; m++){
                                a[m]=resultArray[m];  //чтобы продолжить слияние
                        }
        }
        //возвращаем в качестве результата общий массив с объединенными частями
        return resultArray;
}
```

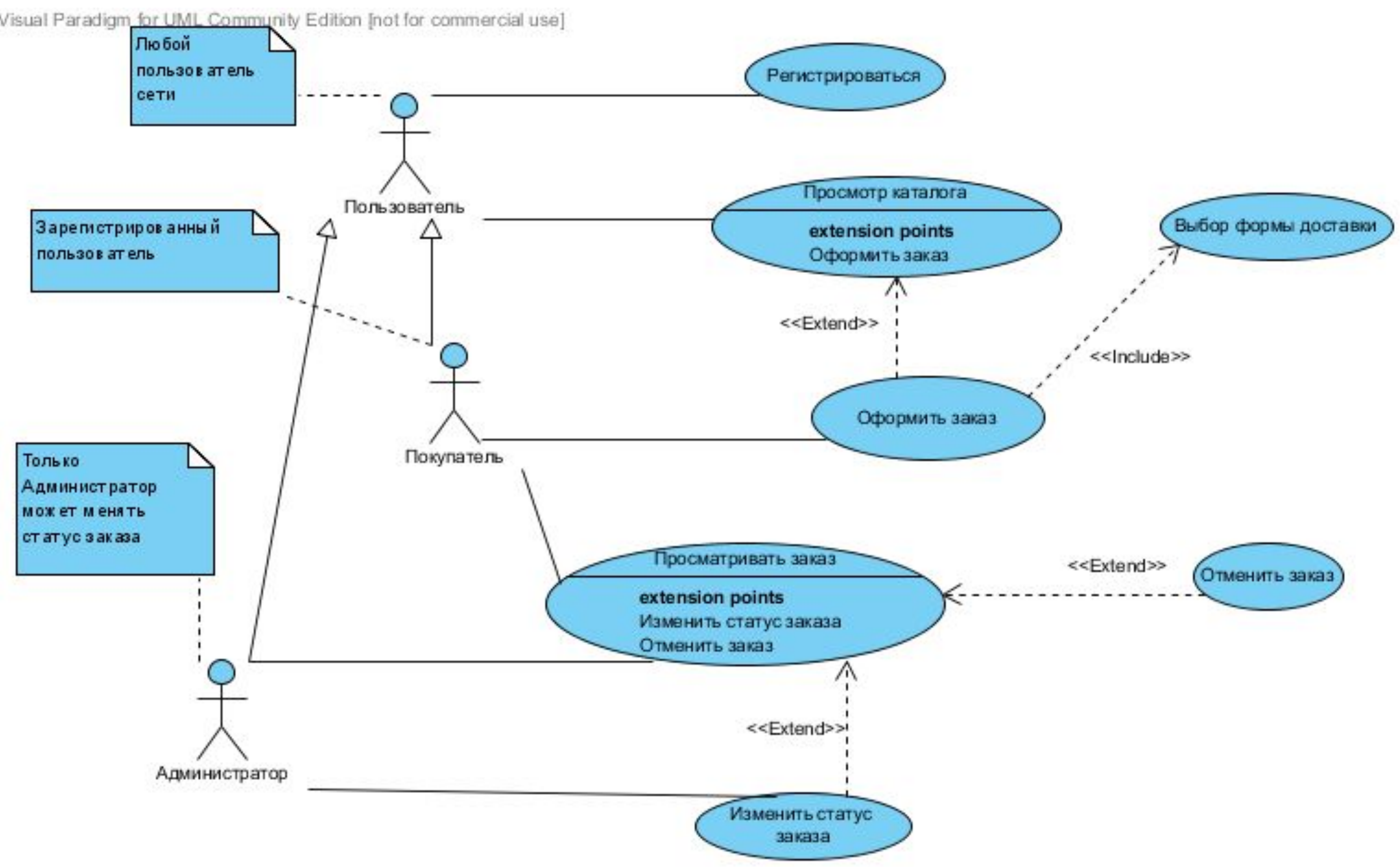
Программирование?

```
private int[] splitArrayParts(){
	System.out.println();
	int[] resultArray = new int[mainArray.length];
	int i=0, j=0, k=0;
	int a[]=flows[0].getLocalArray();
	for (int n = 1; n < threadsCount; n++) {
		k=0; j=0; i=0;
		int b[]=flows[n].getLocalArray();
		while (k<(a.length+b.length)){

			if ((j>=a.length) || ((i<b.length) && (b[i]<=a[j]))){

				resultArray[k]=b[i];
				k++;
				i++;
			}
			else {
				resultArray[k]=a[j];
				k++;
				j++;
			}
		}
		a=new int[k];
		if (n!=threadsCount-1)

			for (int m=0; m<k; m++){
				a[m]=resultArray[m];
			}
	}

	return resultArray;
}
```

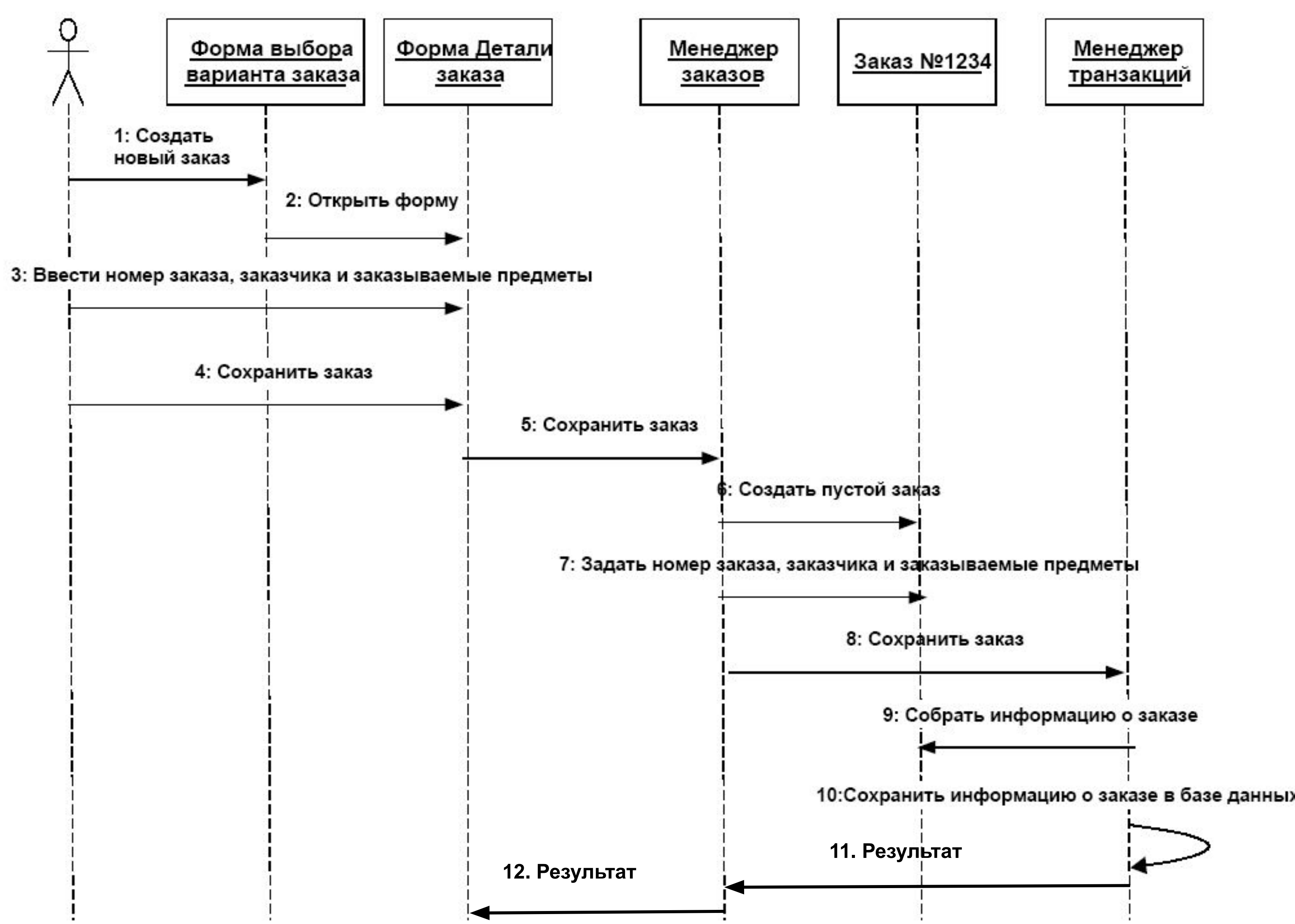
Программирование?



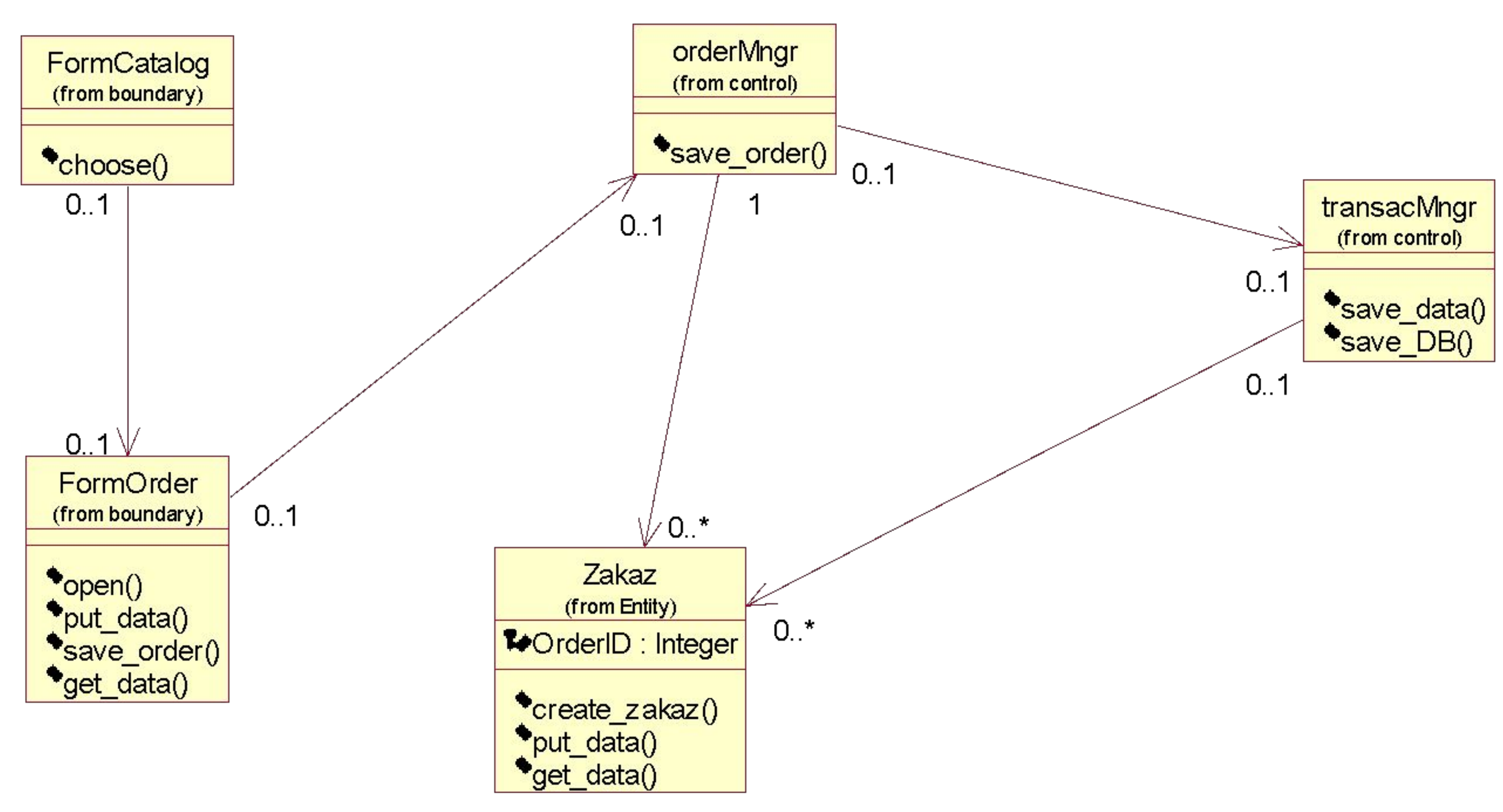
Программирование?



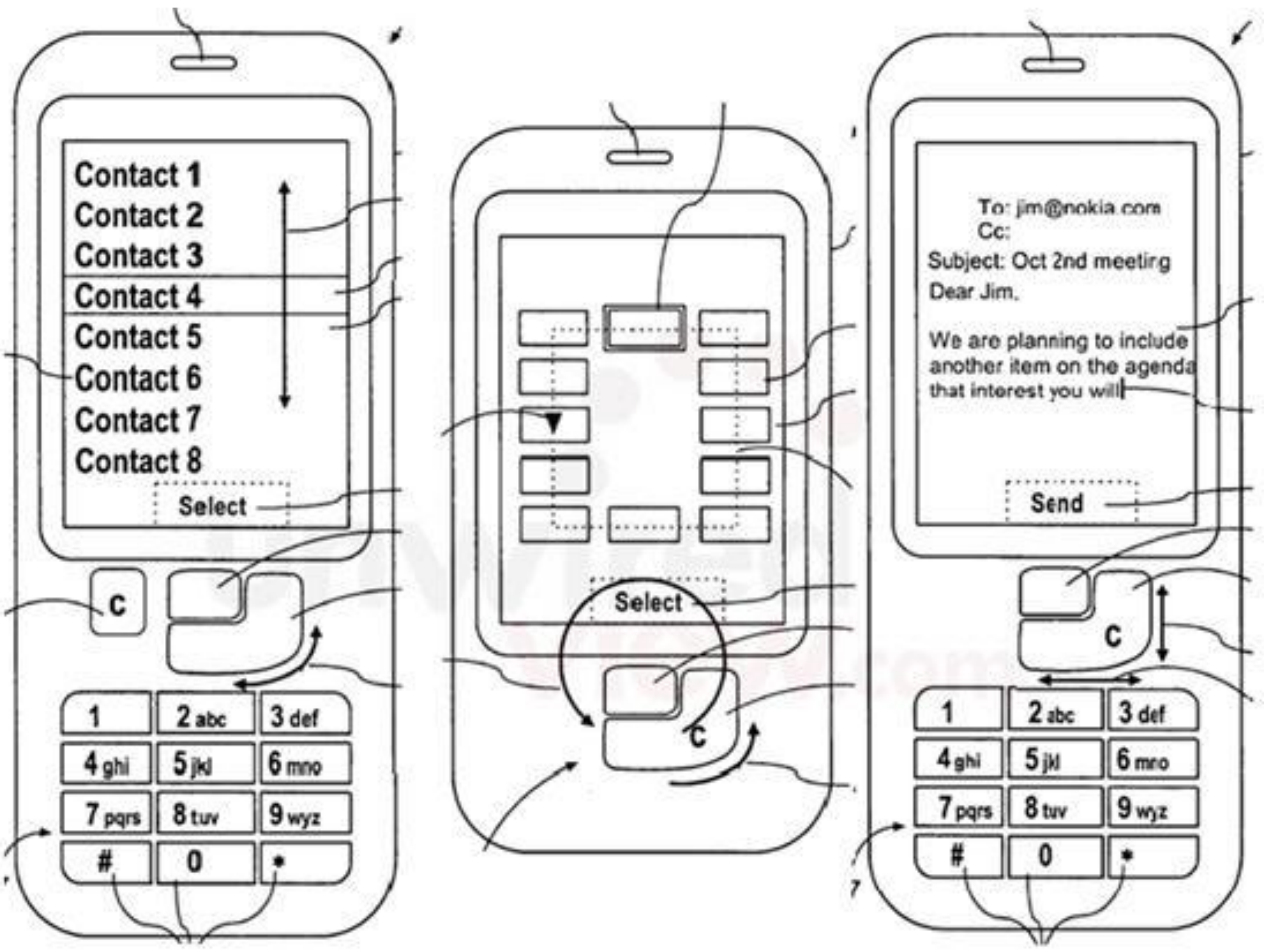
Программирование?



Программирование?



Программирование?



Культурная страничка

Художественная литература:

С. Лукьяненко «Лабиринт отражений»

Художественные фильмы:

«Хакеры» : Анджелина Джоли, The Prodigy

«Пираты Силиконовой Долины» : о Microsoft и Apple

сериал «Кремниевая долина» : реальные методы разработки ПО

Документальный фильм «Революционная ОС» про возникновение Linux

К чему приводят ошибки программистов

Неудача при запуске первого американского спутника к Венере случилась, вероятнее всего, из-за ошибки в программе – вместо требуемой в операторе запятой программист поставил точку. Вот как был записан этот оператор:

```
DO 50 I = 12.525
```

На самом же деле он должен был выглядеть следующим образом:

```
DO 50 I = 12,525
```

К чему приводят ошибки программистов

Одна из первых компьютерных систем противовоздушной обороны США (60-е годы) в первое же дежурство подняла тревогу, приняв восходящую из-за горизонта Луну за вражескую ракету, поскольку этот «объект» приближался к территории США и не подавал сигналов, что он «свой».

К чему приводят ошибки программистов

В **1985–87 гг.** 6 человек в США и Канаде получили смертельную дозу облучения во время сеансов радиационной терапии с применением медицинского ускорителя Therac-25. Эти дозы, как было оценено позже, более чем в 100 раз превышали те, что обычно применяются при лечении.

Расследование показало, что непосредственной причиной инцидентов была программная ошибка, однако основные ошибки были сделаны на стадии проектирования оборудования и системы автоматизации.

К чему приводят ошибки программистов

1988 год, причиной осложнений, возникших при возвращении на Землю из космической экспедиции советско-афганского (29 августа – 21 декабря 1988) и советско-французского (26 ноября – 27 апреля 1988) экипажей, явились ошибки, допущенные в программном обеспечении бортовых компьютеров.

К чему приводят ошибки программистов

В **1991 г.** ракетная установка MIM-104 Patriot не заметила вражескую ракету Scud, которая уничтожила в казарме г. Дахрен (Судовская Аравия) 28 американских солдат.

Бортовая система Patriot работает с накапливаемой ошибкой в системных часах, и в процессе длительного нахождения на боевом дежурстве погрешность достигла 0,3 с, что привело к пропуску быстро двигавшейся ракеты.

К чему приводят ошибки программистов

В **1994 г.** погибло 29 человек в результате аварии английского военного вертолета Chinook, который разбился из-за сбоя в бортовой навигационной системе, неверно определившей высоту полета.

К чему приводят ошибки программистов

4 июня 1996 года был произведен первый запуск ракеты-носителя Ariane 5 – детища и гордости Европейского Сообщества. Уже через неполные 40 сек. все закончилось взрывом. Автоподрыв 50-метровой ракеты произошел в районе ее запуска с космодрома во Французской Гвиане.

Только находившееся на борту научное оборудование потянуло на полмиллиарда долларов, не говоря о прочих разнообразных издержках; а астрономические цифры "упущенной выгоды" от несостоявшихся коммерческих запусков и потеря репутации надежного перевозчика в очень конкурентном секторе мировой экономики.

Причина – неправильное использование наследования при создании ПО.

К чему приводят ошибки программистов

1999 год, Аппарат для исследования Марса Mars Climate Orbiter был запущен 11 декабря 1998 года. Следом за ним был также запущен Mars Polar Lander – 3 января 1999. Оба аппарата были потеряны вскоре после того, как они достигли красной планеты.

Эти два космических корабля стоили NASA около 327,6 миллиона долларов, потраченных на их создание и функционирование.

Причина аварии Mars Polar Lander осталась невыясненной. Причина потери Mars Climate Orbiter заключается в программно-человеческой ошибке, которая привела к тому, что одно подразделение участвовавшее в проекте считало "***в дюймах***", а другое – "***в метрах***" , причем выяснилось это уже после потери аппарата.

К чему приводят ошибки программистов

Запуск ракеты-носителя "Зенит" **12 марта 2000 года** по программе "Морской старт" закончился аварией. Через несколько минут после старта ракета "Зенит" отклонилась от курса и не смогла вывести на заданную орбиту первый спутник для системы сотовой телефонной связи.

После аварии были образованы несколько комиссий для расследования ее причины. В опубликованных выводах экспертов компании Боинг причиной сбоя называется программная ошибка. Из-за этой ошибки не был закрыт клапан в пневматической системе второй

К чему приводят ошибки программистов

На заводе по переработке урана в Западной Австралии в **конце декабря 2001 года** произошел выброс радиоактивного вещества.

Расследование инцидента показало, что произошел он из-за "логической ошибки" в программном обеспечении компьютеров, установленных на заводе.

К чему приводят ошибки программистов

В августе 2003 года северо-восток США остался без электричества в том числе и из-за программной ошибки в системе управления аварийной сигнализацией. Переполнение памяти возникло из-за так называемых «гонок» (race condition) – ошибки программирования многозадачной системы, вызывающей неопределенность порядка выполнения различных частей кода.

Система попала в бесконечный цикл, тем самым оставив операторов без актуальной информации о состоянии энергосистемы. Если бы все было запрограммировано грамотно, то оператор смог бы предотвратить каскадные сбои и минимизировать убытки. По некоторым оценкам, ущерб составил от 7 млрд до 10 млрд долл.

К чему приводят ошибки программистов

Американский аппарат Mars Global Surveyor прибыл к Красной планете в 1997 г. Аппарат прекратил функционирование в **ноябре 2006 г.:** из-за ошибка адресации бортового ПО произошел перегрев батареи с последующим отказом других устройств.

11 февраля 2007 г. 12 истребителей-"невидимок" F-22 перелетали с военной базы США на Гавайях в Японию. В момент пересечения международной временной границы на всех машинах из-за программной ошибки отказали бортовые компьютеры.

Известны также случаи, когда вследствие программной ошибки истребители F-16 в режиме автопилотирования переворачивались "вверх ногами" при преодолении экватора.

К чему приводят ошибки программистов

5 декабря 2010 года: Три спутника, критически важные для завершения составления группировки российской навигационной системы ГЛОНАСС упали в Тихий океан недалеко от Гавайских островов вскоре после их запуска ракетой «Протон-М».

Финансовые потери оцениваются в 4 миллиарда рублей (138 миллионов долларов). В результате расследования виной аварии была признана ошибка в программировании, которая привела к тому, что в ракету залили неправильное количество топлива.

Главная задача программирования – это снижение сложности.

Цель программирования – описание процессов обработки данных.

Данные (data) – это представление фактов и идей в формализованном виде, пригодном для передачи и переработке в некоем процессе.

Информация (information) – это смысл, который придается данным при их представлении.

Обработка данных (data processing) – это выполнение систематической последовательности действий с данными.

Интеллектуальные возможности человека

Выделяют 3 интеллектуальные возможности человека, используемые при программировании:

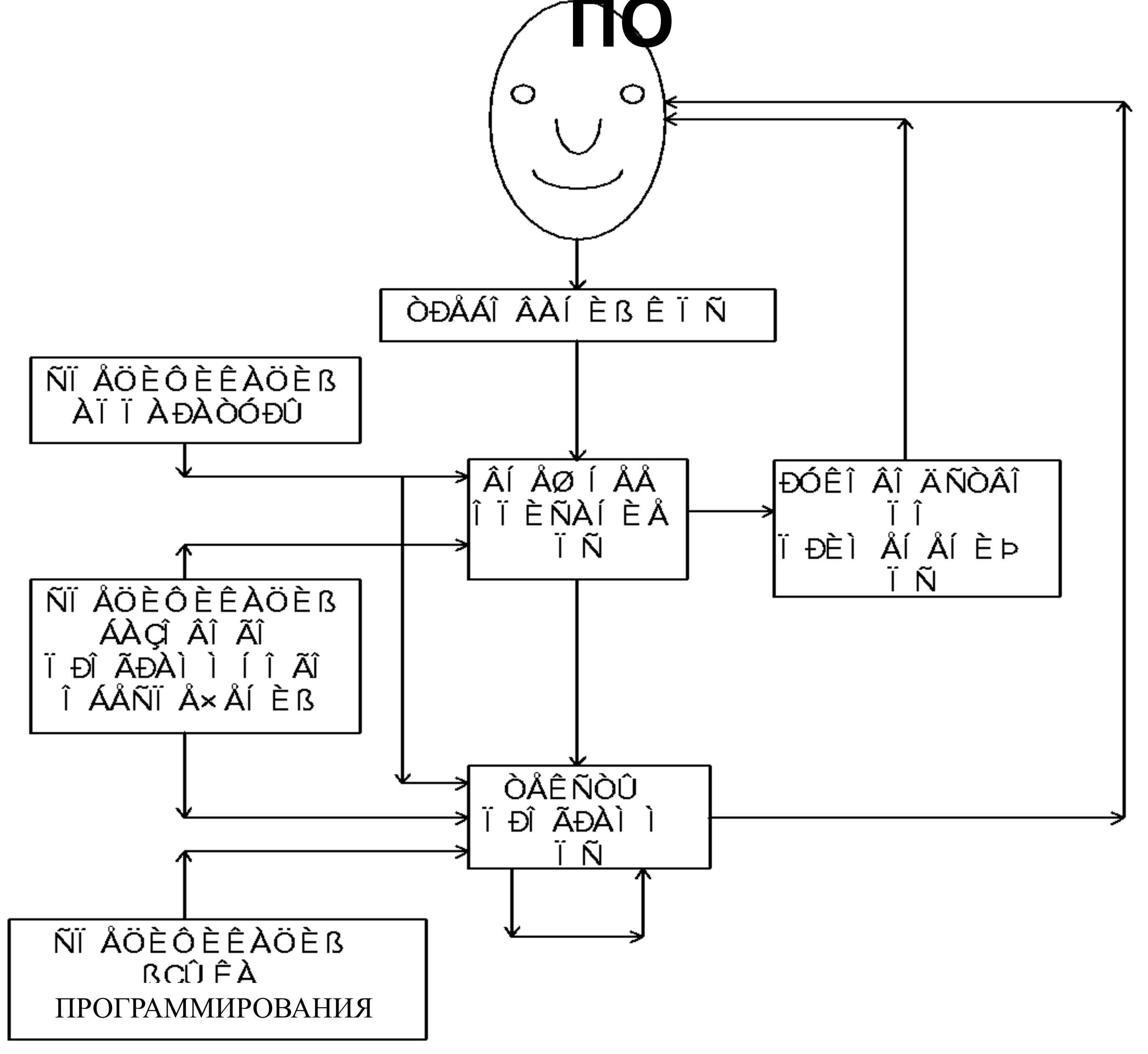
- способность к перебору,
- способность к абстракции,
- способность к математической индукции.

Способность человека ***к перебору*** связана с возможностью последовательного переключения внимания с одного предмета на другой, позволяя узнавать искомый предмет. В среднем человек может уверенно (не сбиваясь) перебирать в пределах **1000** предметов (элементов).

Средством преодоления этой ограниченности является его ***способность к абстракции***, благодаря которой человек может объединять разные предметы или экземпляры в одно понятие, заменять множество элементов одним элементом (другого рода).

Способность человека к ***математической индукции*** позволяет ему справляться с бесконечными последовательностями

Грубая схема разработки и применения ПО



Программирование и алгоритм

Программирование –

это составление программ для вычислительной машины, описывающих алгоритм решения определенных задач.

Другими словами – это создание алгоритма решения задачи и его представление в виде программы.

Задача определяется входными и выходными данными и связями между ними.

Алгоритм —

это строгая и четкая конечная система правил, которая определяет последовательность действий над некоторыми объектами и после конечного числа шагов приводит к решению задачи.

Под *действием* понимается нечто, что имеет конечную продолжительность и приводит к желаемому и совершенно определенному результату.

Каждое действие предполагает наличие некоторого объекта, над которым это действие совершается и по изменению состояния которого можно судить о результате действия.

Действие должно быть таким, чтобы его можно было описать с помощью некоторого языка. Это описание называется **инструкцией**.

Программа – это инструкции, записанные таким образом, чтобы они были «понятны» вычислительной машине.

ЭТАПЫ ПРОГРАММИРОВАНИЯ

1. Математическая формулировка задачи.
2. Выбор метода решения задачи.
3. Разработка алгоритма.
4. Описание алгоритма на алгоритмическом языке (получение программы).
5. Тестирование программы.
6. Проведение расчетов, анализ

Алгоритм

Это описание процесса решения некоторой задачи.

Свойства алгоритма:

– *дискретность,* т.е. процесс решения протекает в виде последовательности отдельных действий, следующих друг за другом;

– *элементарность действий,* т.е. каждое действие является настолько простым, что не вызывает сомнений и возможности неоднозначного толкования;

– *детерминированность* (определенность), т.е. каждое действие однозначно определено и после выполнения каждого действия однозначно определяется, какое действие надо выполнить следующим;

– *конечность,* т. е. алгоритм заканчивается после конечного числа действий (шагов);

– *результативность,* т. е. в момент прекращения работы алгоритма известно, что считать его результатом;

– *массовость,* т. е. алгоритм описывает некоторое множество процессов, применимых при различных входных данных.

Критерии качества алгоритма

- *правильность* (алгоритм решает поставленную задачу);
- *прозрачность* (простота, удобочитаемость алгоритма);
- *эффективность* (быстродействие и краткость).

Блок-схема как одна из форм записи алгоритма

Наименование	Обозначение	Функция
Блок начало-конец (пуск-остановка)		Элемент отображает выход во внешнюю среду и вход из внешней среды (наиболее частое применение − начало и конец программы). Внутри фигуры записывается соответствующее действие.
Блок действия		Выполнение одной или нескольких операций, обработка данных любого вида (изменение значения данных, формы представления, расположения). Внутри фигуры записывают непосредственно сами операции, например, операцию присваивания: a = 10*b + c.

Блок-схема как одна из форм записи алгоритма

Наименование	Обозначение	Функция
Логический блок (блок условия)		Отображает решение или функцию переключательного типа с одним входом и двумя или более альтернативными выходами, из которых только один может быть выбран после вычисления условий, определенных внутри этого элемента. Вход в элемент обозначается линией, входящей обычно в верхнюю вершину элемента. Если выходов два или три, то обычно каждый выход обозначается линией, выходящей из оставшихся вершин (боковых и нижней). Если выходов больше трех, то их следует показывать одной линией, выходящей из вершины (чаще нижней) элемента, которая затем разветвляется.

Блок-схема как одна из форм записи алгоритма

Наименование	Обозначение	Функция
Данные (ввод-вывод)		Преобразование данных в форму, пригодную для обработки (ввод) или отображения результатов обработки (вывод).
Цикл со счетчиком	Действия	Повторяющаяся определенное количество раз часть алгоритма.
Любой цикл	Начало Конец	
Соединитель		Символ отображает вход в часть схемы и выход из другой части этой схемы. Используется для обрыва линии и продолжения её в другом месте (для избежания излишних пересечений или слишком длинных линий, а также, если схема состоит из нескольких страниц). Соответствующие соединительные символы должны иметь одинаковое (при том уникальное) обозначение (числа внутри кругов).

Пример блок-схемы

Рассмотрим задачу: найти максимум из двух чисел.

Язык программирования Pascal

Разработан в 1971 г. швейцарским профессором Николасом Виртом для обучения структурному программированию.

Общий вид структуры программы на языке Pascal

```
program <ИМЯ>;                          — заголовок программы

label  <описание меток>               }
const  <описание констант>            }
type  <описание типов>                } — раздел описаний
var  <описание переменных>            }
<описание процедур и функций>         }

begin                       }
     <оператор>;            }
     <оператор>;            } — раздел операторов
          .   .   .         }
     <оператор>             }
end.
```

Всякий язык программирования имеет три основные составляющие: *алфавит, синтаксис* и *семантику.*

Алфавит языка – это множество символов, которые можно использовать для записи правильных программ.

Синтаксис языка – это совокупность правил построения допустимых конструкций языка, форма их сочетаний при записи алгоритма, т.е. то что определяет правильность программ.

Семантика – это смысл конструкций языка, в том числе и программ, написанных на этом языке.

Основные символы языка Pascal

Это латинские буквы, цифры от 0 до 9 и специальные символы

+ — * / = , . : ; < > [ ] ( ) { } ^ @ $ #

Также есть служебные слова, которые не могут использоваться в качестве идентификаторов (т.е. имен переменных, подпрограмм, модулей). Например, слова **`var`**, **`type`**, ***`if`***.

Задание: найти в литературе и выписать все служебные слова языка Pascal.

Идентификаторы

Это имена переменных, констант, подпрограмм, модулей. В программе **не может быть двух идентификаторов с одним именем**!

Правильный идентификатор **должен начинаться** с латинской буквы. В нем могут присутствовать цифры и знак подчеркивания.

Примеры:

x

X

summa

s1

m_

Типы данных

Объекты (константы, переменные, функции, выражения), которыми оперирует программа, относятся к определенному типу.

Тип — это множество значений, которые могут принимать объекты программы, и совокупность операций, допустимых над этими значениями.

Типы данных

Идентификатор	Длина (байт)	Диапазон значений	Операции
Целые типы			
integer	2	-32768 .. 32767	+, -, /, *, Div, Mod, >=, <=, =, <>, <, >
byte	1	0 .. 255	
word	2	0 .. 65535	
shortint	1	-128 .. 127	
longint	4	-2147483648 .. 2147483647	
Вещественные типы			
real	6	$2{,}9x10^{-39}$ - $1{,}7x10^{38}$	+, -, /, *, >=, <=, =, <>, <, >
single	4	$1{,}5x10^{-45}$ - $3{,}4x10^{38}$	
double	8	$5x10^{-324}$ - $1{,}7x10^{308}$	
extended	10	$3{,}4x10^{-4932}$ - $1{,}1x10^{4932}$	
Логический тип			
boolean	1	true, false	Not, And, Or, Xor, >=, <=, =, <>, <, >
Символьный тип			
char	1	все символы кода ASCII	+, >=, <=, =, <>, <, >

Переменная

Данные хранятся в памяти компьютера, но для указания на конкретную информацию очень неудобно все время записывать физические адреса ячеек.

Эта проблема в языке Pascal решена введением понятия переменной. Переменная – именованный участок памяти для хранения данных определенного типа. Значение переменной (информация в соответствующих ячейках памяти) в ходе выполнения программы может быть изменено. **Имя переменной должно быть правильным идентификатором!**

Объявляются переменные в специальном разделе `var` (см. структуру программы на языке Pascal). Например:

```
var
 x: integer;
 b, summa: integer;
 a: real;
```

Константа

Величина, значение которой в ходе выполнения программы не может быть изменено.

Константы бывают обычные (просто значения, например, 5, 6.7, ‘f’) и именованные.

Имя константы должно быть правильным идентификатором.

Именованные константы объявляются в разделе `const`.

Пример:

```
const
      pi=3.14;
      n=20;
```

Оператор присваивания

```
:=
```

<Переменная> := <Выражение> ;

В левой части может быть только 1 переменная, которой будет присвоено значение выражения из правой части. Тип переменной слева должен соответствовать типу выражения справа.

Выражение состоит из операндов, знаков операций и круглых скобок.

Операндами являются константы, переменные, обращения к функциям.

Примеры:

```
a:=6;
s:=a+5;
y:=sin(x);
z:=y+cos(x);
```

Ввод данных

```
readln(<переменные>);
```

Пример 1:

```
readln(x);
```

Программа приостановит свое выполнение и будет ожидать ввод данных (в зависимости от типа переменной x). Завершается ввод нажатием клавиши Enter.

Пример 2:

```
readln(a,b,c);
```

Ввод нескольких переменных в одном выражении. При вводе с клавиатуры данные должны разделяться пробелом.

Вывод данных на экран

```
write(<выводимые данные>);
writeln(<выводимые данные>);
```

Во втором случае после вывода на экран будет произведен переход на следующую строку.

Примеры:

1. `write(‘Hello,’);`

```
write(‘ world!’);
```

На экране появится:

```
Hello, world!
```

2. `writeln(‘Hello,’);`

```
writeln(‘ world!’);
```

На экране появится:

```
Hello,
 world!
```

Особенности вывода

По умолчанию вывод происходит в поле вывода шириной в количество знаков выводимого числа. При этом вещественные числа выводятся с максимально возможным количеством знаков после вещественной точки.

Например:

```
x:=5/3;
writeln(x);
```

На экране появится

```
1.66666666666667
```

Но есть возможность управлять как количеством знаков в дробной части, так и шириной поля вывода числа.

Особенности вывода

Формат использования:

```
var x: integer;
…
write(x:n);
```

n – количество знакомест, отводимых для вывода числа (неиспользуемые знакоместа заменяются пробелами перед выводимым числом).

или

```
var x: real;
…
write(x:n:m);
```

n – количество знакомест, отводимых для вывода числа; m – количество знаков в дробной части.

Встроенные функции языка Pascal

Функция	Результат функции
abs(x)	Модуль числа (абсолютное значение)
sqr(x)	Квадрат числа
sqrt(x)	Квадратный корень числа
pi	Число пи ~ 3.14...
sin(x)	Синус числа
cos(x)	Косинус числа
ln(x)	Логарифм натуральный
exp(x)	Экспонента (число е в степени х)
int(x)	Целая часть числа
trunc(x)	Целое число без округления
round(x)	Целое число с округлением

Пример использования

Найти сумму квадратов двух чисел.

Условный оператор в языке Pascal

Условные операторы позволяют выбирать для выполнения те или иные части программы в зависимости от некоторых условий.

Синтаксис условного оператора:

```
if <условие> then
begin
…
end
else
begin
…
end;
```

Т.е. если условие верное, то выполняется блок операторов после `then`, иначе выполняется блок операторов после `else`.

Операторы сравнения

Используются в условиях.

Оператор	Операция	Тип результата
=	равно	boolean
<>	не равно	boolean
<	меньше	boolean
>	больше	boolean
<=	меньше или равно	boolean
>=	больше или равно	boolean

Например: `if a<=b then … else …`

Запишем ранее созданную блок-схему нахождения максимума на языке Pascal.

Логические операции

Знак	Операция	Примеры
not	Логическое НЕ	`if not (x<y) then writeln('x >= y')` `else writeln('x<y');`
and	Логическое И	`if (x<y) and (y<z) then writeln('x < y < z');`
or	Логическое ИЛИ	`if (x=2) or (y=2) then writeln('Что-то равно 2');`

Таблицы истинности логических операций

1 – true, 0 – false

and	or
0 and 0 = 0	0 or 0 = 0
0 and 1 = 0	0 or 1 = 1
1 and 0 = 0	1 or 0 = 1
1 and 1 = 1	1 or 1 = 1

Пример

Найти максимум из двух чисел, введенных пользователем.

Оператор выбора *CASE*

Позволяет выбрать одно из нескольких возможных продолжений программы в зависимости от значения выражения:

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

Оператор выбора *CASE*

Пример 1:

```
Write('Введите  число: ');
  Readln( i );
  Case i of
      2, 4, 6, 8: Writeln('Четная цифра');
      1, 3, 5, 7, 9: Writeln('Нечетная цифра');
      10..100: Writeln('Число от 10 до 100');
    else
      Writeln ('Отрицательное число или больше 100')
  end;
```

Пример 2:

```
Write('Введите  номер месяца: ');
  Readln( MONTH );
  case MONTH of
      1, 2, 3 : writeln ('Первый квартал');
      4, 5, 6 : writeln ('Второй квартал');
      7, 8, 9 : writeln ('Третий квартал');
      10, 11, 12 : writeln ('Четвёртый квартал');
  end;
```

Операторы цикла

Цикл с предусловием while

Цикл – это последовательность операторов, которая может выполняться более одного раза.

Циклы с предусловием используются тогда, когда выполнение цикла связано с некоторым логическим условием. Оператор цикла с предусловием имеет две части: условие выполнения цикла и тело цикла.

При выполнении оператора while определенная группа операторов выполняется до тех пор, пока определенное в операторе while условие истинно. Если условие сразу ложно, то оператор не выполнится ни разу.

```
while <условие> do
begin
  группа операторов
end;
```

Цикл с предусловием while

При использовании цикла с предусловием надо помнить следующее:

- значение условия выполнения цикла должно быть определено до начала цикла;
- если значение условия истинно, то выполняется тело цикла, после чего повторяется проверка условия. Если условие ложно, то происходит выход из цикла;
- хотя бы один из операторов, входящих в тело цикла, должен влиять на значение условия выполнения цикла, иначе цикл будет повторяться бесконечное число раз.

Цикл с предусловием while

Задача: найти сумму чисел, введенных пользователем.

```
Program Summa;
Var
    i, N : integer;
    x, S : real;
Begin
  write('Сколько чисел для сложения? ');
  readln (N);
  S:=0;
  i:=1;
  while i<=N do
  begin
    write('Введите ',i,'-е число ');
    readln (x);
    S:=S+x;
    i:=i+1;
  end;
  write('Сумма введенных чисел равна ',s:5:2);
End.
```

Цикл с предусловием while

Задача: Найти сумму цифр в записи данного натурального числа;

```
Program SUM;
Var a,b,s,k:Integer;
Begin
    write('Введите число: ');
    Readln(a);
    s:=0;
    While a<>0 do
    begin
         b:=a mod 10;
         s:=s+b;
         a := a div 10;
    end;
   Writeln(s);
End.
```

Цикл с постусловием repeat

Отличительной особенностью данного цикла является то, что тело цикла выполняется в любом случае как минимум 1 раз, т.к. условие выхода из цикла проверяется после тела цикла.

```
repeat
  группа операторов
until <условие>;  {до тех пор, пока условие не будет верным}
```

Для выполнения в цикле repeat нескольких операторов не следует помещать эти операторы в операторные скобки **begin** ... **end**. Зарезервированные слова **repeat** и **until** действуют как операторные скобки.

Примеры:

a)
```
repeat
     read (Number);
     Sum := Sum+Number;
until Number=-1;
```

b)
```
repeat
     i := i+1;
     writeln (Sqr(i))
until Number=-1;
```

Цикл с постусловием repeat

Задача. Определить, является ли введенное число простым.

```
Program Prostoe;
Var
  i, {возможный делитель}
  Number : integer; {исследуемое число}
Begin
  writeln ('Какое число должно быть проверено? ');
  read (Number);
  i := 1;
  repeat
    i := i+1;
  until Number mod i = 0;
    if Number=i
      then
        writeln (Number,' является простым')
      else
        writeln (Number,' делится на ',i);
  readln;
End.
```

Цикл со счетчиком for

Цикл со счетчиком представляет такую конструкцию, в которой выполнение тела цикла должно повторяться заранее определенное число раз.

```
for i := A to B do
begin
     операторы
end;
Или
for i := A downto B do
begin
     операторы
end;
```

Переменная i – управляющая переменная или переменная цикла (целый тип),

A – начальное значение переменной цикла,

B – конечное значение переменной цикла.

Управляющую переменную цикла нельзя менять в теле цикла!!!

Цикл со счетчиком for

При переходе к обработке оператора цикла for управляющей переменной присваивается заданное начальное значение.

Затем в цикле выполняется исполнительный оператор (или составной оператор).

Каждый раз при выполнении исполнительного оператора управляющая переменная увеличивается на 1 (для for...to) или уменьшается на 1 (для for...downto).

Цикл завершается при достижении управляющей переменной своего конечного значения.

Цикл со счетчиком for

Примеры:

```
for i := 1 to n do
begin
    readln (Number);
    S := S +Number;
end;
```

```
for Range := Number+1 to Multi*3
do
    writeln (Sqrt(Range));
```

```
for Dlina := 15 downto 1 do
    writeln (Sqr(Dlina));
```

```
for x := 1 to 10 do
  for y := 1 to 10 do
    writeln (x,'*',y,'=`,x*y);
```

Вложенный цикл со счетчиком for

Часто исполнительная часть одного из циклов For является новым оператором цикла For.

Структуры такого рода называются **вложенными** циклами.

При завершении внутреннего цикла управляющая переменная внешнего цикла увеличивается, а внутренний цикл начинается заново.

Повторение этих действий будет продолжаться до завершения внешнего цикла.

Приведенный ниже вложенный цикл печатает пары чисел, начиная от (1,1), (1,2),... и кончая (10,10):

```
for x:= 1 to 10 do
  for y:= 1 to 10 do
    writeln (‘(‘,x,’,’,y,’), ’);
```

Пример цикла for

Вычислить N! (факториал):

```
Program Faktorial;
Var n, i, f: integer;
Begin
    f:=1;
    Write('Введите n: ');
    Readln(n);
    For i:=2 to n do
         f:=f*i;
    Writeln(n,'!=',f);
End.
```

Подпрограммы

Подпрограмма – это часть программы, оформленная в виде отдельной синтаксической конструкции и снабженная именем.

Вызов подпрограммы (т.е. выполнение действий, заданных в подпрограмме) может быть произведен в некоторой точке программы посредством указания имени этой подпрограммы. Обмен информацией между программой и подпрограммой может быть реализован с помощью формальных и фактических параметров.

Формальный параметр – это идентификатор, который используется в подпрограмме для обозначения объектов, к которым применяется заданная в ней последовательность действий. Фактический параметр – это объект, который подставляется в подпрограмму вместо соответствующего формального параметра.

Фактическими параметрами могут быть константы, имена переменных, выражения любого, заранее описанного типа. Формальные параметры указываются в описании подпрограммы, а фактические – в обращении к подпрограмме.

Подпрограммы

В языке Pascal подпрограммы представлены двумя видами: процедуры и функции.

Процедура предназначена для выполнения какой-то законченной последовательности действий.

Любая процедура начинается с заголовка. В отличии от основной программы заголовок в процедуре обязателен. Он состоит из зарезервированного слова Procedure, за которым следует идентификатор имени процедуры, а далее в круглых скобках список формальных параметров:

```
Procedure <имя процедуры> (список формальных параметров);
```

За заголовком могут идти такие же разделы, что и в основной программе. В отличии от основной программы процедура завершается не точкой, а точкой с запятой.

Пример программы с процедурой:

Нахождение факториала числа. Вариант 1:

```
program f;
var
    x: integer;

procedure factorial(n: integer);
var
    fact,i: integer;
begin
   fact:=1;
   for i:=1 to n do
   begin
     fact:=fact*i;
   end;
   writeln(n,'!=',fact);
end;

begin
  write('Введите число: ');
  readln(x);
  factorial(x);
end.
```

Нахождение факториала числа. Вариант 2:

```
program f;
var
    x,fa: integer;

procedure factorial(n: integer; var fact: integer);
var
    i: integer;
begin
   fact:=1;
   for i:=1 to n do
   begin
     fact:=fact*i;
   end;
end;

begin
  write('Введите число: ');
  readln(x);
  factorial(x,fa);
  writeln(x,'!=',fa);
end.
```

Подпрограммы-функции

Функция – это часть программы, которая вычисляет и возвращает значение.

Формат описания функции:

```
function имя функции (формальные параметры):тип результата;
раздел описаний функции
begin
   исполняемая часть функции
   имя функции := результат;
end;
```

Обязательно!!! В теле функции должно быть присвоение имени функции какого-либо значения!

Пример программы с функцией:

Вычисление факториала:

```
program f1;
var a:integer;

Function Factorial(N:Byte):Longint; // определение функции
Var Fact:longint; I:byte;
Begin
   Fact:=n;
   For i:=n-1 downto 2 do
      Fact:=fact*i;
   Factorial:=fact; //имени функции обязательно присваиваем
//значение, которое она будет возвращать
End;

begin
   write('Введите число:');
   readln(a);
   writeln('факториал вашего числа равен ',Factorial(a));
end.
```

Принцип локализации

Если объект (константа, переменная, процедура, функция или тип) описан в некоторой подпрограмме, то он называется локальным объектом подпрограммы.

Описание локальных объектов позволяет ограничивать область использования объекта той подпрограммой, в которой он имеет смысл.

Подпрограмма, т.е. фрагмент текста программы, для которого действительно описание объекта, называется его областью видимости.

В теле подпрограммы, наряду с локальными объектами, используются так называемые нелокальные объекты.

Любой объект, упоминаемый в подпрограмме, но не описанный в ней, является нелокальным. Если его описание принадлежит главной программе, то нелокальный объект называют глобальным.

Существование объектов, локальных в некоторой области видимости, позволяет, например, использовать один и тот же идентификатор для обозначения разных объектов, определенных в разных подпрограммах.

Область видимости

Правила для определения области видимости:

1. Областью видимости идентификатора является блок, в котором он описан, и все вложенные блоки.

2. Если идентификатор, описанный в блоке, повторно описан во вложенных в него блоках, то вложенные блоки исключаются из области видимости идентификатора.

3. В ряде случаев (в частности, при работе с модулями), если имеются одноименные локальный и нелокальный объекты, можно получить доступ к нелокальному объекту, расширив его область видимости. Это достигается указанием имени модуля, в котором он описан, перед именем объекта.

Принцип локализации

Временем жизни объекта называется время, в течение которого он существует в памяти вычислительной машины.

Так как областью существования локального объекта является тело подпрограммы, размещение его в памяти происходит при входе в подпрограмму.

Значения локальных объектов не определены при входе в подпрограмму (первом и повторном). После завершения подпрограммы он уничтожается и значение его теряется.

Время жизни для локальных объектов — это время с момента вызова подпрограммы до ее завершения.

Для глобальных объектов время жизни — это время выполнения программы. Следовательно, для сохранения значений переменных между вызовами подпрограммы следует продлить ее время жизни, т.е. пользоваться нелокальными переменными, в частности глобальными.

Структуры данных

Массив, строка, запись и множество называют фундаментальными структурами.

Переменные таких типов могут менять только своё значение, сохраняя местоположение в памяти и размер памяти, занимаемой ими.

Различают логические и физические уровни структурирования данных.

Логическая структура данного — идеальная схема представления или модель данного с точки зрения пользователя.

Физическая структура данного — это схема размещения или хранения данного в вычислительной машине. Говорят, что структура данного отображается в структуру хранения.

Общая классификация структур данных

В зависимости от отсутствия или наличия явно заданных связей между элементами данных следует различать несвязные структуры (векторы, массивы, строки) и связные структуры (связные списки).

Весьма важный признак структур данных – их изменчивость (изменение числа элементов и/или связей между элементами структур). По признаку изменчивости различают структуры статические (векторы, массивы, записи, таблицы), полустатические (очереди, деки), динамические (списки) и файлы.

Статические структуры характеризуются следующими свойствами:

– постоянство структуры в течение всего времени существования;

– смежность элементов и непрерывность области памяти, отведённой сразу для всех элементов структуры;

– простота и постоянство отношений между элементами структуры

Общая классификация структур данных

Важный признак структуры данных – характер упорядоченности её элементов. По этому признаку структуры можно делить на линейно – упорядоченные, или линейные структуры (массивы) и нелинейные структуры (многосвязные списки, древовидные структуры, графовые структуры).

В зависимости от характера взаимного расположения элементов в памяти структуры данных можно разделить на структуры с последовательным размещением в памяти их элементов (векторы, строки, массивы, стеки) и структуры с произвольным связанным распределением в памяти (односвязные, двусвязные, циклически связные, ассоциативные

Массив как структура данных

Массив – это упорядоченная совокупность конечного числа данных одного типа.

Тип компонент массива может быть любым, число компонент массива задаётся при его описании и в дальнейшем не изменяется.

Массив – структура с так называемым случайным доступом, все его компоненты могут выбираться произвольно и являются одинаково доступными.

К любому элементу массива можно обратиться, задав индекс (индексы), который однозначно определяет относительную позицию элемента в массиве.

Тип индексов задаёт тип значений, которые используются для обращения к отдельным элементам массива. Тип индекса может быть одним из упорядоченных типов, т.е. любым скалярным типом, кроме real (например, целые числа и символы).

Одномерный массив

Массив с одним индексом называют одномерным.

Можно сказать, что одномерный массив соответствует понятию вектора. Индекс определяет положение элемента массива относительно его начала.

Общая форма описания переменной типа массив:

```
Var <имя>: Array [<тип-индексов>] of <тип-элементов>;
```

Выбор отдельной компоненты одномерного массива осуществляется указанием идентификатора массива, за которым в квадратных скобках следует индексное выражение.

Индексное выражение должно давать значения, лежащие в диапазоне, определяемом типом индекса.

Например: M[1], M[2],…,M[N].

Работу с массивом удобно организовать в цикле.

Пример создания и использования одномерного массива

Организовать ввод данных в массив и вывод элементов на экран:

```
program f1;
const n=5;
var i:integer;
    mas: array [1..n] of integer;

begin
     for i:=1 to n do
     begin
       write('Введите ',i,'-й элемент массива :');
       readln(mas[i]);
     end;
     writeln('Вот ваш массив: ');
     for i:=1 to n do
       write(mas[i],' ');
     writeln;
end.
```

Нестандартные способы индексации массива

Например, в программе могут присутствовать следующие описания:

```
Var  Cod: Array[Char]   Of   1..100;
   L: Array[Boolean]   Of  Char;
```

В такой программе допустимы следующие обозначения элементов массивов:

```
cod['x'];  L[true];  cod[chr(65)];  L[a>0]
```

В некоторых случаях бывает удобно в качестве индекса использовать перечисляемый тип.

Например, данные о количестве учеников в четырех десятых классах одной школы могут храниться в следующем массиве:

```
Type  Index=(A,B,C,D);
Var Class_10: Array[Index] Of Byte;
```

И если, например, элемент Class_10[A] равен 35, то это означает, что в 10 «А» классе 35 чел. Такое индексирование

Одномерный массив в разделе Type

Можно описать массив как специальный тип данных, а потом использовать созданный тип для объявления переменных данного типа.

Синтаксис такого описания:

```
type <имя типа>=array[<тип-индексов>] of <тип компонент>;
var <идентификатор массива> : <имя типа>;
```

Например:

```
Type Mas1=Array [1..100] Of Integer;
 Mas2=Array [-10..10] Of Char;
Var Num: Mas1;
 Sim: Mas2;
```

Именно такой вариант нужно использовать, если массив выступает в роли возвращаемого параметра процедуры.

Одномерный массив в качестве параметра процедур и функций

Задача: Найти максимальный элемент массива. Заполнение массива, вывод массива и поиск максимального элемента оформить как 2 процедуры и функцию соответственно.

```
Program Mass_Max;
Const n=100;
Type vector = array [1..n] of Integer;
Var v : vector;
    i : integer;
    num: integer;

Procedure Enter (Var vect: vector; n: integer);
Var i: Integer;
  Begin
    For i:=1 to n do
      vect[i]:=Random(10);
  End;

Procedure printArray (Var vect: vector; n: integer);
var i: integer;
begin
  for i:=1 to n do
     write(v[i]:3);
  writeln;
end;
```

```
function arr_max(vect: vector; n: integer): integer;
Var i: Integer;
    max: integer;
begin
  max:=vect[1];
  for i:=2 to n do
    if vect[i]>max then max:=vect[i];
   arr_max:=max;
end;

Begin
  Randomize;
  write('Введите размерность массива: ');
  readln(num);
  Enter (v,num);
  printArray(v,num);
  writeln('Максимальный элемент массива: ',arr_max(v,num));
End.
```

Строковый тип данных

Строка – это последовательность символов. Другими словами, строка – это массив символов.

Каждый символ строки занимает 1 байт памяти (код ASCII). Количество символов в строке называется ее длиной. Длина строки может находиться в диапазоне от 0 до 255.

Строковые величины могут быть константами и переменными.

Строковая константа есть последовательность символов, заключенная в апострофы. Например:

'Язык программирования Pascal'

'Android-смартфон'

'Google – это самый известный поисковик'

Строковый тип данных

Строковая переменная описывается в разделе описания переменных следующим образом:

```
Var <идентификатор>: String[<максимальная длина строки>];
```

Например:

```
Var Name: String[20];
```

Параметр длины можно не указывать в описании. В таком случае подразумевается, что он равен максимальной величине – 255.

Например:

```
Var slovo: String;
```

Строковый тип данных

Строковая переменная занимает в памяти на 1 байт больше, чем указанная в описании длина. Нулевой байт содержит значение текущей длины строки.

Если строковой переменной не присвоено никакого значения, то ее текущая длина равна нулю.

По мере заполнения строки символами ее текущая длина возрастает, но она не должна превышать значения, заданного при объявлении строковой переменной.

Строковый тип данных

К каждому символу строки можно обратиться как к элементу массива, т.е. используя имя переменной строкового типа и номер, который искомый символ занимает в строке.

Например:

```
Var Name: String[20];
…
Name:='Василий';
Writeln(name[3]);
```

На экран будет выведена буква

```
с
```

Работа с данными строкового типа

Сложение строк

Оператор **+** позволяет соединить 2 строки в одну:

```
Var Name: String[20];
    fam: String[30];
    all: String[50];
begin
  Name:='Василий';
  fam:='Иванов';
  all:=Name+' '+fam;
  Writeln(all);
end.
```

Что будет выведено на экран?

На экране появится:

```
Василий Иванов
```

Работа с данными строкового типа

Функция определения длины строкового выражения

```
Length(S: String): Integer;
```

Функция получает в качестве параметра выражение строкового типа (например, строковую переменную) и выдает длину строки.

Пример:

```
Var
 S: String;
Begin
 writeln('Введите строку: ');
 Readln(S);
 Writeln(' " ', S, ' " ');
 Writeln('Длина строки = ', Length(S));
end.
```

Что будет выведено на экран?

```
Введите строку:
Привет
 " Привет "
Длина строки = 6
```

Работа с данными строкового типа

Функция выделения подстроки из строки

Copy(S: **String**; Index: **Integer**; Count: **Integer**): **String**;

Из строки `S` начиная с элемента с номером `Index` копируется кол-во элементов, равное `Count`. В результате функция возвращает новую строку по заданным параметрам.

```
Var S: String;
begin
  S := 'ABCDEF';
  S := Copy(S, 2, 3);
  writeln(s);
end.
```

BCD

Что будет выведено на экран?

Работа с данными строкового типа

Процедура удаления подстроки из строки

Delete(**Var** S: **String**; Index: **Integer**; Count: **Integer**);

Из строки `S` удаляется `Count` элементов, начиная с элемента с номером `Index`. В результате сама исходная строка меняется, поэтому данную процедуру следует применять осторожно.

```
Var s: string;
begin
 s := 'Honest Abe Lincoln';
 Delete(s,8,4);
 Writeln(s);
end.
```

Что будет выведено на экран?

```
Honest Lincoln
```

Работа с данными строкового типа

Процедура вставки в строку подстроки

Insert(S1: **String**; **Var** S2: **String**; Index: **Integer**);

Вставляет строку `S1` в строку `S2` начиная с позиции `Index`. В результате строка `S2` изменяется.

```
Var S: String;
begin
 S := 'Honest Lincoln';
 Insert('Abe ', S, 8);
 writeln(s);
end.
```

Что будет выведено на экран?

```
Honest Abe Lincoln
```

Работа с данными строкового типа

Функция определения позиции подстроки в строке

Pos(Substr: **String**; S: **String**): **Byte**;

Возвращает номер элемента в строке `S`, с которого начинается подстрока `Substr`.

```
Var S: String;
begin
 S := '   123.5';
 while Pos(' ', S) > 0 do
    S[Pos(' ', S)] := '0';
 writeln(s);
end.
```

Что будет выведено на экран?

```
000123.5
```

Пример разбиения строки на слова

```
var s,s1: string;
    i,l: integer;
begin
  write('Введите строку: ');
  readln(s);
  l:=Length(s);
  for i:=1 to l do
  begin
    if s[i]<>' ' then s1:=s1+s[i]
    else
    begin
      if s1<>'' then writeln(s1);
      s1:='';
    end;
  end;
  writeln(s1);
end.
```

Двумерные массивы

Двумерные массивы являются аналогами матриц. Положение элемента в двумерном массиве определяется двумя индексами.

Первый индекс элемента двумерного массива определяет номер строки, а второй – номер столбца, на пересечении которых расположен элемент.

Описание двумерного массива можно производить следующим образом:

Способ 1. Для квадратных матриц:

```
Const
    N = ранг_матрицы;
Type
    matr=array [1..N,1..N] of тип_элементов_матрицы;
Var
    Имя_матрицы : matr;
```

Двумерные массивы

Например:

```
Const
    N = 10;
Type
    matr=array [1..N,1..N] of integer;
Var
    mass : matr;
```

Для неквадратных матриц

```
Const
 N = кол-во строк;
 M = кол-во столбцов;
Type
    matr=array [1..N,1..M] of тип_элементов_матрицы;
Var
    Имя_матрицы : matr;
```

Двумерные массивы

Способ 2. Универсальный.

Var *<имя массива>:* ***array*** *[1.. кол-во_строк, 1..кол-во_столбцов] of тип_элементов_массива;*

Например:

Var mas1: **array** [1..5, 1..10] **of real;**

Пример организации ввода-вывода данных в двумерный массив:

```
Program Array_Full;
Const n=10;
Type mas = array [1..n, 1..n] of Integer;
Var m : mas;
    a,b: integer;

Procedure Enter (Var tabl: mas; x: integer; y: integer);
Var i, j: Integer;
Begin
   For i:= 1 to x do
      For j:= 1 to y do
         tabl[i,j]:=Random(10);
End;
```

Продолжение примера ввода-вывода данных в двумерный массив

```
Procedure MasPrint(tabl: mas; x: integer; y: integer);
Var i, j: Integer;
Begin
   writeln;
   For i:= 1 to x do
   begin
      For j:= 1 to y do
         write(tabl[i,j]:3);
      writeln;
   end;
   writeln;
end;

Begin
  Randomize;
  write('Введите 2 числа для размерности массива: ');
  readln(a,b);
  Enter(m,a,b);
  MasPrint(m,a,b);
End.
```

Матричная алгебра

Задача: найти произведение двух матриц.

Например, $A = \begin{pmatrix} -1 & -2 & -4 \\ -1 & -2 & -4 \\ 1 & 2 & 4 \end{pmatrix}$ и $B = \begin{pmatrix} 1 & 2 & 3 \\ 2 & 4 & 6 \\ 3 & 6 & 9 \end{pmatrix}$

Решение: Вычислим произведение A*B. Согласно правилу умножения матриц, элемент матрицы AB, стоящий в i-ой строке и j-м столбце (c_{ij}) равен сумме произведений элементов i-й строки матрицы A на соответствующие элементы j-го столбца матрицы B.

Так, например,

$$C_{23}=(-1)*3+(-2)*6+(-4)*9=-51$$

Подсчитав таким образом все элементы матрицы AB, находим:

$$AB = \begin{pmatrix} -17 & -34 & -51 \\ -17 & -34 & -51 \\ 17 & 34 & 51 \end{pmatrix}$$

Матричная алгебра

Элементы результирующей матрицы С определяются по формуле:

$$c_{ij} = \sum_{k=1}^{m} a_{ik} b_{kj}; i = 1,2,...n; j = 1,2,...p$$

где n – число строк матрицы А; m – число столбцов матрицы А и число строк матрицы В; p – число столбцов В. В общем случае результирующая матрица С имеет n строк и p столбцов.

```
Program UM;
uses crt;
Const n=10;
Type mas = array [1..n, 1..n] of Integer;
Var  A,b,C : mas;
     K: integer;

Procedure Enter (Var tabl: mas; x,y: integer);
Var i, j: Integer;
Begin
   For i:= 1 to x do
     For j:= 1 to y do
        tabl[i,j]:=Random(2);
End;
```

Продолжение программы перемножения матриц

```
Procedure List(var tabl:mas; x,y: integer);
   Var  i, j: integer;
   Begin
       For i:=1 to x do
       begin
          For j:=1 to y do
             write (tabl[i,j]:4);
          writeln;
       end;
       writeln;
   End;

procedure proizved(var a,b,c:mas; x,y,z: integer);
var i,j,k: integer;
begin
    For I:=1 to x do
       For J:=1 to y do
       Begin
     C[I,J]:=0;
     For K:=1 to z do
         C[I,J]:=C[I,J]+A[I,K]*B[K,J];
       End;
end;

Begin
 enter(a,2,3);
 enter(b,3,3);
 writeln('Матрица A');
 list(a,2,3);
 writeln('Матрица B');
 list(b,3,3);

 proizved(a,b,c,2,3,3);
 writeln('Матрица C');
 list(c,2,3);
End.
```

Тип данных множество

Множество – набор однотипных элементов базового типа, каким-то образом связанных друг с другом.

Базовый тип – это порядковый тип, кроме word, integer, longint.

Число элементов исходного множества не может быть больше 256, а порядковые номера элементов должны находится в пределах от 0 до 255.

```
Type
   <имя множества> = set  of  <тип компонент>;
Var
    <переменная>:<имя множества>;
```

или

```
Var
    <переменная>: set  of  <тип компонент>;
```

Пример объявления множеств и вывода на экран

```
type TDigit = set of 0..9 ;
     TSimv = set of 'a'..'z';
var digit : TDigit;
    simv : TSimv;
  rus_bukvi: set of 'а'..'я';

begin
   rus_bukvi:=['а'..'г'];
   digit:=[1,2,5..7];
   simv:=['i'..'x','a'];
   writeln(rus_bukvi);
   writeln(digit);
   writeln(simv);
end.
```

Допустимые операции с множествами

Операция	Описание
+	объединение
–	разность
*	пересечение
= , <= , >=	проверка эквивалентности двух множеств
< >	проверка неэквивалентности двух множеств
in	логический оператор проверки присутствия компонента в множестве

Примеры решения задач с использованием множеств

Задача: Ввести строку символов, состоящую из латинских букв, цифр и пробелов. Осуществить проверку правильности введенных символов (т.е. чтобы там были только цифры, латинские буквы и пробелы).

```
Var  Str: string;
     L: byte;
     Test: boolean;
Begin
    Writeln('Введите строку');
    Readln (str);
    L:=Length (Str);
    Test:= L>0;
    While Test and (L>0) do //выход из цикла будет осуществлен, если переменная test
    Begin           // станет равна false или L станет равна 0
         Test:=Str[L] in ['0'..'9','A'..'Z','a'..'z',' '];
    {test = true, если str[L] является одним из перечисленных символов}
    {иначе test = false}
         Dec (L)     {аналог L:=L - 1}
    End;
    If Test then  WriteLn ('Правильная строка')
    Else WriteLn ('Неправильная строка');
End.
```

Примеры решения задач с использованием множеств

Другое решение данной задачи:

```
Var Str: string;
    L,i: byte;
    Test: boolean;
Begin
   Writeln ('Введите строку');
   Readln(str);
   L:=Length (Str);
   for i:=1 to L do
   Begin
      test:=Str[i] in ['0'..'9','A'..'Z','a'..'z',' '];
      if test=false then break;
   End;
   If Test then  WriteLn ('Правильная строка')
   Else   WriteLn ('Неправильная строка');
End.
```

Примеры решения задач с использованием множеств

Еще одно решение этой задачи:

```
Var   Str: string;
   L,i,k: byte;
Begin
  Writeln ('Введите строку');
  Readln(str);
  L:=Length (Str);
  for i:=1 to L do
  Begin
    if Str[i] in ['0'..'9','A'..'Z','a'..'z',' '] then k:=k+1;
  End;
  If k=L then  WriteLn ('Правильная строка')
  ElseWriteLn ('Неправильная строка');
End.
```

Примеры решения задач с использованием множеств

Задача: *Заполнить множество А путем ввода n значений:*

```
var   A: set of 0..200;
      j,x,n: byte;
begin
     write('Введите кол-во эл-ов в мн-ве: ');
     readln(n);
     A:=[];       // задаем пустое мн-во
     for j := 1 to n do
     begin
          repeat //цикл
            x:=random(10); //генерируем число
          until not (x in A); //проверяем его в мн-ве
          A:=A+[x] // добавляем в мн-во введенный элемент
     end;
     write('Вот ваше мн-во: ');
     for x := 0 to 200 do
       if x in A then write(x:3); //если x в мн-ве, то напечатать его
     writeln;
     writeln(A); //можно просто вывести на экран мн-во
end.
```

Тип данных запись

Запись представляет собой наиболее общий и гибкий структурированный тип данных, так как она может быть образована из неоднотипных компонентов и в ней явным образом выражена связь между элементами данных, характеризующими реальный объект.

Запись – это структурированный тип данных, состоящий из фиксированного числа компонентов одного или нескольких типов.

Определение типа записи начинается идентификатором record и заканчивается зарезервированным словом end.

Между ними располагается список компонентов, называемых полями, с указанием идентификаторов полей и типа каждого поля.

Тип данных запись

Формат:

```
type
   <имя типа> = record
      <идентификатор поля>:<тип компонента>;
      <идентификатор поля>:<тип компонента>;
      . . .
      <идентификатор поля>:<тип компонента>;
   end;

var
<идентификатор, ...> : <имя типа>;
```

Тип данных запись

Пример:

```
type Car = record
    gosNumber : string; {гос.номер}
    Marka : string[20]; {Марка автомобиля}
    FIO : string[40]; {Фамилия, инициалы
          владельца}
    Address : string[60] {Адрес владельца}
    end;
var
  m, V : Car;
```

После объявления в программе переменной типа «запись» к каждому ее полю можно обратиться, указав сначала идентификатор переменной-записи, а затем через точку – имя поля.

Пример использования записи

Создать список автомобилей и в нем найти все с маркой машины, введенной пользователем.

```
type Car = record
    gosNumber : string; {гос.номер}
    Marka : string[20]; {Марка автомобиля}
    FIO : string[40]; {Фамилия, инициалы владельца}
    end;

    cars=array [1..10] of car;
var
    autos : cars;
    i,n : integer;
    name : string;
    flag : boolean;

procedure enterCars (var m: cars; n: integer);
var i: integer;
begin
    for i:=1 to n do
    begin
     writeln('Введите ',i,'-ю запись: ');
     write('Марка машины:':20);
     readln(m[i].marka);
     write('Номер машины:':20);
     readln(m[i].gosNumber);
     write('ФИО владельца:':20);
     readln(m[i].fio);
    end;
end;
```

```pascal
procedure printHead;
begin
   writeln('Вот ваши записи:');
   writeln('№ ':4,'|','Марка  ':10,'|','Владелец   ':20,'|','Гос.номер
   ':15);
end;

begin
    write('Введите кол-во записей: ');
    readln(n);
    enterCars(autos, n);
    printHead;
    for i:=1 to n do
    begin
     writeln(i:4,'|',autos[i].Marka:10,'|',autos[i].FIO:20,'|',
                         autos[i].gosNumber:15);
    end;
    write('Введите марку машины: ');
    readln(name);
    printHead;
    for i:=1 to n do
    begin
     if (name=autos[i].Marka) then
     begin
        writeln(i:4,'|',autos[i].Marka:10,'|',autos[i].FIO:20,'|',
                         autos[i].gosNumber:15);
        flag:=true;
     end;
    end;
    if flag=false then writeln('Не найдено.');
end.
```

Тип данных файл

С одной стороны, файл – это область памяти на внешнем носителе, в котором хранится некоторая информация.

Файл в таком понимании называют физическим файлом, т. е. существующим физически на некотором материальном носителе информации.

С другой стороны, файл – это одна из структур данных, используемых в программировании.

Файл в таком понимании называют логическим файлом, т.е. существующим в нашем логическом представлении при написании программы.

Тип данных файл

Структура физического файла представляет собой простую последовательность байт памяти носителя информации.

Структура логического файла – это способ восприятия файла в программе.

Любой файл имеет следующие характеристики-требования:

- у него есть имя (набор из восьми, допустимых для имени файла, символов плюс расширение, указываемое после точки в имени файла, состоящее из трех символов);
- он должен содержать данные одного типа (любой тип языка Pascal, кроме типа Файл, то есть не существует типа «Файл файлов»);

Размер создаваемого файла никак не регламентируется при создании файла и

Тип данных файл

Работа с файлами в Паскале осуществляется следующим образом:

- сначала объявляется переменная файлового типа, с указанием свойств переменной (то есть типом содержимого);
- затем данная файловая переменная связывается («ассигнуется») с именованным дисковым пространством (то есть непосредственно с конкретным файлом, содержащим или который будет содержать данные того же типа, что и связываемая переменная-файл).

Тип данных файл

Переменная файлового типа может быть объявлена одной из следующих строк:

```
<имя> = file of <тип>;
<имя> = text; {текстовый файл}
<имя> = file; {файл без типа}
```

где <имя> – имя переменной-файла;

file of – зарезервированные слова (файл из);

text – имя стандартного типа текстовых файлов;

<тип> – имя любого стандартного типа языка Pascal, кроме типа файл.

Тип данных файл

Например файл, содержащий список учеников и их возраст:

```
type
     pupil = record
          surname : string[30];
          name : string[30];
          age : word
     end;
Var
     journal : file of pupil;
```

Тип данных файл

Текстовые файлы – это файлы, содержащие символы, разделенные на строки, а в конце каждой строки стоит признак конца строки.

Текстовые файлы не имеют прямого доступа.

При чтении и записи числа преобразуются автоматически.

К ним применима процедура **Append**(<имя переменной текстового файла>). Она открывает текущий файл, с которым связана данная переменная, текущий указатель помещает в конец для добавления новой информации.

Нетипизированные файлы предназначены для низкоуровневой работы с файлами.

С их помощью можно обратиться к файлу любого типа и логической структуры.

За одно обращение считывается/записывается число байт, приблизительно равное величине буфера ввода/вывода. В качестве буфера может выступать любая

Процедуры и функции обработки файловых переменных

Assign (<*имя файловой переменной*>, ’<*путь и имя файла на диске*>’) – связь переменной файлового типа с конкретным внешним файлом.

Reset (f) – процедура открытия существующего файла и подготовка к чтению файла, связанного с файловой переменной f. Указатель текущей позиции файла устанавливается в его начало.

Rewrite (f) – процедура создания нового физического файла и подготовка к записи файла, связанного с файловой переменной f**.** Если такой файл существует, то он удаляется, и на этом месте создается новый пустой файл. Указатель текущей позиции файла устанавливается в его начало.

Процедуры и функции обработки файловых переменных

Readln (f) – пропуск строки файла до начала следующей.

Writeln (f) – запись признака конца строки и переход на следующую.

Read (f, x) – процедура чтения компоненты файла. Данные выводятся из файла.

Write (f, x) – процедура записи значения переменной в файл, который хранится на диске. Указатель перемещается на следующий элемент. Если указатель текущей позиции файла находится за последним элементом, т.е. в конце файла, то файл расширяется.

Eof (f) – признак конца файла – логическая функция для определения, достигнут ли конец файла.

Close (f) – процедура закрытия файла (**!!! именно в этот момент происходит реальная запись в файл**).

Примеры решений задач с использованием файлов

1. *Прочитать из текстового файла А все записанные в него целые числа (их можно ввести в него через Блокнот и сохранить файл), преобразовать их в вещественные и вывести в текстовый файл В по 4 числа в строку.*

```
Var F1,F2: text; // объявляем 2 переменные типа текстовых файлов
    X: real;
    s: string;
    I:integer;
Begin
  Assign (F1,'A.txt'); //связываем переменную F1 с файлом A.txt
          //(находится там же, где и программа)
  Reset(F1);      // открываем его для чтения
          //(т.е. он уже должен быть создан ранее)
  Assign (F2,'B.txt'); //связываем F2 с файлом B.txt
          //(будет находиться там же, где и программа)
  Rewrite (F2);   // открываем его для записи
  writeln('содержимое файла A.txt:');
```

```
Repeat  // цикл
  For I:=1 to 4 do  // цикл от 1 до 4,
   //т.к. нужно в файле В формировать по 4 числа в строке
   If not seekeof(F1) then  // если еще не конец файла,
   Begin          //связанного с F1, то делаем
      Read (F1,x); //считываем из файла, связанного с F1,
        //1 компонент и помещаем его в переменную х
      write(x:-5:0);  //выводим на экран, что считали из
//файла в текущий момент, -5 означает, что в поле из 5
//позиций с выравниванием по левому краю
      Write (F2,x:-5:1) // выводим текущее считанное
             //значение в файл, связанный с F2
   End;
   Writeln(F2);  // делаем в файле переход на новую строку
   //(чтобы следующее значение выводилось с новой строки)
Until seekeof (F1);//будет повторяться,
             //пока файл не закончится
writeln;
```

```
  Close (F1);  // закрываем файл, связанный с F1
  Close (F2); // закрываем файл, связанный с F2
  Reset(F2); //открываем файл, связанный с F2,
      //для чтения
  writeln('содержимое файла B.txt:');
  Repeat //цикл
    Readln(F2,s); // считываем по целой строке из B
    writeln(s); //и записываем в строковую переменную s
  Until seekeof (F2);//пока не конец файла, связанного
            //с F2
  Close (F2);
End.
```

Пример работы программы:

содержимое файла A.txt:

1 2 3 4 5

содержимое файла B.txt:

1.00 2.00 3.00 4.00

5.00

2. Работа с числовыми файлами. Записать в файл числа (кол-во вводит пользователь, сами числа задаются генератором случайных чисел). Записать в другой файл числа, кратные 5 или 4.

```
var f1,f2: file of integer; //файловые переменные типа integer (в
файле будут только целые числа)
    i,n,x: integer;
begin
  assign(f1,'a.dat'); //связываем файловую переменную f1 с файлом a.dat
  rewrite(f1); //открываем его на запись или перезапись
       //(если он уже существует)
  assign(f2,'b.dat'); // связываем файловую переменную f2 с файлом b.dat
  rewrite(f2); // открываем его на запись или перезапись
       //(если он уже существует)
  write('Введите кол-во чисел: ');
  readln(n);
  for i:=1 to n do  // будем в цикле записывать данные в файл
  begin
    x:=random(30);
    write(f1,x);   // текущее значение x записывается в файл
  end;
  close(f1); //закрываем файл, связанный с f1
    //(это нужно, чтобы все записанные в него данные сохранились)
```

```
  reset(f1);  // открываем файл для чтения
  writeln('Вот числа из первого файла:');
  for i:=1 to n do
  begin
    read(f1,x);  // считываем 1 значение из файла,
          //связанного с f1, и записываем его в x
    write(x:5);  // выводим x на экран
    if (x mod 5 =0) or (x mod 4 = 0) then write(f2,x);
  end;
  close(f1); // закрываем файл, связанный с f1
  close(f2); // закрываем файл, связанный с f2
  reset(f2); // открываем файл, связанный с f2, для чтения
  writeln; writeln('Вот числа из второго файла:');
  while (not eof(f2)) do  // т.к. мы не знаем, сколько было
//записано элементов в файл, связанный с f2, то используем
//цикл с условием, который будет работать, пока не будет
//считан признак конца файла (while (not eof(f2)) do)
  begin
    read(f2,x);  // считываем 1 элемент из файла, связанного с
          //f2, и записываем его в x
    write(x:5);  // выводим x на экран
  end;
  close(f2); // закрываем файл, связанный с f2
end.
```

Работа с графикой в PascalABC.NET

Для работы с графикой в среде программирования PascalABC.NET нужно подключить модуль GraphABC.

Процедуры модуля GraphABC:

```
procedure SetPixel(x,y,color: integer);
```

Закрашивает один пиксел с координатами `(x,y)` цветом `color`.

```
procedure MoveTo(x,y: integer);
```

Передвигает невидимое перо к точке с координатами `(x,y)`; эта функция работает в паре с функцией `LineTo(x,y)`.

```
procedure LineTo(x,y: integer);
```

Рисует отрезок от текущего положения пера до точки `(x,y)`; координаты пера при этом также становятся равными `(x,y)`.

```
procedure Line(x1,y1,x2,y2: integer);
```

Рисует отрезок с началом в точке `(x1,y1)` и концом в точке `(x2,y2)`.

```
procedure Circle(x,y,r: integer);
```

Рисует окружность с центром в точке `(x,y)` и радиусом `r`.

```
procedure Ellipse(x1,y1,x2,y2: integer);
```

Рисует эллипс, заданный своим описанным прямоугольником с координатами противоположных вершин `(x1,y1)` и `(x2,y2)`.

```
procedure Rectangle(x1,y1,x2,y2: integer);
```

Рисует прямоугольник, заданный координатами противоположных вершин `(x1,y1)` и `(x2,y2)`.

```
procedure RoundRect(x1,y1,x2,y2,w,h: integer);
```

Рисует прямоугольник со скругленными краями; `(x1,y1)` и `(x2,y2)` задают пару противоположных вершин, а `w` и `h` – ширину и высоту эллипса, используемого для скругления

```
procedure Arc(x,y,r,a1,a2: integer);
```

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).

```
procedure Pie(x,y,r,a1,a2: integer);
```

Рисует сектор окружности, ограниченный дугой (параметры процедуры имеют тот же смысл, что и в процедуре `Arc`).

```
procedure TextOut(x,y: integer; s: string);
```

Выводит строку `s` в позицию `(x,y)` (точка `(x,y)` задает верхний левый угол прямоугольника, который будет содержать текст из строки `s`).

```
procedure FloodFill(x,y,color: integer);
```

Заливает область одного цвета цветом `color`, начиная с точки `(x,y)`.

```
procedure FillRect(x1,y1,x2,y2: integer);
```

Заливает прямоугольник, заданный координатами противоположных вершин `(x1,y1)` и `(x2,y2)`, цветом текущей кисти.

```
procedure Polygon(points: array of Point);
```

Рисует заполненный многоугольник, координаты вершин которого заданы в массиве `points`.

```
procedure Polyline(points: array of Point);
```

Рисует ломаную по точкам, координаты которых заданы в массиве `points`. **uses** graphABC;

```
uses
  graphABC;

var
  P: array of Point; //Массив точек

begin
  SetWindowSize(600, 450); //Устанавливаем размер окна
  SetPenColor(clGreen); //Цвет линий
  SetPenWidth(3); //Толщина линий
  SetLength(P, 3); //Устанавливаем размер массива points
  p[0].x := 300; p[0].y := 100; //Координаты первой вершины
  p[1].x := 500; p[1].y := 200; //Координаты второй вершины
  p[2].x := 300; p[2].y := 400; //Координаты третьей вершины
  Polygon(P) //Рисуем ломаную по точкам из массива P
end.
```

```
function PenX: integer;
function PenY: integer;
```

Возвращают текущие координаты пера.

```
procedure SetPenColor(color: integer);
```

Устанавливает цвет пера, задаваемый параметром `color`.

```
function PenColor: integer;
```

Возвращает текущий цвет пера.

```
procedure SetPenWidth(w: integer);
```

Устанавливает ширину пера, равную `w` пикселам.

```
function PenWidth: integer;
```

Возвращает текущую ширину пера.

```
procedure SetPenStyle(ps: integer);
```

Устанавливает стиль пера, задаваемый параметром `ps`.

```
function PenStyle: integer;
```

Возвращает текущий стиль пера.

Стили пера задаются следующими именованными константами:

```
psSolid     psClear       psDash
psDot       psDashDot     psDashDotDot
```

```
procedure SetBrushColor(color: integer);
```

Устанавливает цвет кисти, задаваемый параметром `color`.

```
function BrushColor: integer;
```

Возвращает текущий цвет кисти.

```
procedure SetBrushStyle(bs: integer);
```

Устанавливает стиль кисти, задаваемый параметром `bs`.

```
function BrushStyle: integer;
```

Возвращает текущий стиль кисти.

SetBrushHatch(bhLargeConfetti);

Стили кисти задаются следующими именованными константами:

```
bhSolid  bhCross   bhClear
bhDiagCross  bhHorizontal  bhBDiagonal
bhVertical      bhFDiagonal
```

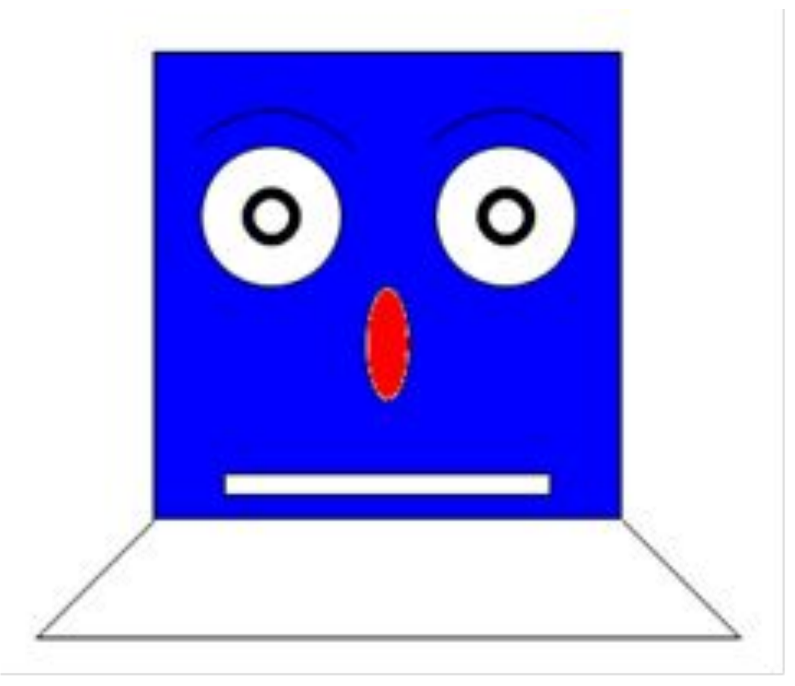
```
procedure ClearWindow;
```

Очищает графическое окно белым цветом.

```
procedure ClearWindow(c: ColorType);
```

Очищает графическое окно цветом `c`.

Пример. Составим программу, рисующую голову робота. Рисунок содержит два прямоугольника, 4 окружности, две дуги, эллипс, три прямые линии. Заранее определяются все координаты и размеры элементов рисунка.



```
uses graphABC;
begin
 Rectangle(100,100,300,300);  {голова}
 floodfill(105,120,clBlue);  // закрашиваем голову синим
 Circle(150,170,30);       {левый глаз}
 Circle (250,170,30);   {правый глаз}
 Arc (150,170,45,135,40);  {левая бровь}
 Arc (250,170,45,135,40);  {правая бровь}
 Ellipse(190,200,210,250); {нос}
 floodfill(195,220,clRed); // закрашиваем нос красным
 Rectangle (130,280,270,290); {рот}
 MoveTo (100,300);  {переход вниз влево}
 LineTo(50,350); {три}
 LineTo(350,350);   {линии}
 LineTo(300,300);   {шеи}
 SetPenWidth(5); // устанавливаем толщину линии в 5 пикселей
 Circle(150,170,10);       {левый зрачок}
 Circle(250,170,10);       {правый зрачок}
end.
```

Графика и движение

```
uses graphabc;
var i: integer;

begin
  for i:=0 to 300 do
  begin
    SetPenColor(clBlack);
    circle(60,60+i,50);
    Sleep(50);
    ClearWindow();
  end;
end.
```

Создание модулей в языке Pascal

Структура модуля

Структура модуля аналогична структуре программы, однако есть несколько существенных различий.

```
unit <идентификатор>; //должен совпадать с именем файла
interface
   uses <список модулей>;      { Необязательный }
   { глобальные описания }
implementation
   uses <список_модулей>;    { Необязательный }
   { локальные описания }
   { реализация процедур и функций }
begin
   { код инициализации }
end.
```

Создание модулей в языке Pascal

Интерфейсная секция

Интерфейсная часть – «общедоступная» часть в модуле – начинается зарезервированным словом interface, следует сразу после заголовка модуля и заканчивается перед зарезервированным словом implementation.

Интерфейс определяет, что является «видимым» (доступным) для любой программы (или модуля), использующей данный модуль.

В интерфейсной части (секции) модуля можно определять константы, типы данных, переменные, процедуры и функции.

Процедуры и функции, видимые для любой программы, использующей данный модуль, описываются в секции интерфейса, однако их действительные тела – реализации – находятся в секции реализации.

В интерфейсной части перечисляются все заголовки процедур и функций.

Секция реализации

Содержит программную логику процедур и функций.

Секция реализации – «приватная» часть – начинается зарезервированным словом implementation.

Все, что описано в секции интерфейса, является видимым в секции реализации: константы, типы, переменные, процедуры и функции.

Кроме того, в секции реализации могут быть свои дополнительные описания, которые не являются видимыми для программ, использующих этот модуль.

Программа не знает об их существовании и не может ссылаться на них или обращаться к ним.

Однако, эти скрытые элементы могут использоваться «видимыми» процедурами и функциями, то есть теми подпрограммами, чьи заголовки указаны в секции интерфейса.

Обычные процедуры и функции, описанные в интерфейсной секции, должны повторно указываться в секции

Создание модулей в языке Pascal

Пример. *Напишем модуль с двумя функциями: первая меняет значения двух переменных; вторая находит максимум из двух переменных.*

```
unit IntLib;
interface  {объявляем процедуры и функции, которые будут реализованы позже}
   procedure ISwap(var I,J : integer);
{меняет значения переменных: I на значение J, J на значение I}
   function IMax(I,J : integer) : integer;
{находит максимум из двух переменных}
```

```
implementation  {расписываем процедуры и
функции, объявленные ранее}
  procedure ISwap;
  var   Temp : integer;
  begin
      Temp := I; I := J; J := Temp
  end;  { конец процедуры ISwap }

  function IMax: integer;
  begin
    if I > J then IMax := I else IMax := J
  end;  { конец функции IMax }
end.  { конец модуля IntLib }
// ctrl+F9 (создание бинарного файла модуля)
```

Программа, использующая созданный модуль

```
program IntTest;
uses IntLib;
var
  A,B  : integer;
begin
  Write('Введите два целочисленных значения:');
  Readln(A,B);
  ISwap(A,B);
  Writeln('A = ',A,' B = ',B);
  Writeln('Максимальное значение равно ',
           IMax(A,B));
end.
```

Динамические структуры данных

Для работы с динамическими структурами данных используются указатели.

Указатели представляют собой специальный тип данных.

Они принимают значения, равные адресам размещения в оперативной памяти соответствующих динамических переменных.

Пример

```
type
    pint=^integer;

var x,y:integer;
    px: pint;

begin
   x:=1;
   new(px);
   px:=@x;
   px^:=5;
   writeln(x);

end.
```

Динамические структуры данных

Списком называется структура данных, каждый элемент которой посредством указателя связывается со следующим элементом.

На самый первый элемент (голову списка) имеется отдельный указатель.

Из определения следует, что каждый элемент списка содержит поле данных (оно может иметь сложную структуру) и поле ссылки на следующий элемент.

Поле ссылки последнего элемента должно содержать пустой указатель (nil).

Число элементов связанного списка может расти или уменьшаться в зависимости от того, сколько данных мы хотим хранить в нем.

Список

Чтобы добавить новый элемент в список, необходимо:

1. Получить память для него.
2. Поместить туда информацию.
3. Добавить элемент в конец списка (или начало).

Элемент списка состоит из разнотипных частей (хранимая информация и указатель), и его естественно представить записью.

Перед описанием самой записи описывают указатель на нее:

```
Type { описание списка из целых чисел }
  PList = ^TList;
  TList = record
     Data: Integer;
     Next : PList;
  end;
```

Создание списка

```
Type { описание списка из целых чисел }
  PList = ^TList;
  TList = record
     Data : Integer;
     Next : PList;
  end;

var
   x,list,head : PList;

begin
     new(head);
     head^.data:=3;
     head^.next:=nil;
     x:=head;
     New(x^.Next);
     x := x^.Next;
     x^.Data := 5; { значение этого элемента }
     x^.Next := nil; {следующего значения нет }
     New(x^.Next); { выделим области памяти для следующего элемента }
     x := x^.Next ; { переход к следующему (3-му) элементу списка }
     x^.Data := 1; { значение этого элемента }
     x^.Next := nil; {следующего значения нет }
```

Просмотр списка

```
List := Head;
    While List <> nil do
    begin
     WriteLn(List^.Data);
     List := List^.Next;
{ переход к следующему элементу;
аналог для массива  i:=i+1 }
    end;

end.
```

Удаление элемента из списка

При удалении элемента из списка необходимо различать три случая:

1. Удаление элемента из начала списка.
2. Удаление элемента из середины списка.
3. Удаление из конца списка.

Удаление элемента из начала списка

```
{ запомним адрес первого элемента списка }
  List := Head;
{теперь Head указывает на второй элемент списка }
  Head := Head^.Next;
 { освободим память, занятую переменной List }
  Dispose(List);
```

Удаление элемента из середины списка

Для этого нужно знать адреса удаляемого элемента и элемента, находящегося в списке перед ним.

```
List := Head;
While (List<>nil) and (List^.Data<>Digit) do
begin
    x := List;
    List := List^.Next;
end;
x^.Next := List^.Next;
Dispose(List);
```

Удаление из конца списка

Оно производится, когда указатель x показывает на предпоследний элемент списка, а List – на последний.

```
List := Head;
 x:= Head;
 While List^.Next<>nil do
 begin
     x := List;
     List := List^.Next;
 end;
 x^.Next := nil;
 Dispose(List);
```

x

List

x^.Next

nil

Рекурсия

Рекурсией называется способ задания функции, при котором значение определяемой функции для произвольных значений аргументов выражается известным образом через значения этой функции для других значений аргументов.

Различают два вида рекурсии подпрограмм:

- *прямая* или *явная* рекурсия – характеризуется существованием в теле подпрограммы оператора обращения к самой себе;
- *косвенная* или *неявная* рекурсия – образуется при наличии цепочки вызовов других подпрограмм, которые в конечном итоге приведут к вызову исходной.

Косвенная или неявная рекурсия

```
function iMax(x,y: integer):integer;
begin
  if x>y then iMax:=x else iMax:=y;
end;

var
  a,b,c,d: integer;

begin
  write('Введите 4 числа: ');
  readln(a,b,c,d);
  writeln('Максимальное: ',
iMax( iMax(a,b), iMax(c,d) ) );
end.
```

Явная рекурсия

```
function fakt(n:integer):integer;
begin
 if n=0 then fakt:=1 else fakt:=n*fakt(n-1);
end;

var i: integer;

begin
   write('Введите число ->');
   readln(i);
   writeln(fakt(i));
end.
```

Многие математические функции можно выразить рекурсивно.

Например, для неотрицательных значений n имеем:

$$\text{степенная функция} \quad x^n = \begin{cases} 1, & \text{если} \quad n = 0; \\ x \cdot x^{n-1}, & \text{если} \quad n > 0; \end{cases}$$

$$\text{функция факториал} \quad n! = \begin{cases} 1, & \text{если} \quad n = 0; \\ n \cdot (n-1)!, & \text{если} \quad n > 0; \end{cases}$$

Данное определение факториала состоит из двух утверждений.

Первое утверждение носит название базисного.

Базисное утверждение нерекурсивно.

Второе утверждение носит название рекурсивного или индуктивного.

Оно строится так, чтобы полученная в результате повторных применений цепочка определений сходилась к базисному утверждению.

Рекурсивное возведение числа в степень

```
function step(a,n:integer):integer;
begin
 if n=0 then step:=1
   else step:=a*step(a,n-1);
end;

var x,y: integer;

begin
     write('Введите число и степень-> ');
     readln(x,y);
     writeln(step(x,y));
end.
```

Рекурсивное вычисление суммы элементов одномерного массива

```
Type LinMas = Array[1..100] Of Integer;
Var A : LinMas;
    I, N : Byte;

Function Summa(N : Byte; A: LinMas) : Integer;
Begin
    If N = 0 Then Summa := 0 Else Summa := A[N] + Summa(N - 1, A)
End;

Begin
    Write('Количество элементов массива? ');
    ReadLn(N);
    Randomize;
    For I := 1 To N Do
    Begin
       A[I] := Random(5)+1;
       Write(A[I] : 4)
    End;
    WriteLn; WriteLn('Сумма: ', Summa(N, A))
End.
```

Рекурсивное определение строки-палиндрома

```
Function Pal(S: String) : Boolean;
Begin
     If Length(S)<=1  Then Pal:=True
     Else Pal:= (S[1]=S[Length(S)])
         and Pal(Copy(S, 2, Length(S) - 2));
End;

Var S : String;

Begin
    Write('Введите строку: '); ReadLn(S);
    If Pal(S) Then
      WriteLn('Строка является палиндромом')
    Else WriteLn('Строка не является палиндромом')
End.
```

Рекурсия и графика

```
uses graphabc;
var x,y,r,d,m:integer;
procedure ris(x,y,r:integer);
var i:integer;
begin
    if r<10 then exit;
   circle(x,y,r);
   sleep(300);
   ris(x+r,y,r div 2);
   ris(x-r,y,r div 2);
end;
begin
   x:=320;
   y:=240;
   r:=120;
   ris(x,y,r);
end.
```

Рекурсия и фракталы

```
uses graphabc;
var
   x,y,r:integer;
procedure ris(x,y,r:integer);
var
   x1,y1,t:integer;
begin
   if r<=1 then begin putpixel(x,y,clRed);exit end;
   for t:=0 to 6 do
   begin
      x1:=x+trunc(r*cos(t*pi/3));
      y1:=y+trunc(r*sin(t*pi/3));
      line(x,y,x1,y1);
      ris(x1,y1,r*2 div 5);
      sleep(5);
   end;
end;
begin
   x:=320;
   y:=240;
   r:=80;
   ris(x,y,r);
end.
```

Кривая дракона

```
uses graphabc;
var k:integer;
procedure ris(x1,y1,x2,y2,k:integer);
var xn,yn:integer;
begin
   if k>0 then
   begin
      xn:=(x1+x2) div 2 +(y2-y1) div 2;
      yn:=(y1+y2) div 2 -(x2-x1) div 2;
      ris(x1,y1,xn,yn,k-1);
      ris(x2,y2,xn,yn,k-1);
   end
   else begin line(x1,y1,x2,y2); end;
end;
begin
   readln ( k );{задаем порядок кривой}
   ris(200,300,500,300,k);
end.
```

Сортировка массива

Под сортировкой массивов понимают процесс перестановки элементов массива в определённом порядке.

Цель сортировки – облегчить последующий поиск элементов в отсортированном массиве.

Методы сортировки важны при обработке данных, с ними связаны многие фундаментальные приёмы построения алгоритмов.

Сортировки могут быть выполнены с использованием различных алгоритмов: как простых, так и усложнённых (но более эффективных).

Сортировка массива

Основное требование к методам сортировки: экономное использование памяти и быстродействие.

Хорошие алгоритмы сортировки требуют порядка $n \bullet \log_2 n$ сравнений.

Простые методы сортировки можно разбить на три основных класса в зависимости от лежащего в их основе приёма:

1) сортировка выбором;
2) сортировка обменом;
3) сортировка включением.

Простые методы сортировки требуют порядка $n*n$ сравнений элементов (ключей).

Сортировка обменом (метод пузырька)

Сортировка обменом предусматривает систематический обмен значениями (местами) тех пар, в которых нарушается упорядоченность, до тех пор, пока таких пар не останется.

Проходы по массиву повторяются N-1 раз или до тех пор, пока на очередном проходе не окажется, что обмены больше не нужны, что означает – массив отсортирован.

При каждом проходе алгоритма по внутреннему циклу, очередной наибольший элемент массива ставится на своё место в конце массива рядом с предыдущим «наибольшим элементом», а наименьший элемент перемещается на одну позицию к началу массива («всплывает» до нужной позиции как пузырёк в воде, отсюда и название алгоритма).

Реализация сортировки методом пузырька

```
procedure bubbleSort(var data: vector; n: integer);
var i,j, tmp: integer;
begin
    for i:=1 to n-1 do
    begin
        for j:=1 to n-i do
        begin
            if (data[j] > data[j+1]) then
            begin
                tmp:=data[j];
                data[j]:= data[j+1];
                data[j+1]:=tmp;
            end;
        end;
    end;
end;
```

5 2 4 6 1 3

Реализация сортировки методом пузырька

```
procedure bubbleSort(var data: vector; n: integer);
var i,j, tmp: integer;
begin
    for i:=1 to n-1 do
    begin
        for j:=1 to n-i do
        begin
            if (data[j] > data[j+1]) then
            begin
                tmp:=data[j];
                data[j]:= data[j+1];
                data[j+1]:=tmp;
            end;
        end;
    end;
end;
```

Сортировка посредством простого выбора

Сортировка основана на идее многократного выбора (находится сначала наибольший элемент, затем второй по величине и т. д.) и сводится к следующему:

1) найти элемент с наибольшим значением;

2) поменять значениями найденный элемент и последний;

3) уменьшить на единицу количество просматриваемых элементов;

4) если <количество элементов для следующего просмотра больше единицы> то <повторить пункты, начиная с 1-го>.

Реализация сортировки выбором

```
type vector=array[1..100] of integer;

procedure sort_choice (var a: vector; n: integer) ;
var X : integer;
i , j , k : integer;
begin
   for i:=n downto 2 do
   begin
      k:=1;
      for j:=2 to i do
         if a[j] > a[k] then k:=j ;
      X:=a[k];
      a[k]:=a[i];
      a[i]:=x
   end
end;
```

Сортировка посредством простого выбора

Второй вариант сортировки выбором:

- находим номер минимального значения в текущем списке;
- производим обмен этого значения со значением первой неотсортированной позиции (обмен не нужен, если минимальный элемент уже находится на данной позиции);
- теперь сортируем хвост списка, исключив из рассмотрения уже отсортированные элементы.

Реализация второго варианта сортировки выбором

```
type vector=array[1..100] of integer;

procedure sort_choice2(var a: vector;n : integer) ;
var i,j,x,min:integer;
begin
    for i:=1 to n-1 do
    begin
            min:=i;
            for j:=i+1 to n do
            begin
                if (a[j] < a[min]) then min:=j;
            end;
            x:=a[i];
            a[i]:=a[min];
            a[min]:=x;
        end;
end;
```

	8
	5
	2
	6
	9
	3
	1
	4
	0
	7

Сравнение разных видов сортировки

	Insertion	Selection	Bubble	Shell	Merge	Heap	Quick	Quick3
Random								
Nearly Sorted								
Reversed								
Few Unique								

Методы поиска данных в массиве

Применяются для поиска нужного элемента в массиве или выяснения факта его отсутствия.

1. Линейный поиск

2. Бинарный поиск

Методы поиска данных в массиве

Линейный (последовательный) поиск

Самый простой из известных.

Множество элементов просматривается последовательно в некотором порядке, гарантирующем, что будут просмотрены все элементы массива (например, слева направо).

Если в ходе просмотра множества будет найден искомый элемент, просмотр прекращается с положительным результатом; если же будет просмотрено все множество, а элемент не будет найден, алгоритм должен выдать отрицательный результат.

Именно так поступает человек, когда ищет что-то в неупорядоченном множестве. Например, при поиске нужной визитки в некотором неупорядоченном множестве визиток человек просто перебирает все визитки в поисках нужной.

```
i := 1;
while (i<=n) and (A[i]<>Key) do Inc(i);
if A[i]=Key then <элемент найден> else <элемент не найден>;
```

Методы поиска данных в массиве

Линейный (последовательный) поиск

Удалению элемента всегда предшествует успешный поиск.

При удалении нам придется сначала найти положение удаляемого элемента, затем все элементы, следующие за ним, сдвинуть на одну позицию к началу массива и уменьшить n. То есть в любом случае надо пробежать весь массив от начала до конца:

```
i := 1;
while (i<=n)and(A[i]<>Key) do Inc(i); {поиск}
if A[i]=Key then
begin
 while i<n do
 begin
   A[i] := A[i+1]; {сдвигаем}
   Inc(i);
 end;
 Dec(n);
end;
```

Методы поиска данных в массиве

Бинарный поиск

Этот метод поиска предполагает, что массив отсортирован.

Суть метода: Областью поиска (l, r) назовем часть массива с индексами от l до r, в которой предположительно находится искомый элемент.

Сначала областью поиска будет часть массива (l, r), где l=1, а r=n, то есть вся заполненная элементами множества часть массива.

Теперь найдем индекс среднего элемента m=(l+r) div 2. Если Key>A[m], то можно утверждать (поскольку массив отсортирован), что если Key есть в массиве, то он находится в одном из элементов с индексами от m+l до r, следовательно, можно присвоить l=m+1, сократив область поиска. В противном случае можно положить r=m. На этом заканчивается первый шаг метода. Остальные шаги аналогичны.

На каждом шаге метода область поиска будет сокращаться вдвое. Как только l станет равно r, то есть область поиска сократится до одного элемента, можно будет проверить этот элемент на равенство искомому и сделать вывод о результате

Методы поиска данных в массиве

Бинарный поиск

Поиск по ключу 5 в массиве:

1 1 1 2 2 3 3 3 4 4 4 4 5 5 6 6 7 7 8 8

Бинарный поиск

1) left = 0, right = 19, mid = 10

1 1 1 2 2 3 3 3 4 4 4 4 5 5 6 6 7 7 8 8

2) left = 11, right = 19, mid = 15

1 1 1 2 2 3 3 3 4 4 4 4 5 5 6 6 7 7 8 8

3) left = 11, right = 14, mid = 12

1 1 1 2 2 3 3 3 4 4 4 4 5 5 6 6 7 7 8 8

4) left = 13, right = 14

1 1 1 2 2 3 3 3 4 4 4 4 5 5 6 6 7 7 8 8

Методы поиска данных в массиве

Бинарный поиск

```
L := 1; R := N;
while L < R do
begin
  M := (L + R) div 2;
  if a[M] < x then L := M + 1 else R := M
end;
if a[R] = x then
  write(' Номер i=', R, ' для элемента a[i]=x')
else
  write('x не найден');
```

Поиск по ключу 5 в массиве:

1	1	1	2	2	3	3	3	4	4	4	4	5	5	6	6	7	7	8	8

Бинарный поиск

1) left = 0, right = 19, mid = 10

1	1	1	2	2	3	3	3	4	4	4	4	5	5	6	6	7	7	8	8

2) left = 11, right = 19, mid = 15

1	1	1	2	2	3	3	3	4	4	4	4	5	5	6	6	7	7	8	8

3) left = 11, right = 14, mid = 12

1	1	1	2	2	3	3	3	4	4	4	4	5	5	6	6	7	7	8	8

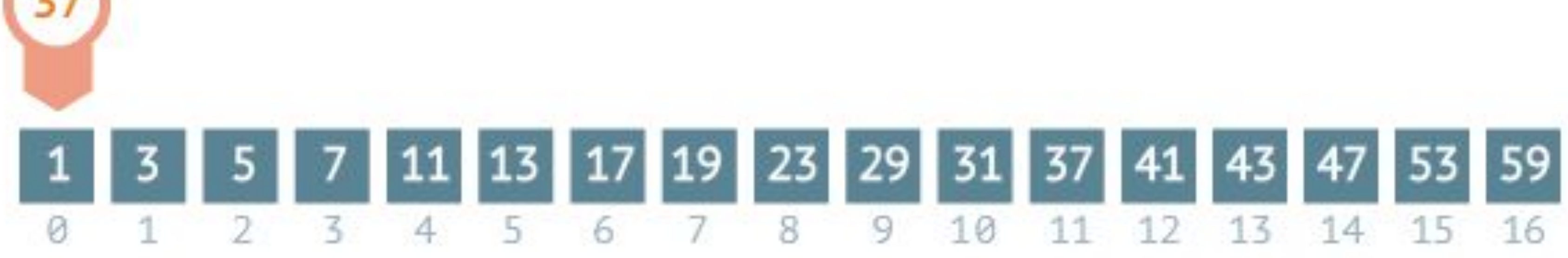
4) left = 13, right = 14

1	1	1	2	2	3	3	3	4	4	4	4	5	5	6	6	7	7	8	8

Методы поиска данных в массиве

Бинарный поиск

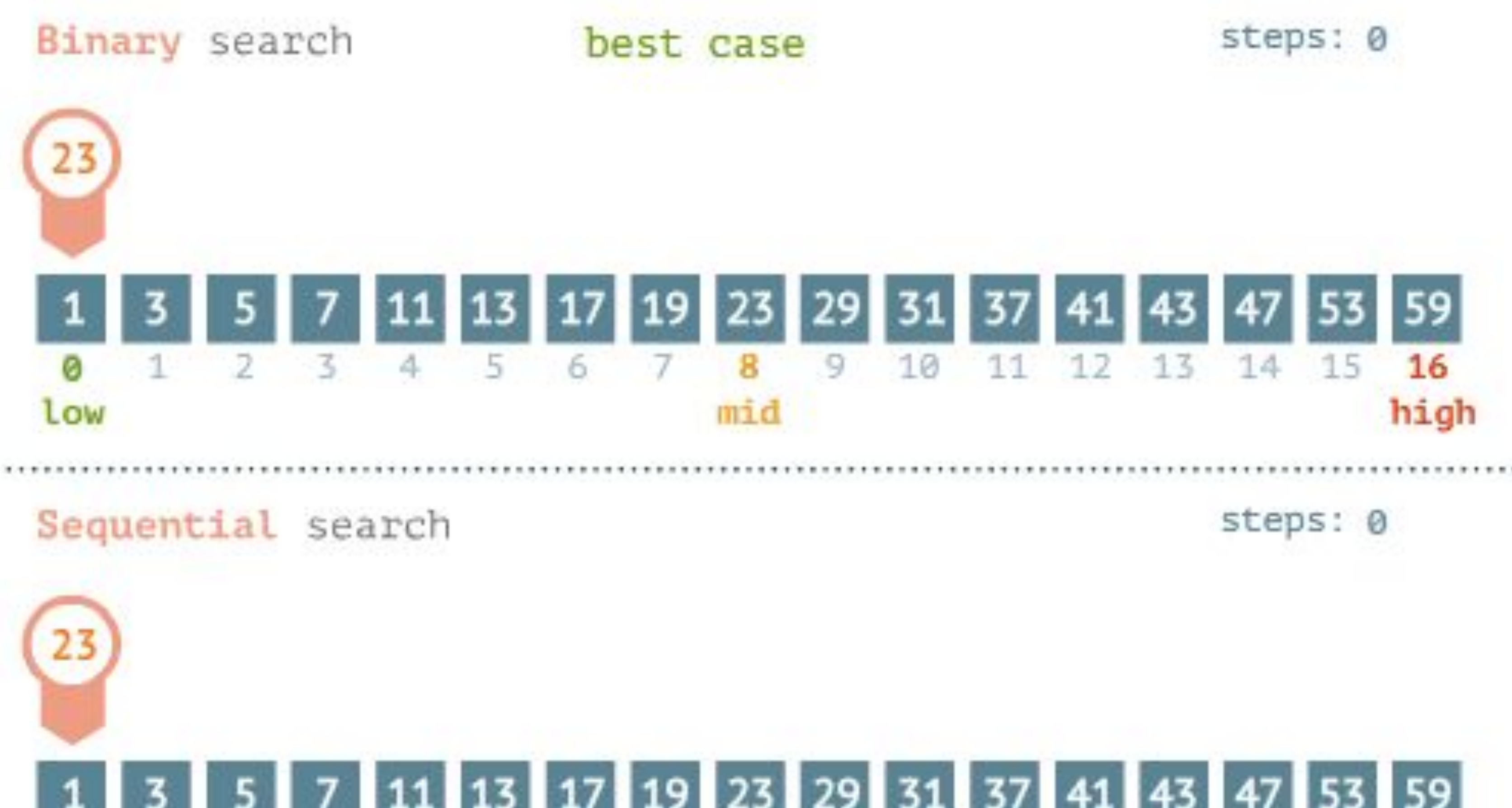
Сравнение с линейным



Методы поиска данных в массиве

Бинарный поиск

Сравнение с линейным. Лучший случай



Методы поиска данных в массиве

Бинарный поиск

Сравнение с линейным. Худший случай

Обработка событий мыши и клавиатуры на графическом окне

OnMouseDown: **procedure** (x,y,mousebutton: integer);
Событие нажатия на кнопку мыши. (x,y) - координаты курсора мыши в момент наступления события, mousebutton = 1, если нажата левая кнопка мыши, и 2, если нажата правая кнопка мыши

OnMouseUp: **procedure** (x,y,mousebutton: integer);
Событие отжатия кнопки мыши. (x,y) - координаты курсора мыши в момент наступления события, mousebutton = 1, если отжата левая кнопка мыши, и 2, если отжата правая кнопка мыши

OnMouseMove: **procedure** (x,y,mousebutton: integer);
Событие перемещения мыши. (x,y) - координаты курсора мыши в момент наступления события, mousebutton = 0, если кнопка мыши не нажата, 1, если нажата левая кнопка мыши, и 2, если нажата правая кнопка мыши.

OnKeyDown: **procedure** (key: integer);
Событие нажатия клавиши. key - виртуальный код нажатой клавиши

OnKeyUp: **procedure** (key: integer);
Событие отжатия клавиши. key - виртуальный код отжатой клавиши

OnKeyPress: **procedure** (ch: char);
Событие нажатия символьной клавиши. ch - символ, генерируемый нажатой символьной клавишей

OnResize: **procedure**;
Событие изменения размера графического окна

OnClose: **procedure**;
Событие закрытия графического окна

Обработка событий мыши и клавиатуры на графическом окне

Пример: Свободное рисование

```
uses GraphABC;

//процедура обработки нажатия мыши
procedure MouseDown(x,y,mb: integer);
begin
  MoveTo(x,y);
end;

//процедура обработки перемещения мыши
procedure MouseMove(x,y,mb: integer);
begin
  if mb=1 then LineTo(x,y); //если нажата левая кнопка мыши
end;

begin
  // Привязка обработчиков к событиям
  OnMouseDown := MouseDown;
  OnMouseMove := MouseMove
end.
```

Обработка событий мыши и клавиатуры на графическом окне

VK_Left	VK_A	VK_LWin	VK_Control
VK_Up	VK_B	VK_RWin	VK_Alt
VK_Right	VK_C	VK_Apps	VK_Modifiers
VK_Down	VK_D	VK_Sleep	VK_Select
VK_PageUp	VK_E	VK_LineFeed	VK_Print
VK_PageDown	VK_F	VK_NumPad0	VK_Snapshot
VK_Prior	VK_G	VK_NumPad1	VK_Scroll
VK_Next	VK_H	VK_NumPad2	VK_LShiftKey
VK_Home	VK_I	VK_NumPad3	VK_RShiftKey
VK_End	VK_J	VK_NumPad4	VK_LControlKey
VK_Insert	VK_K	VK_NumPad5	VK_RControlKey
VK_Delete	VK_L	VK_NumPad6	VK_LMenu
VK_F1	VK_M	VK_NumPad7	VK_RMenu
VK_F2	VK_N	VK_NumPad8	VK_KeyCode
VK_F3	VK_O	VK_NumPad9	VK_Shift
VK_F4	VK_P	VK_Multiply	VK_ShiftKey
VK_F5	VK_Q	VK_Add	VK_ControlKey
VK_F6	VK_R	VK_Separator	VK_Menu
VK_F7	VK_S	VK_Subtract	VK_Pause
VK_F8	VK_T	VK_Decimal	VK_Enter
VK_F9	VK_U	VK_Divide	VK_Return
VK_F10	VK_V	VK_NumLock	VK_Back
VK_F11	VK_W	VK_CapsLock	VK_Tab
VK_F12	VK_X	VK_Capital	VK_Help
	VK_Y	VK_PrintScreen	VK_Space
	VK_Z		

Обработка событий мыши и клавиатуры на графическом окне

```
//Пример перемещения графического окна клавишами клавиатуры
uses GraphABC;

procedure KeyDown(Key: integer);
begin
   case Key of
  VK_Left:  Window.Left := Window.Left - 2;
  VK_Right: Window.Left := Window.Left + 2;
  VK_Up:    Window.Top := Window.Top - 2;
  VK_Down:  Window.Top := Window.Top + 2;
   end;
end;

begin
  // Привязка обработчиков к событиям
  OnKeyDown := KeyDown;
end.
```