Лекция № 9

Алгоритмы, их свойства и виды

Вопросы лекции:

1. Понятие алгоритма.
2. Свойства алгоритма.
3. Формы представления алгоритмов.
4. Основные алгоритмические структуры.

Понятие алгоритма

Алгоритмизация – это процесс построения алгоритма решения задачи, результатом которого является выделение этапов процесса обработки данных, формальная запись содержания этих этапов и определение порядка их выполнения.

Подготовка задачи для решения на ЭВМ состоит из нескольких этапов:

.Постановка задачи
.Формализация задачи
.Построение алгоритма
.Составление программы на языке программирования
.Отладка и тестирование программы
.Проведение расчетов и анализ полученных результатов

Алгоритм - это система правил, описывающая последовательность действий, которые необходимо выполнить, чтобы решить задачу.

Алгоритм - некоторая последовательность предписаний (правил), однозначно определяющих процесс преобразования исходных и промежуточных данных в результат решения задачи.

Свойства алгоритма:

Дискретность означает, что выполнение алгоритма разбивается на последовательность законченных действий - шагов. Каждое действие должно быть завершено исполнителем прежде, чем он перейдет к выполнению следующего. Значения величин в каждом шаге алгоритма получаются по определенным правилам из значения величин, определенных на предшествующем шаге.

Определенность предполагает то обстоятельство, что каждое правило алгоритма настолько четко и однозначно, что значения величин, получаемые на каком-либо шаге, однозначно определяются значениями величин, полученными на предыдущем шаге, и при этом точно известно, какой шаг будет выполнен следующим.

Результативность (и **конечность**) алгоритма предполагает, что его исполнение сводится к выполнению конечного числа действий и всегда приводит к некоторому результату. В качестве одного из возможных результатов является установление того факта, что задача не имеет решений.

Основные характеристики алгоритма:

Массовость понимается, что алгоритм решения задачи разрабатывается в общем виде так, чтобы его можно было применить для целого класса задач, различающихся лишь наборами исходных данных. В этом свойстве и заключена основная практическая ценность алгоритма.

Под **эффективностью** алгоритма будем понимать такое его свойство (качество), которое позволяет решить задачу за приемлемое для разработчика время. К параметру, характеризующему эффективность алгоритма, следует отнести также объем памяти компьютера, необходимый для решения задачи.

Формы представления алгоритмов

1. ***Словесный*** – содержание этапов вычислений задается на естественном языке в произвольной форме с требуемой детализацией.

Словесное описание имеет минимум ограничений и является наименее формализованным. Однако при этом алгоритм получается и наименее строгим, допускающим появление неопределенностей. Также в этой форме алгоритм может оказаться очень объемным и трудным для восприятия человеком.

ПРИМЕР. Пусть задан массив чисел. Требуется проверить, все ли числа принадлежат заданному интервалу. Интервал задается границами A и B.

п.1 Берем первое число. На п.2.
п.2 Сравниваем: выбранное число принадлежит интервалу;
если да, то на **п.3, если нет – на п.6.**
п.3 Все элементы массива просмотрены? Если да, то на п.5,
если нет – то на **п.4.**
п.4 Выбираем следующий элемент. На п.2.
п.5 Печать сообщения: все элементы принадлежат интерва-
лу. На **п.7.**
п.6 Печать сообщения: не все элементы принадлежат интер-
валу. На **п.7.**
п.7 Конец.

При этом способе отсутствует наглядность вычислительного процесса, т. к. нет достаточной формализации.

Формы представления алгоритмов

2. *Формульно-словесный* – задание инструкций с использованием математических символов и выражений в сочетании со словесными пояснениями.

Например, требуется написать алгоритм вычисления площади треугольника по трем сторонам.

п.1 – вычислить полупериметр треугольника

$p=(a+b+c)/2$. К п.2.

п.2 – вычислить $S = \sqrt{p(p-a)(p-b)(p-c)}$

К п.3.

п.3 – вывести S , как искомый результат и прекратить вычисления.

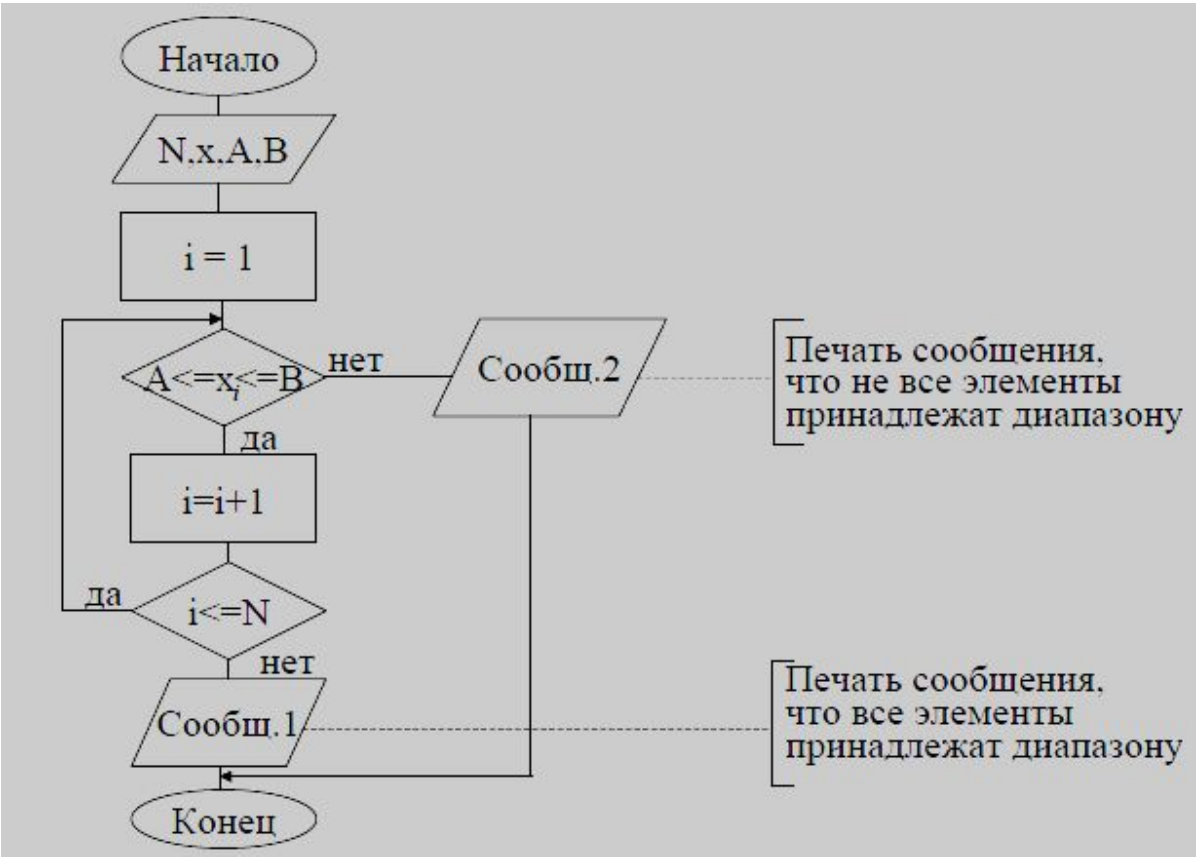
При использовании этого способа может быть достигнута любая степень детализации, более наглядно, но не строго формально.

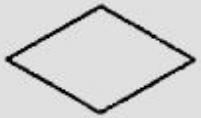
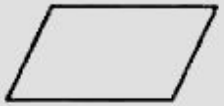
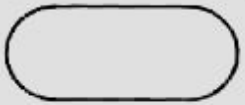
Формы представления алгоритмов

3. *Блок - схемный* – это графическое изображение логической структуры алгоритма, в котором каждый этап процесса переработки данных представляется в виде геометрических фигур (блоков), имеющих определенную конфигурацию в зависимости от характера выполняемых операций.

Внутри блоков указывается поясняющая информация, характеризующая выполняемые ими действия. Конфигурацию и размер блоков, а также порядок построения схем определяет ГОСТ 19002 и ГОСТ19003.

Блок-схемы могут быть традиционные и структурированные.



Название символа	Обозначение	Пояснение
Процесс		Вычислительное действие или последовательность действий
Решение		Блок проверки условия, имеющий один вход и ряд альтернативных выходов, один из которых может быть активизирован после выполнения условий
Модификация		Модификация команды или группы команд с целью воздействия на некоторую последующую функцию
Предопределенный процесс		Процесс, состоящий из одной или нескольких операций (шагов) подпрограммы или модуля
Ввод-вывод		Ввод-вывод информации, при котором отсутствует необходимость в описании фактического носителя данных
Пуск-останов		Начало или конец алгоритма, вход (выход) подпрограммы
Линии потока данных		Отображает поток данных или управления (при необходимости могут быть добавлены стрелки-указатели)
Соединитель		Используется для обрыва линии и продолжения ее в другом месте блок-схемы
Комментарий		Символ используют для добавления комментариев или пояснительных записей

Формы представления алгоритмов

4. Псевдокод **-** позволяет формально изображать логику программы, не заботясь при этом о синтаксических особенностях конкретного языка программирования. Обычно представляет собой смесь операторов языка программирования и естественного языка. Является средством представления логики программы, которое можно применять вместо блок-схемы. Запись алгоритма в виде псевдокода:

Выбираем первый элемент ($i=1$)

IF $A>x_i$ или $x_i>B$ **THEN**

печать сообщения и переход на конец

ELSE

переход к следующему элементу($i=i+1$)

IF массив не кончился ($i <= n$) **THEN**

переход на проверку интервала

ELSE

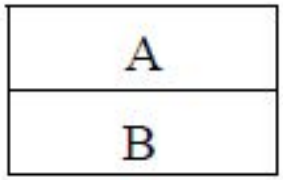
печать сообщения, что все элементы входят в интервал

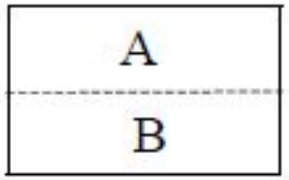
Конец

Формы представления алгоритмов

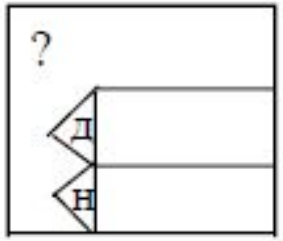
5. Структурные диаграммы - могут использоваться в качестве структурных блок-схем, для показа межмодульных связей, для отображения структур данных, программ и систем обработки данных. Существуют различные структурные диаграммы: диаграммы Насси-Шнейдермана, диаграммы Варнье, Джексона, МЭСИД и др.

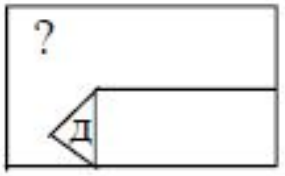
Основные элементы МЭСИД:



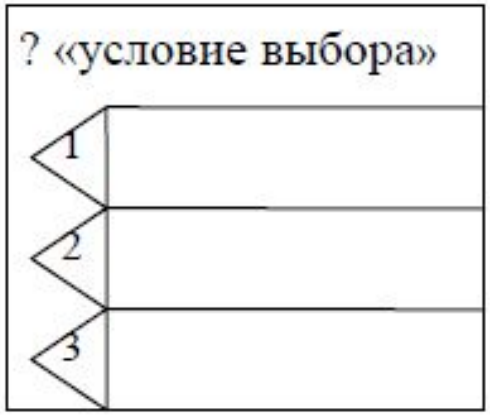


- следование

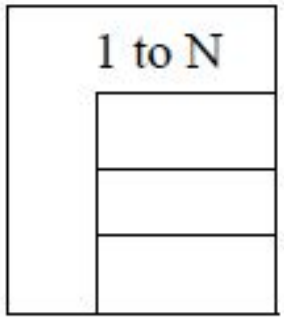


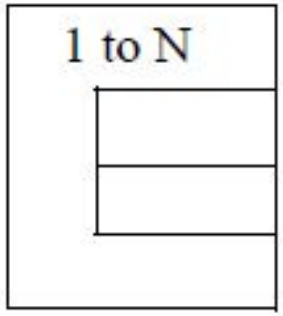


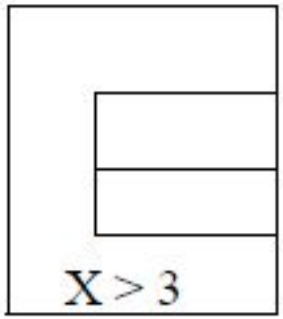
- развилка



- выбор







- повторение

Формы представления алгоритмов

6. Языки программирования - изобразительные средства для непосредственной реализации программы на ЭВМ.

Программа – алгоритм, записанный в форме, воспринимаемой ЭВМ. Каждая машина имеет свой собственный язык (машинный язык) и может выполнять программы только на этом языке. Это последовательность машинных команд. Писать программы на машинном языке очень сложно и утомительно. Для повышения производительности труда программистов применяются искусственные языки программирования. При этом требуется перевод программы, написанной на таком языке, на машинный язык. Этот перевод выполняет ***транслятор***. Наиболее часто встречающимся транслятором интерпретирующего типа является транслятор с языка Бейсик, где команды читаются, преобразуются и выполняются сразу. Итогом работы такого транслятора являются требуемые результаты.

Текст программы на исходном языке сначала переводится в текст на машинном языке и получается так называемый объектный модуль. Затем объектный модуль должен быть обработан программой Редактором межпрограммных связей и только после этого программа будет готова к выполнению.

Виды алгоритмов и их реализация

Алгоритмы в зависимости от цели, начальных условий задачи, путей ее решения, определения действий разработчика **подразделяются на**:

- **механические**, или **детерминированные** (жесткие);
- **гибкие**, или **стохастические** (вероятностные и эвристические).

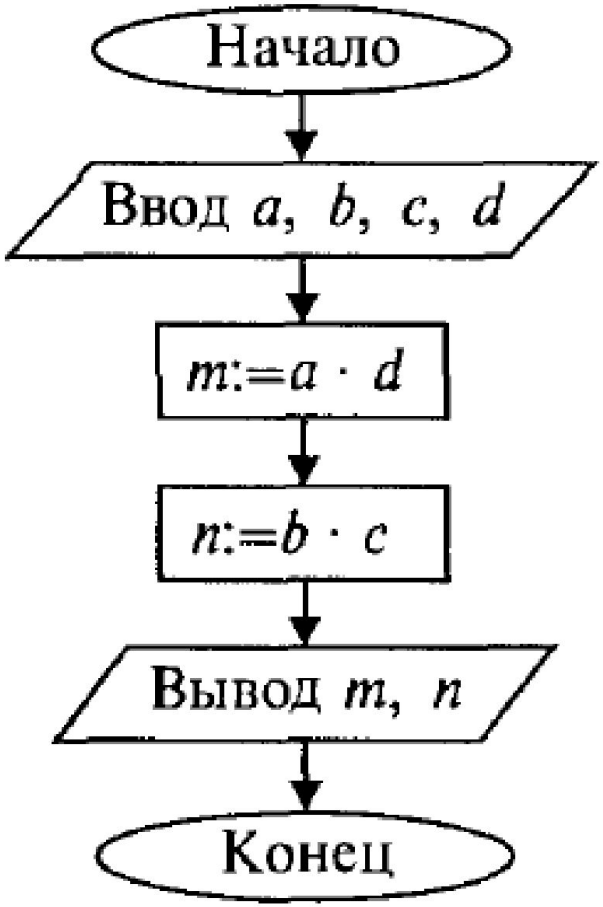
Механический алгоритм задает определенные действия, обозначая их в единственной последовательности, обеспечивающей однозначный требуемый (искомый) результат в том случае, если выполняются условия процесса, для которых разработан алгоритм. К таким алгоритмам относятся алгоритмы работы машин, станков, двигателей и т. п.

Вероятностный (стохастический) алгоритм предлагает программу решения задачи несколькими путями или способами, приводящими к достижению результата.

Эвристический алгоритм (от греческого слова «эврика») — это такой алгоритм, в котором достижение конечного результата однозначно не определено, так же как не обозначена вся последовательность действий. В этих алгоритмах используются универсальные логические процедуры и способы принятия решений, основанные на аналогиях, ассоциациях и прошлом опыте решения похожих задач. При реализации эвристических алгоритмов большую роль играет интуиция разработчика.

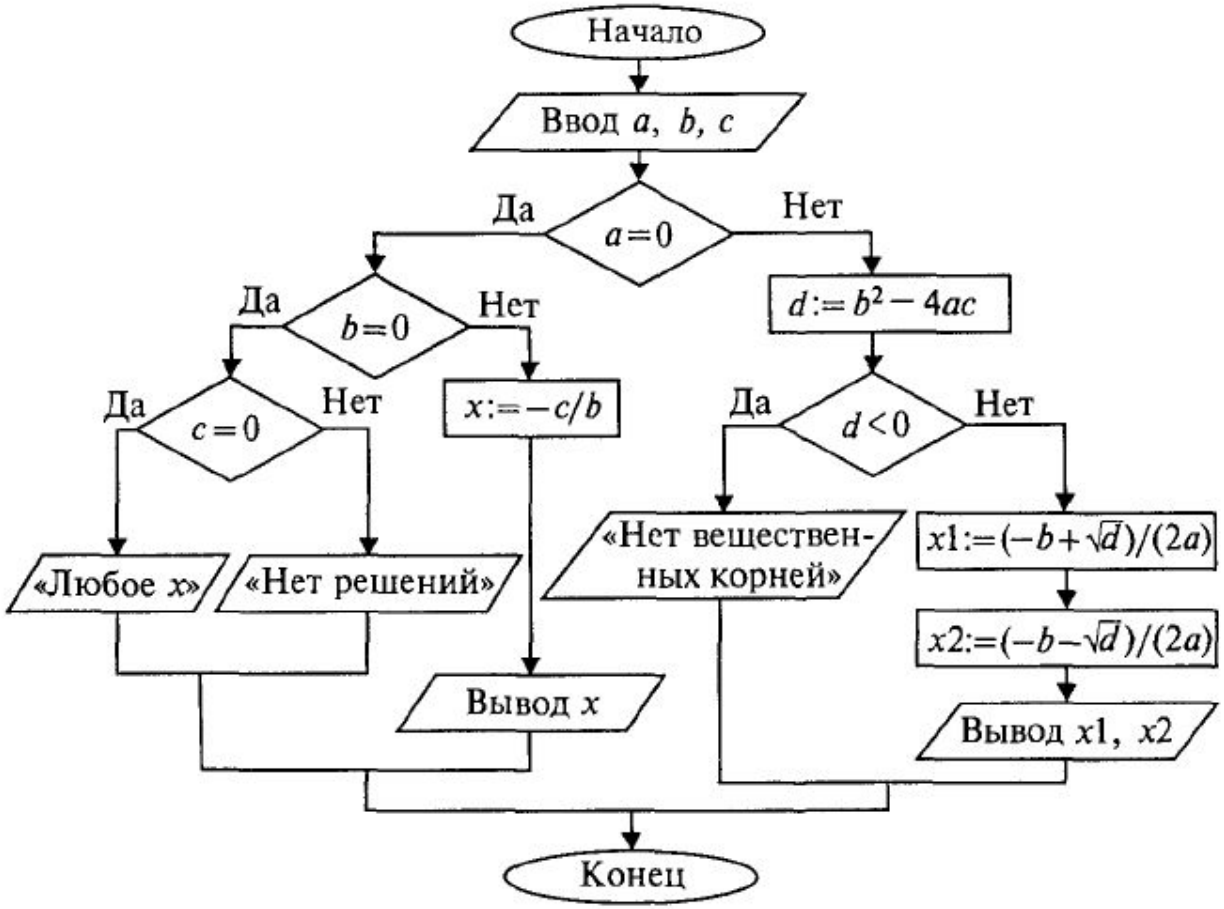
Основные алгоритмические структуры

Линейный вычислительный процесс - это процесс, блоки которого выполняются последовательно один за другим (порядок выполнения блоков естественный).



Основные алгоритмические структуры

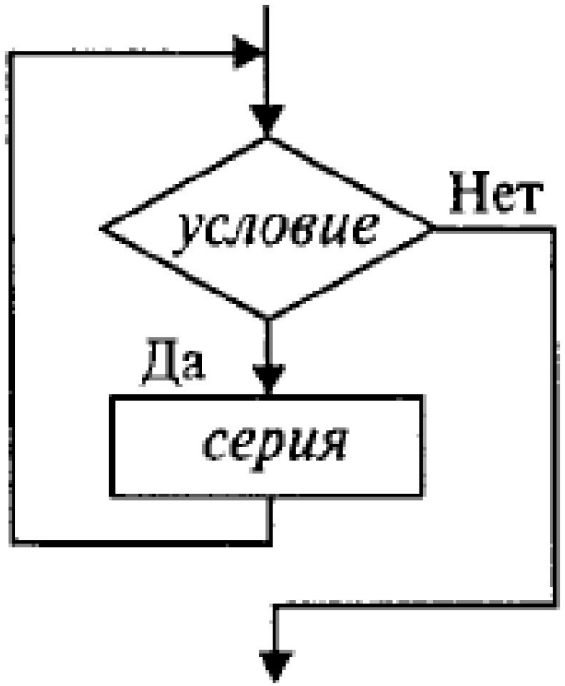
Разветвляющаяся структура используется тогда, когда возникает необходимость в зависимости от исходных данных или от полученных промежуточных результатов осуществлять вычисление по одним или другим формулам, то есть в зависимости от выполнения какого-то логического условия вычислительный процесс должен идти по одной или другой ветви. Такой процесс называют **разветвляющимся**.



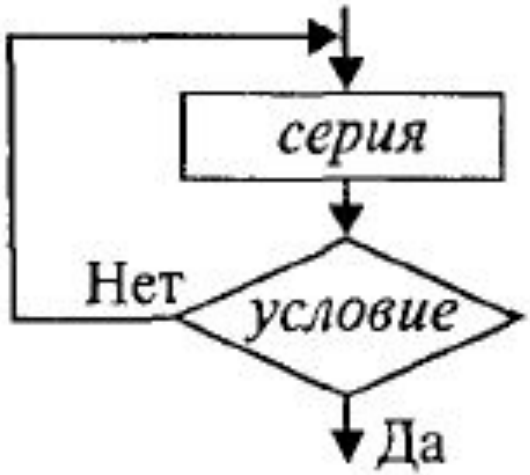
Основные алгоритмические структуры

Циклическая структура. Очень часто встречаются процессы, когда решение задачи сводится к многократному вычислению по одним и тем же математическим зависимостям при различных входящих в них величинах. Многократно повторяющиеся участки этого вычислительного процесса называют **циклами**, а сам процесс - **циклическим**.

Для окончания циклического вычисления необходима проверка некоторого логического условия.



цикл с предусловием



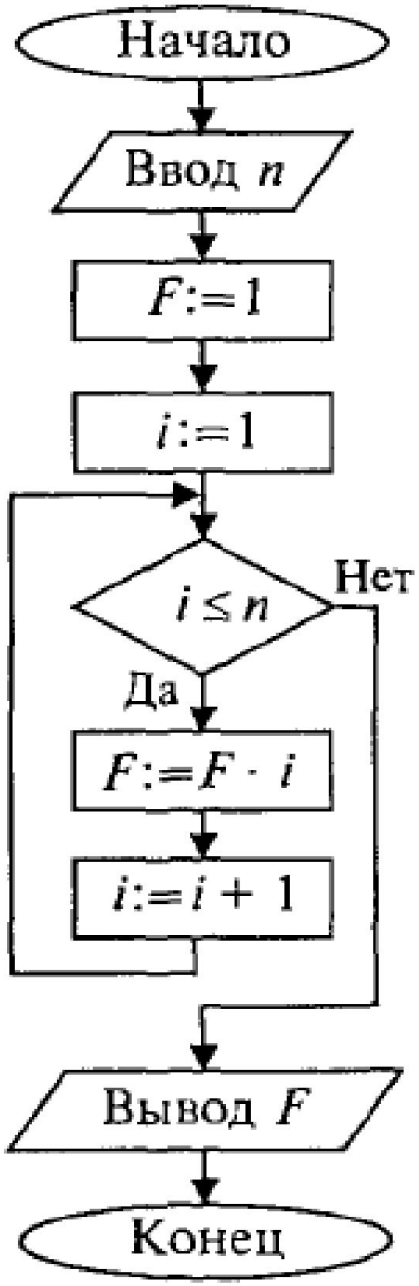
цикл с постусловием

Циклические алгоритмы

В качестве примера рассмотрим алгоритм нахождения факториала натурального числа:

$$n! = \begin{cases} 1, & \text{если } n = 0; \\ 1 \cdot 2 \ldots n, & \text{если } n \geq 1. \end{cases}$$

- Как правило в создаваемых программах присутствуют алгоритмические участки всех видов: линейные, ветвящиеся, циклические.
- Отдельные часто употребляемые алгоритмы можно выносить за пределы основной программы, такие алгоритмы называются **вспомогательными**.
- Согласно теоремам структурного программирования любой алгоритм может быть представлен при помощи операций присваивания, ветвления и цикла.



Структурное программирование

На практике широко используются **два подхода к алгоритмизации**:

1. Традиционный (с использованием блок-схем);
2. Структурный (с использованием структурированной записи).

Традиционный подход к составлению алгоритмов с применением блок-схем грешит большим числом ошибок в программах из-за их громоздкости и запутанности. Из-за этого традиционный подход к составлению программ чреват большим числом ошибок в создаваемых программах.

Структурный подход к программированию заключается в обязательном предварительном составлении структурированных алгоритмов с записью их на псевдокоде. Простота чтения, понимания и исправления структурированных описаний позволяет существенно уменьшить количество ошибок в алгоритмах и программах и сократить время их отладки на ЭВМ.

Структурное программирование

При структурном подходе к составлению алгоритмов и программ используются **три основных правила композиции**:

- альтернативный выбор;
- циклический повтор;
- вспомогательные алгоритмы (подпрограммы).

Структурированными считаются алгоритмы и программы, составленные только с использованием трех правил структурной композиции.

Неструктурированными считаются алгоритмы и программы, в которых используются операторы go to … или отсутствует ступенчатая запись циклов и альтернатив.

Обычно при составлении схемы алгоритма процесс вычисления идет сверху вниз, возвращаясь назад только в циклах, что позволяет анализировать алгоритм как обычный текст, т.е. сверху вниз.

Структурное программирование

Структурное программирование сверху - вниз, идея которого заключается в том, что структура программы должна отражать структуру решаемой задачи, чтобы алгоритм решения был ясно виден из исходного текста. Для этого надо иметь средства для создания программы более точно отражающие конкретную структуру алгоритма.

С этой целью в программирование введено понятие **подпрограммы** — **набора операторов, выполняющих нужное действие и не зависящих от других частей исходного кода.**

Программа разбивается на множество мелких подпрограмм, каждая из которых выполняет одно из действий, предусмотренных исходным заданием. Комбинируя эти подпрограммы, удается формировать итоговый алгоритм уже не из простых операторов, а из законченных блоков кода, имеющих определенную смысловую нагрузку, причем обращаться к таким блокам можно по названиям. Получается, что **подпрограммы — это новые операторы или операции языка, определяемые программистом**.

Возможность применения подпрограмм относит язык программирования к классу ***процедурных*** языков.

Структурное программирование

Сначала выделяется несколько подпрограмм, решающих самые глобальные задачи (например, инициализация данных, главная часть и завершение), потом каждый из этих модулей детализируется на более низком уровне, разбиваясь в свою очередь на небольшое число других подпрограмм, и так происходит до тех пор, пока вся задача не окажется реализованной.

Такой подход удобен тем, что позволяет человеку постоянно мыслить на предметном уровне, не опускаясь до конкретных операторов и переменных. Кроме того, появляется возможность некоторые подпрограммы не реализовывать сразу, а временно откладывать, пока не будут закончены другие части.

Немаловажно, что небольшие подпрограммы значительно проще отлаживать, что существенно повышает общую надежность всей программы.

Очень важная характеристика подпрограмм — это возможность их ***повторного использования***. С интегрированными системами программирования поставляются большие библиотеки стандартных подпрограмм, которые позволяют значительно повысить производительность труда за счет использования чужой работы по созданию часто применяемых подпрограмм.

Структурное программирование

Подпрограммы бывают двух видов — ***процедуры*** и ***функции***. Отличаются они тем, что процедура просто выполняет группу операторов, а функция вдобавок вычисляет некоторое значение и передает его обратно в главную программу *(возвращает* значение). Это значение имеет определенный тип (говорят, что функция *имеет* такой-то тип).

Чтобы работа подпрограммы имела смысл, ей надо получить данные из внешней программы, которая эту подпрограмму *вызывает*. Данные передаются подпрограмме в виде ***параметров*** или ***аргументов***, которые обычно описываются в ее заголовке так же, как переменные.

Подпрограммы вызываются, как правило, путем простой записи их названия с нужными параметрами. В Бейсике есть оператор **CALL** для явного указания того, что происходит вызов подпрограммы.

Подпрограммы активизируются *только* в момент их вызова. Операторы, находящиеся внутри подпрограммы, выполняются, только если эта подпрограмма явно вызвана. Пока выполнение подпрограммы полностью не закончится, оператор главной программы, следующий за командой вызова подпрограммы, выполняться не будет.

Подпрограммы могут быть ***вложенными*** — допускается вызов подпрограммы не только из главной программы, но и из любых других подпрограмм.

Структурное программирование

В некоторых языках программирования допускается вызов подпрограммы из себя самой. Такой прием называется ***рекурсией*** и потенциально опасен тем, что может привести к зацикливанию — бесконечному самовызову.

Подпрограмма состоит из нескольких частей: заголовка с параметрами, тела подпрограммы (операторов, которые будут выполняться при ее вызове) и завершения подпрограммы.

Локальные переменные, объявленные внутри подпрограммы, имеют областью действия только ее тело.

После того как функция рассчитала нужное значение, ей требуется явно вернуть его в вызывающую программу. Для этого может использоваться специальный оператор или особая форма оператора присваивания, когда в левой части указывается имя функции, а справа — возвращаемое значение.

Пример функции, вычисляющей значение квадрата аргумента в Бейсике:

```
FUNCTION SQR% (X  AS  INTEGER)
SQR%  =  X*X
END  FUNCTION
```

Структурное программирование

Во время создания подпрограммы заранее не известно, какие конкретно параметры она может и будет получать. Поэтому в качестве переменных, выступающих в роли ее аргументов в заголовке, могут использоваться произвольные допустимые названия, даже совпадающие с уже имеющимися. Компилятор все равно поймет, что это не одно и то же.

Параметры, которые указываются в заголовке подпрограммы, называются ***формальными***. Они нужны только для описания тела подпрограммы. А параметры (конкретные значения), которые указываются в момент вызова подпрограммы, называются ***фактическими*** параметрами. При выполнении операторов подпрограммы формальные параметры как бы временно заменятся на фактические.

Пример.

```
int a,y;
 a = 5;
 y = SQR(a);
```

Программа вызывает функцию SQR() с одним фактическим параметром «a». Внутри подпрограммы формальный параметр «x» получает значение переменной «a» и возводится в квадрат. Результат возвращается обратно в программу и присваивается переменной «y».

Объектно-ориентированное программирование

Реальные объекты окружающего мира обладают тремя базовыми характеристиками: они имеют **набор свойств**, способны разными методами изменять эти свойства и реагировать на события, возникающие как в окружающем мире, так и внутри самого объекта. Именно в таком виде в языках программирования и реализовано понятие *объекта* как совокупности **свойств** (структур данных, характерных для этого объекта), **методов** их обработки (подпрограмм изменения свойств) и **событий**, на которые данный объект может реагировать и которые приводят, как правило, к изменению свойств объекта.

Объекты могут иметь идентичную структуру и отличаться только значениями свойств. В таких случаях в программе создается новый тип, основанный на единой структуре объекта. Он называется ***классом***, а каждый конкретный объект, имеющий структуру этого класса, называется ***экземпляром класса***.

Описание нового класса похоже на описание новой структуры данных, только к полям (свойствам) добавляются методы — подпрограммы.

При определении подпрограмм, принадлежащих конкретному классу, его методов, в заголовке подпрограммы перед ее названием явно указывается, к какому классу она принадлежит.

Класс — это тип данных, такой же, как любой другой базовый или сложный тип. На его основе можно описывать конкретные объекты (экземпляры классов).

Объектно-ориентированное программирование

Объектно-ориентированное программирование – это методология программирования, основанная на представлении программы в виде совокупности объектов, каждый из которых является экземпляром определенного класса, а классы образуют иерархию наследования.

В данном определении можно выделить три части:

- ООП использует в качестве базовых элементов объекты, а не алгоритмы;
- каждый объект является экземпляром какого-либо определенного класса;
- классы организованы иерархически.

ООП часто называют новой ***парадигмой*** программирования, у которой имеются следующие предшественники:

- директивная (структурное программирование – Pascal, C)$
- логическая – Prolog;
- функциональная – Lisp, Effel.

Объектно-ориентированное программирование

Концепция ООП подразумевает, что основой управления процессом реализации программы является передача сообщений объектам. Поэтому объекты должны определяться совместно с сообщениями, на которые они должны реагировать при выполнении программы.

Объектно-ориентированный язык программирования должен обладать следующими свойствами:

1. **абстракции** – формального представления о качествах или свойствах предмета путем мысленного удаления некоторых частностей или материальных объектов;
2. **инкапсуляции** – механизма, связывающего вместе код и данные, которыми он манипулирует, и защищающего их от внешних помех и некорректного использования;
3. **наследования** – процесса, с помощью которого один объект приобретает свойства другого, т.е. поддерживается иерархическая классификация;
4. **полиморфизма** – свойства, позволяющего использовать один и тот же интерфейс для общего класса действий.

Объектно-ориентированное программирование

Разработка объектно-ориентированных программ состоит из следующих последовательных работ:

- определение основных объектов, необходимых для решения данной задачи;
- определение закрытых данных (данных состояния) для выбранных объектов;
- определение второстепенных объектов и их закрытых данных;
- определение иерархической системы классов, представляющих выбранные объекты;
- определение ключевых сообщений, которые должны обрабатывать объекты каждого класса;
- разработка последовательности выражений, которые позволяют решить поставленную задачу;
- разработка методов, обрабатывающих каждое сообщение;
- очистка проекта, т.е. устранение всех вспомогательных промежуточных материалов, использовавшихся при проектировании;
- кодирование, отладка, компоновка и тестирование.

Литература:

1. Каймин В.А. Информатика: учебник. – 5-е изд. – М.: ИНФРА-М, 2008. -285 с.
2. Колдаев В.Д. Основы алгоритмизации и программирования: Учебное пособие / Под ред.проф. Л. Г. Гагариной. – М.: ИД «Форум»: ИНФРА-М, 2006. -416 с.
3. Дукин А. Н. Самоучитель Visual Basic 2010 / А. Н. Дукин, А. А. Пожидаев. – СПб.: БХВ-Петербург. 2010 - 560 с.:ил.
4. Зибров В.В. VISUAL BASIC 2010 на примерах. – СПб.: БХВ-Петербург, 2010. – 336 с.:ил.