

# **Основы алгоритмизации и программирования на языках высокого уровня**

- **Лекции: 32 часа**
- **Лабораторные работы: 40 часов**
- **Самостоятельная работа: 26 часа**

**srv-iit3\courses3\BNL\ОАПЯВУ**

**логин: ИТ7\spfuser**

**пароль: ИТ7user**

- **Конспект лекций по курсу:**

**М.П. Батура, В.Л. Бусько, А.Г. Корбит, Т.М. Кривоносова**  
**ОСНОВЫ АЛГОРИТМИЗАЦИИ И ПРОГРАММИРОВАНИЯ.**  
**ЯЗЫК СИ. – Мн.: БГУИР, 2007г.**

- **Е.М. Демидович ОСНОВЫ ИНФОРМАТИКИ И**  
**ПРОГРАММИРОВАНИЯ. – Мн.: Изд-во МИУ (в 2х частях),**  
**2005г.**

- **Шилт Г., Самоучитель С++. – СПб.: ВНУ, 2009г.**

- **Побегайло А.П. С.С++ для студента. СПб: БХВ-Петербург, 2006г.**
- **Либерти Д. Джонс Б. Освой самостоятельно С++ за 21 день. – М.: Вильямс, 2006г.**
- **Пахомов Б. С/С++ и MS Visual 2008 С++ для начинающих. – СПб.: БХВ–Петербург, 2009г.**

# **Тема 1. Введение в алгоритмизацию**

**1.1 Решение задач с использованием средств программирования. Алгоритм. Свойства алгоритмов.**

**1.1 Графическое описание алгоритма. Стандартизация графического представления алгоритмов.**

# Языки программирования можно разделить

- 1) **Машинно-ориентированные (Ассемблер)**
- 2) **Процедурно-ориентированные (Pascal, Fortran, C)**
- 3) **Объектно-ориентированные (C++, Java, C#)**
- 4) **Языки логических программ (Prolog)**
- 5) **Языки решения интеллектуальных задач (Lisp, СУБД+СППР, ЭС)**
- 6) **Языки описания сценариев (Perl, Visual Basic, ASP)**

# 1.1 Решение задач с использованием средств программирования. Алгоритм. Свойства алгоритмов

**Решение задачи с использованием средств программирования можно разбить на следующие этапы:**

- **математическая или информационная формулировка задачи;**
- **выбор метода (например, численного) решения поставленной задачи;**
- **построение алгоритма решения поставленной задачи;**
- **запись построенного алгоритма, т.е. написание текста программы;**
- **отладка программы – процесс обнаружения, локализации и устранения возможных ошибок;**
- **выполнение программы – получение требуемого результата.**

# Алгоритм и его свойства

- **Под алгоритмизацией понимается сведение задачи к последовательности этапов, выполняемых друг за другом так, что результаты предыдущих этапов используются при выполнении следующих.**
- **Алгоритмом называется система правил, четко описывающая последовательность действий, которые необходимо выполнить для решения задачи.**

# Свойства алгоритмов

**Дискретность** – значения новых величин (**выходных данных**) вычисляются по определенным правилам из других величин с уже известными значениями (**входные данные**).

**Определенность (детерминированность)** – каждое правило из системы однозначно, а данные однозначно связаны между собой, т.е. последовательность действий алгоритма строго и точно определена.

**Результативность (конечность)** – алгоритм решает поставленную задачу за конечное число шагов.

**Массовость** – алгоритм разрабатывается так, чтобы его можно было применить для целого класса подобных задач.



# **Способы описания алгоритмов**

- **Запись на естественном языке(словесное описание)**
- **Изображение в виде схем(графическое описание)**
- **Запись на алгоритмическом языке (программа)**

# Словесное описание

- **В программировании метаязыком называется язык, предназначенный для описания языка программирования.**
- **При использовании данного способа для описания алгоритмов используются следующие типовые этапы:**

- **Этап обработки(вычисления)**

**V=выражение**

**Где V – переменная**

- **Проверка условия**

**Если условие, то идти к N**

- **Переход к этапу с номером N**

**Идти к N**

- **Конец вычислений**

**Останов.**

# Пример

- **Дать словесное описание алгоритма решения квадратного уравнения  $a*x^2+b*x+c=0$**
1.  **$D=b^2-4*a*c$**
  2. **Если  $D<0$ , идти к 4**
  3.  **$x_1=(-b+ \sqrt{D})/(2*a)$   
 $x_2=(-b- \sqrt{D})/(2*a)$**
  4. **Останов.**

**НЕДОСТАТОК** – *малая наглядность*

# **1.2 Графическое описание алгоритма. Стандартизация графического представления алгоритмов**

**Графическое описание алгоритма – это представление алгоритма в виде схемы, состоящей из последовательности блоков (геометрических фигур), каждый из которых отображает содержание очередного шага алгоритма.**

**Внутри фигур кратко записывают выполняемое действие.  
Такую схему называют **схемой программы**.**

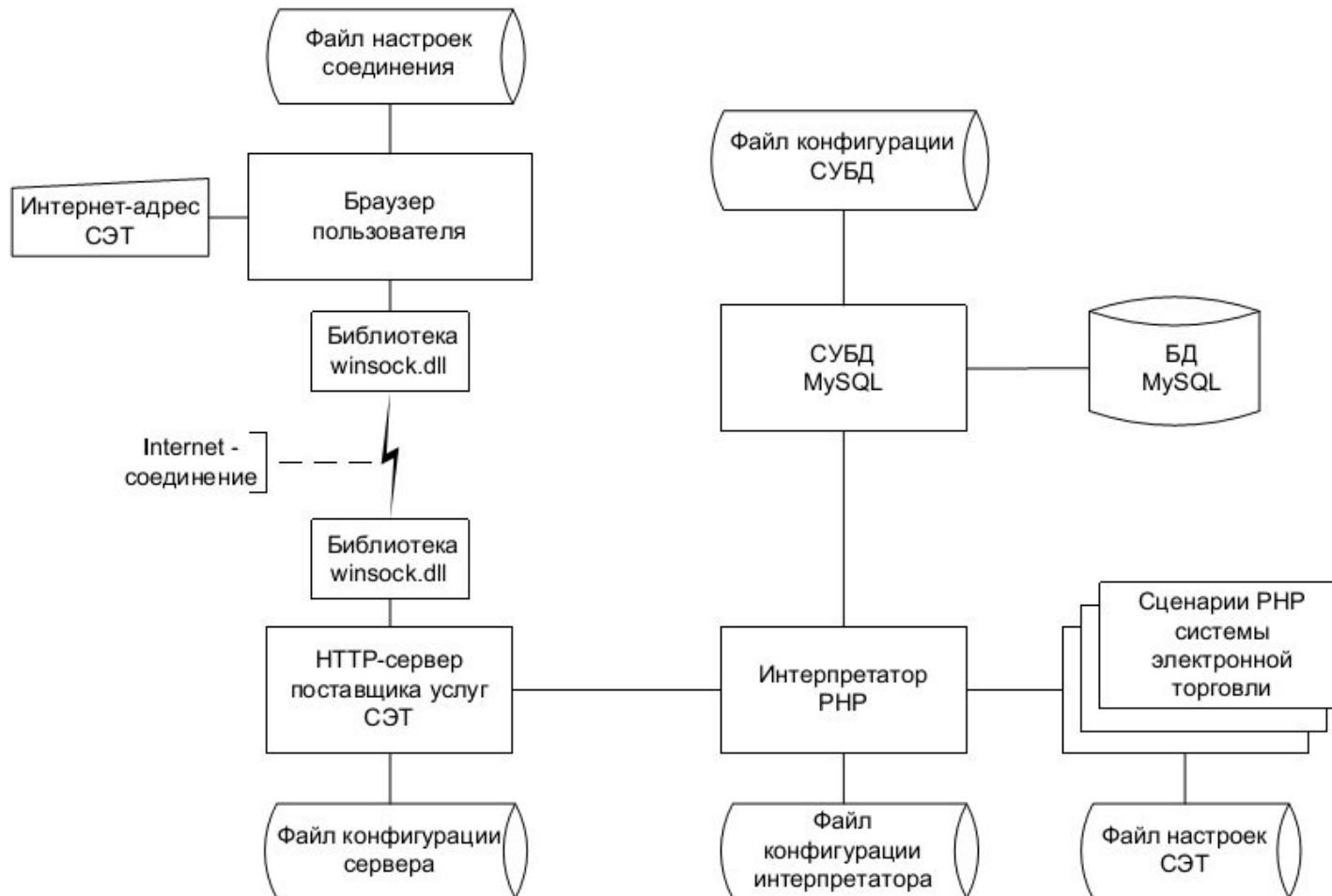
**Схема программы – отображает последовательность операций  
в программе.**

**ГОСТ 19.701-90,  
ISO – International Standards Organization – Международная  
организация по стандартизации 5807-85 "Схемы алгоритмов,  
данных, программ и систем "**

## **виды схем, предназначенные для использования в программной документации**

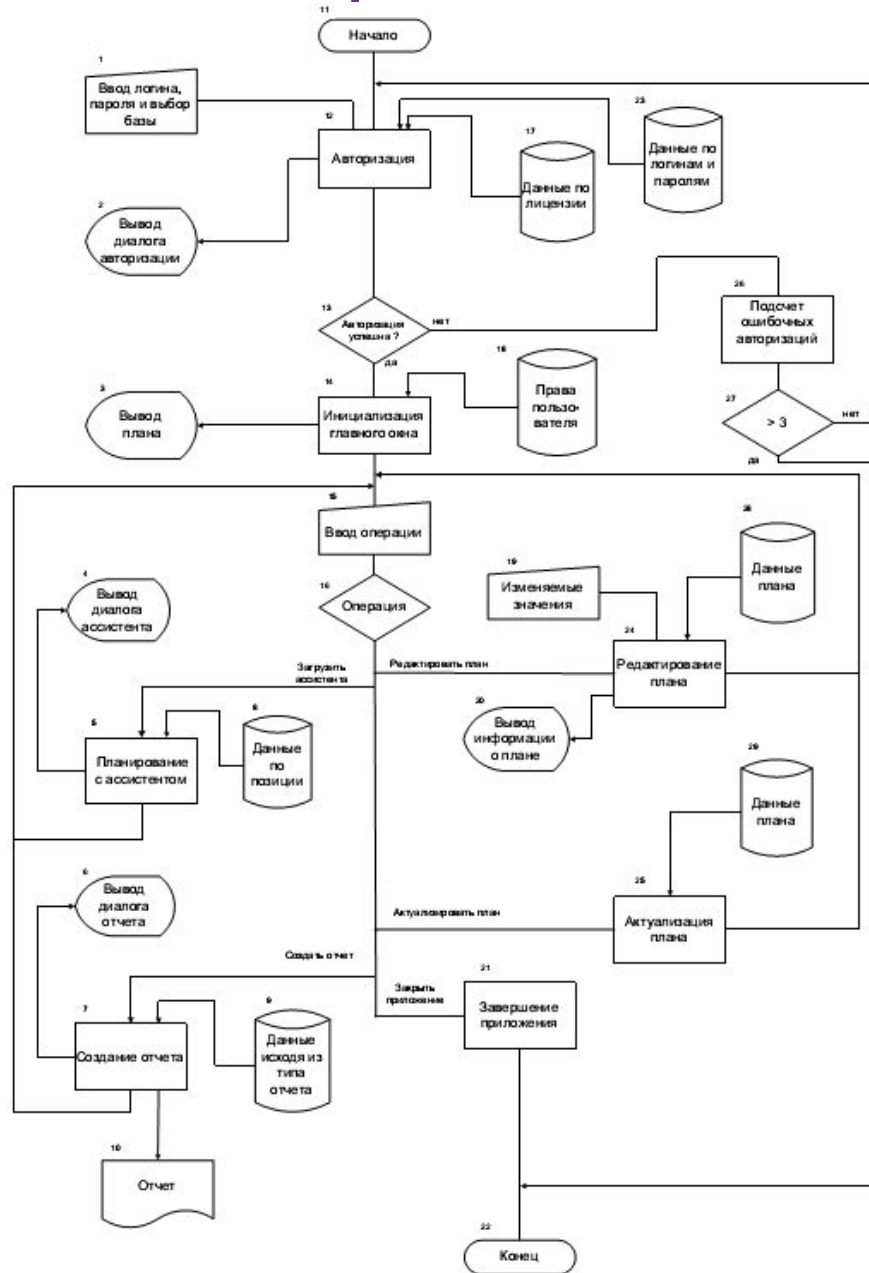
- ***Схема данных*** отображают путь данных при решении задач и определяют этапы обработки, а также различные применяемые носители
- ***Схема программ*** отображают последовательность операций в программе(аналогично схеме алгоритма)
- ***Схема работы системы*** отображают управление операциями и потоки данных в системе. В схеме работы системы каждая программа может отображаться более чем в одном потоке управления.
- ***Схема взаимодействия программ*** отображают путь активации программы и взаимодействия с соответствующими данными. Каждая программа в схеме взаимодействия программ показывается только один раз.
- ***Схема ресурсов системы*** отображает конфигурацию блоков данных и обрабатывающих блоков, которая требуется для решения задач.

# Схема ресурсов системы

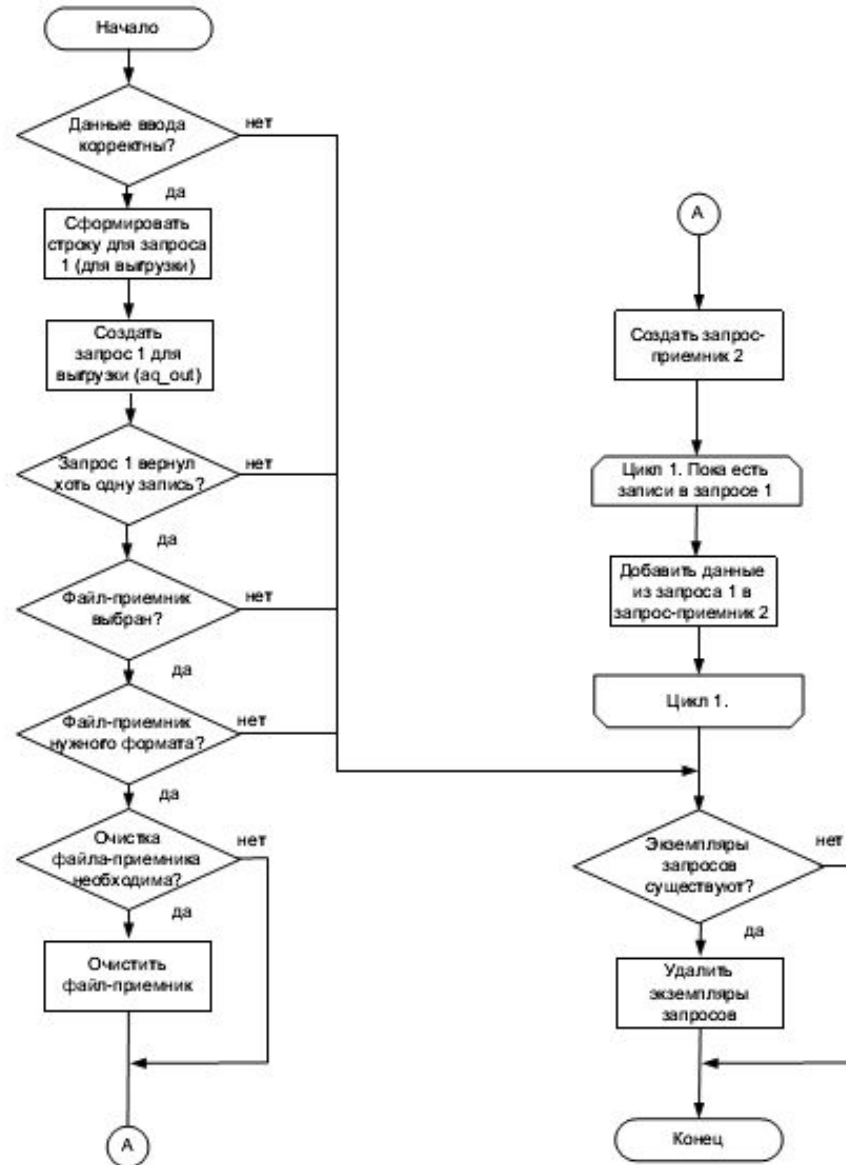




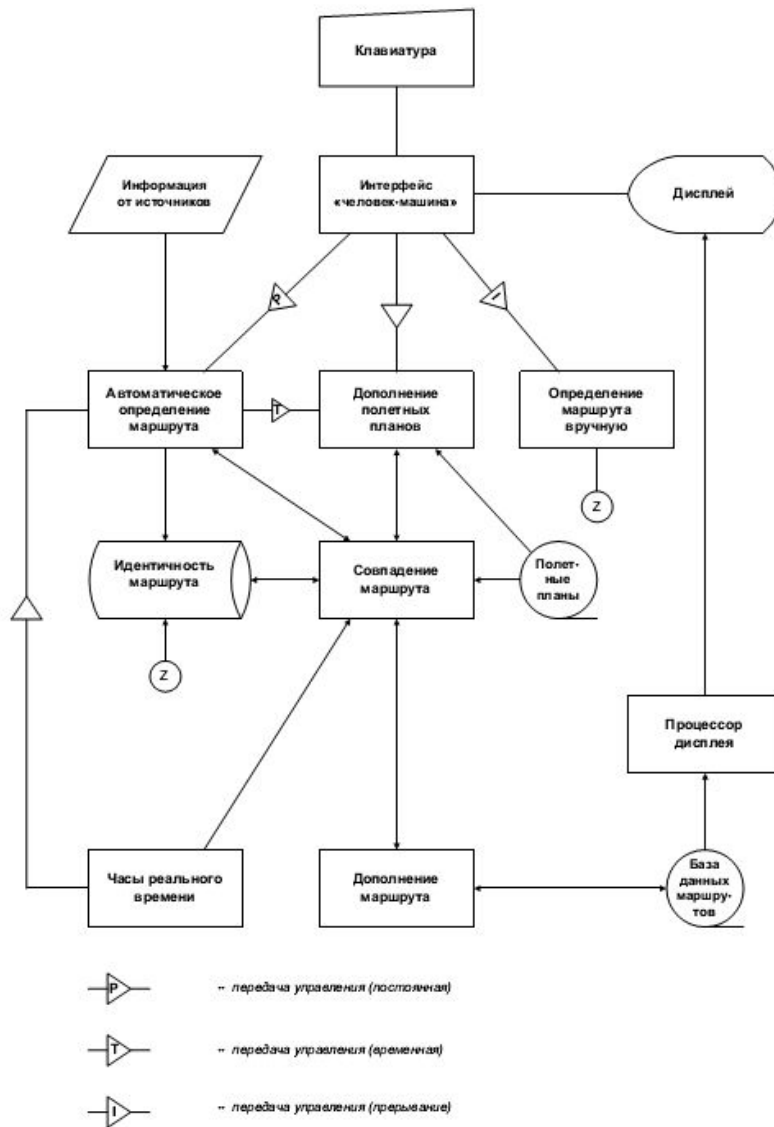
# Схема работы системы



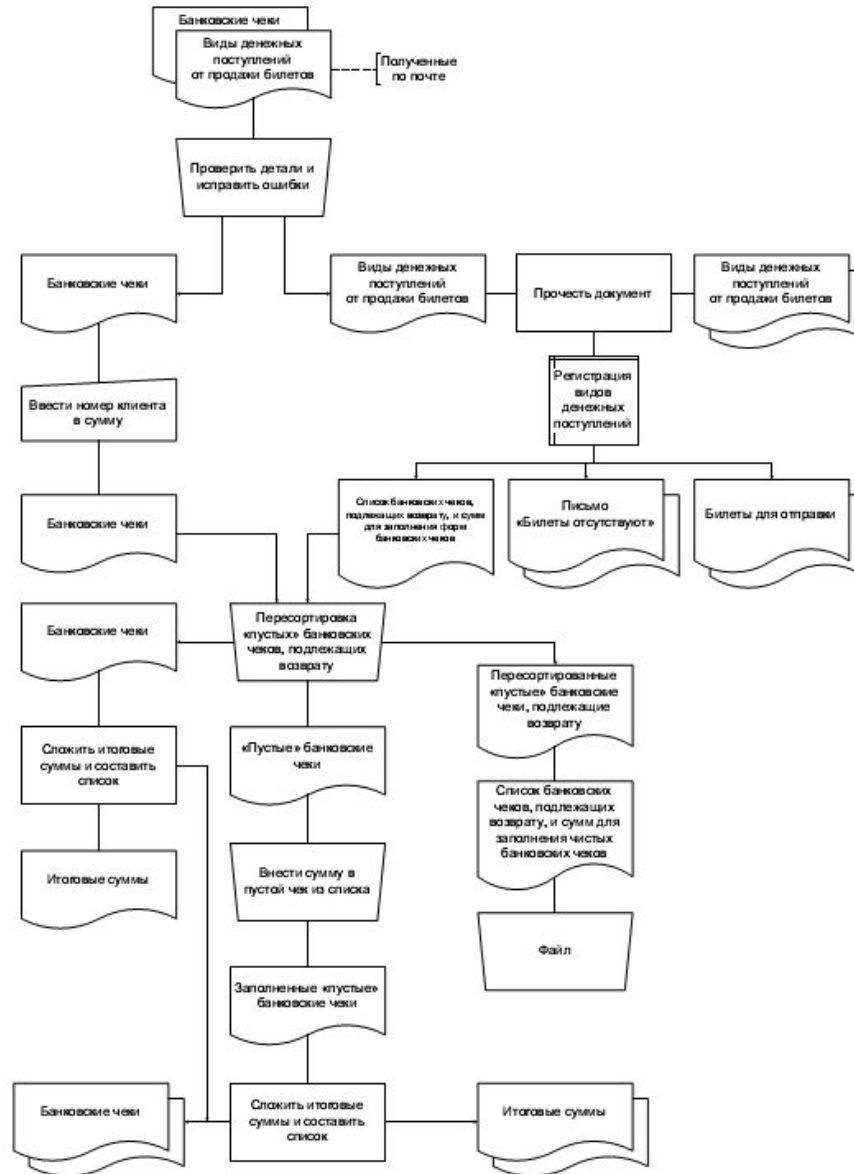
# Схема программы



# Схема взаимодействия программ



# Схема данных



## **Схема состоит из символов четырех типов**

- 1) Символов данных (могут отображать тип носителя данных);**
- 2) Символы процессов ( выполняемых над данными);**
- 3) Символы линий, указывающих потоки данных между процессами и носителями данных;**
- 4) Специальные символы (для удобства чтения схемы).**

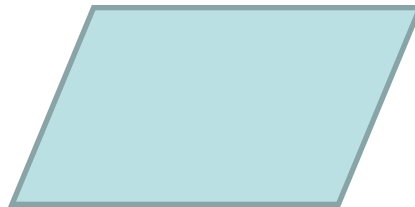
**Каждая из трех первых групп в свою очередь подразделяется на две подгруппы:**

- Основные символы**
- Специфические символы**

# Символы данных

- **К основным символам данных относятся символы, не конкретизирующие носитель данных.**

**Данные, носитель которых не определен**



# Запоминаемые данные

**Символ отображает хранимые данные.**

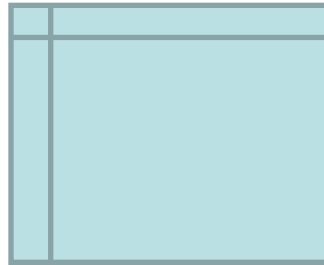
**Конкретный носитель данных не определяется.**

**Данный символ используют в схемах программ, например, для изображения результирующей информации, которую нужно запомнить, причем тип приемника значения не имеет.**



# **Специфические символы данных конкретизируют носитель входных/выходных данных**

- **Оперативное запоминающее устройство**



**Отображает данные, хранящиеся в оперативном  
запоминающем устройстве**

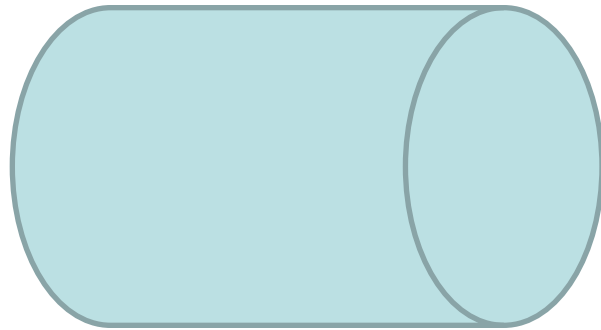


# Ручной ввод



**Символ отображает данные, вводимые вручную  
во время обработки с устройства любого типа**

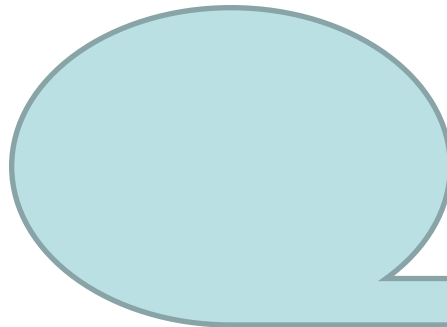
# Запоминающее устройство с прямым доступом



- **Символ отображает данные, хранящиеся в запоминающем устройстве с прямым доступом(например магнитный диск)**

# **Запоминающее устройство с последовательным доступом**

- **Символ отображает данные, хранящиеся в запоминающем устройстве с последовательным доступом (магнитная лента, кассета с магнитной лентой, магнитофонная кассета).**



# Дисплей



**Символ отображает данные, представляемые в удобной для человека форме на отображающем устройстве(дисплей)**

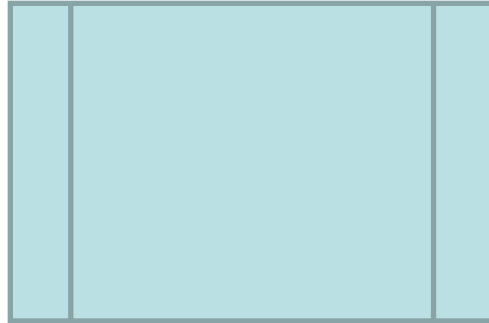
# Символы процесса

**Процесс**



**Вычислительные операции любого вида  
можно изображать только с помощью  
данного символа**

# Предопределенный процесс



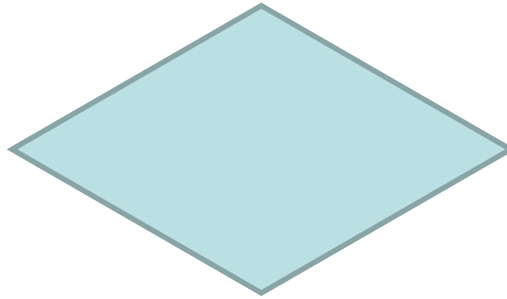
**Символ, отображает процесс, состоящий из одной или нескольких операций или шагов программы, которые определены в другом месте**

# Подготовка



**Символ отображает модификацию команды или группы команд с целью воздействия на некоторую последующую функцию(установка переключателя, модификация индексного регистра или инициализация программы)**

# Решение

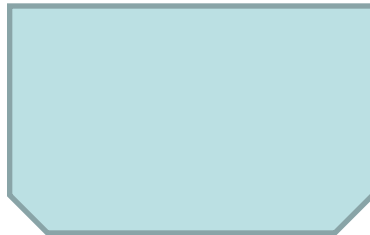
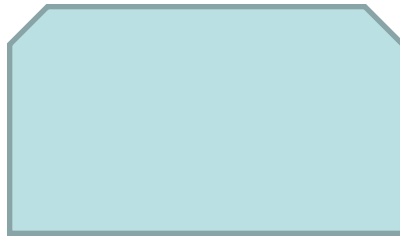


- **Символ отображает функцию переключательного типа, имеющую один вход и ряд альтернативных выходов, один из которых активизируется после вычисления условий, записанных внутри этого символа. Соответствующие результаты вычисления записываются рядом с линиями, отображающими эти выходы**



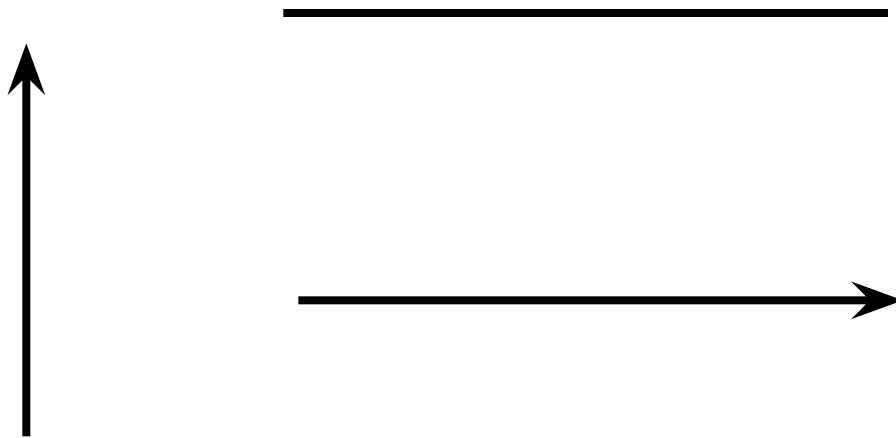
# Граница цикла

**Символ состоит из двух частей,  
отображающих начало и конец цикла.  
Обе части символа должны иметь один и  
тот же идентификатор**



# Символы линий

**Линия – отображает поток данных или управления. При необходимости или для повышения удобочитаемости к линии могут быть добавлены стрелки**



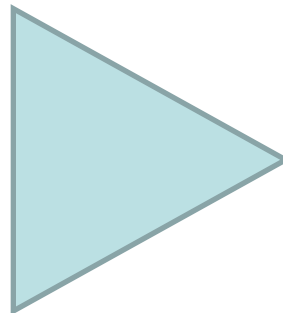
# Пунктирная линия

**В схемах программ используется для  
обведения выделяемого участка, а также  
как часть символа комментария**



# Передача управления

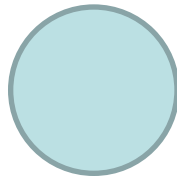
- **Символ отображает непосредственную передачу управления от одного процесса к другому, иногда с возможностью прямого возвращения к инициирующему процессу после того, как инициированный процесс завершит свои функции. Тип передачи управления должен быть назван внутри символа (например, запрос, вызов, событие).**



# **Специальные символы**

**Соединитель отображает выход в другую часть схемы и вход из другой части этой схемы и используется для обрыва линии и продолжения ее в другом месте.**

**Для обозначения используются буквы или арабские цифры**



# Терминатор

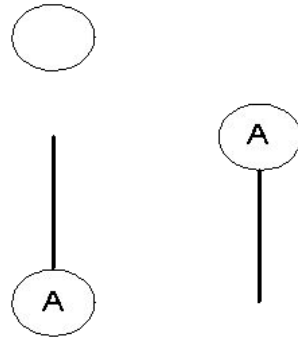
**Терминатор отображает выход во внешнюю среду и вход из внешней среды ( в схемах программы – это начало и конец программы)**



- **Комментарий используют для добавления комментариев (пояснительных записей). Пунктирная линия связана с соответствующим символом и может обводить группу символов, если комментарии относятся ко всей группе.**
- **Пропуск применяется в схемах для отображения пропуска символа или группы символов. Используется только в символах линии или между ними.**

. . . .

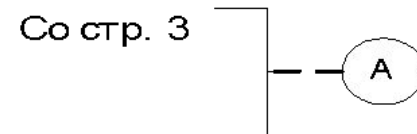
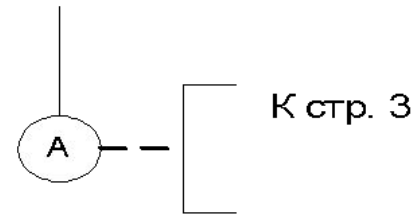
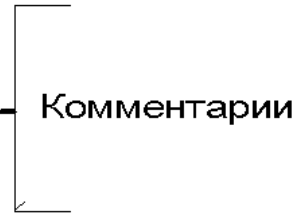
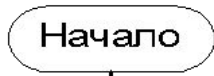
# Специальные СИМВОЛЫ



**Соединитель** – используется при обрыве линии и продолжении ее в другом месте .



**Терминатор** – вход из внешней среды или выход во внешнюю среду (начало или конец схемы программы).





# **ПРАВИЛА ПРИМЕНЕНИЯ СИМВОЛОВ И ВЫПОЛНЕНИЯ СХЕМ**

- **Символ предназначен для графической идентификации функции, которую он отображает, независимо от текста внутри этого символа.**
- **Символы в схеме должны быть расположены равномерно. Следует придерживаться разумной длины соединений и минимального числа длинных линий.**

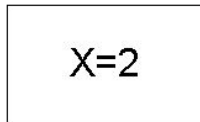
# **ПРАВИЛА ПРИМЕНЕНИЯ СИМВОЛОВ И ВЫПОЛНЕНИЯ СХЕМ**

- **Большинство символов задумано так, чтобы дать возможность включения текста внутри символа. Формы символов, установленные настоящим стандартом, должны служить руководством для фактически используемых символов. Не должны изменяться углы и другие параметры, влияющие на соответствующую форму символов. Символы должны быть, по возможности, одного размера.**
- **Символы могут быть вычерчены в любой ориентации, но, по возможности, предпочтительной является горизонтальная ориентация. Зеркальное изображение формы символа обозначает одну и ту же функцию, но не является предпочтительным.**

# **ПРАВИЛА ПРИМЕНЕНИЯ СИМВОЛОВ И ВЫПОЛНЕНИЯ СХЕМ**

- **Минимальное количество текста, необходимого для понимания функции данного символа, следует помещать внутри данного символа. Текст для чтения должен записываться слева направо и сверху вниз независимо от направления потока.**
- **Если объем текста, помещаемого внутри символа, превышает его размеры, следует использовать символ комментария.**
- **Если использование символов комментария может запутать или разрушить ход схемы, текст следует помещать на отдельном листе и давать перекрестную ссылку на символ.**

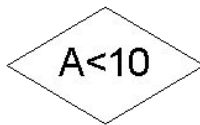
# Символы процесса



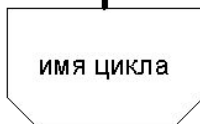
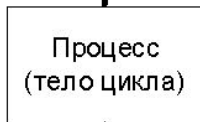
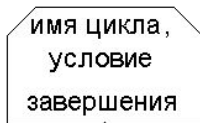
**Процесс** - отображение функций обработки данных



**Предопределенный процесс** (определение группы операций, которые определены в другом месте, например в подпрограмме)



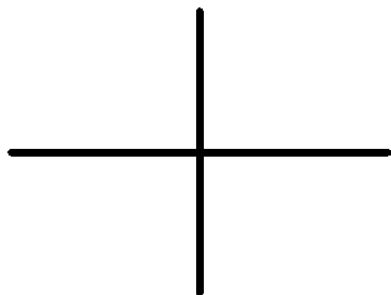
**Решение** (отображение функции, имеющих один вход и ряд альтернативных выходов, из которых только один может быть активизирован после анализа логического условия)



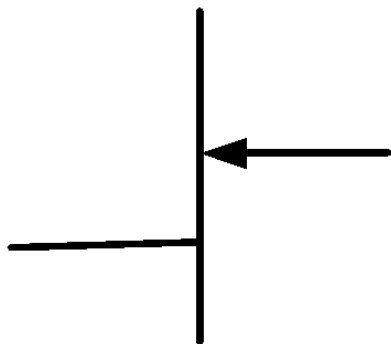
**Граница цикла** – начало и конец цикла (или наоборот), обе части символа имеют один и тот же идентификатор. Для циклов с постусловием условие завершения указывают в нижней границе.

# Символы линий – отображают поток данных или управления

**Линии** – горизонтальные или вертикальные только с прямым углом перегиба. Стрелки не ставятся, если управление идет сверху вниз или слева направо.



Пересечение линий потока управления



Объединение линий потока управления  
(места объединения смещены относительно друг друга)

- **Линии в схемах должны подходить к символу либо слева, либо сверху, а исходить либо справа, либо снизу. Линии должны быть направлены к центру символа.**
- **При необходимости линии в схемах следует разрывать для избежания излишних пересечений или слишком длинных линий, а также, если схема состоит из нескольких страниц.**
- **Соединитель в начале разрыва называется внешним соединителем, а соединитель в конце разрыва внутренним соединителем.**

# **Повторяющееся представление**

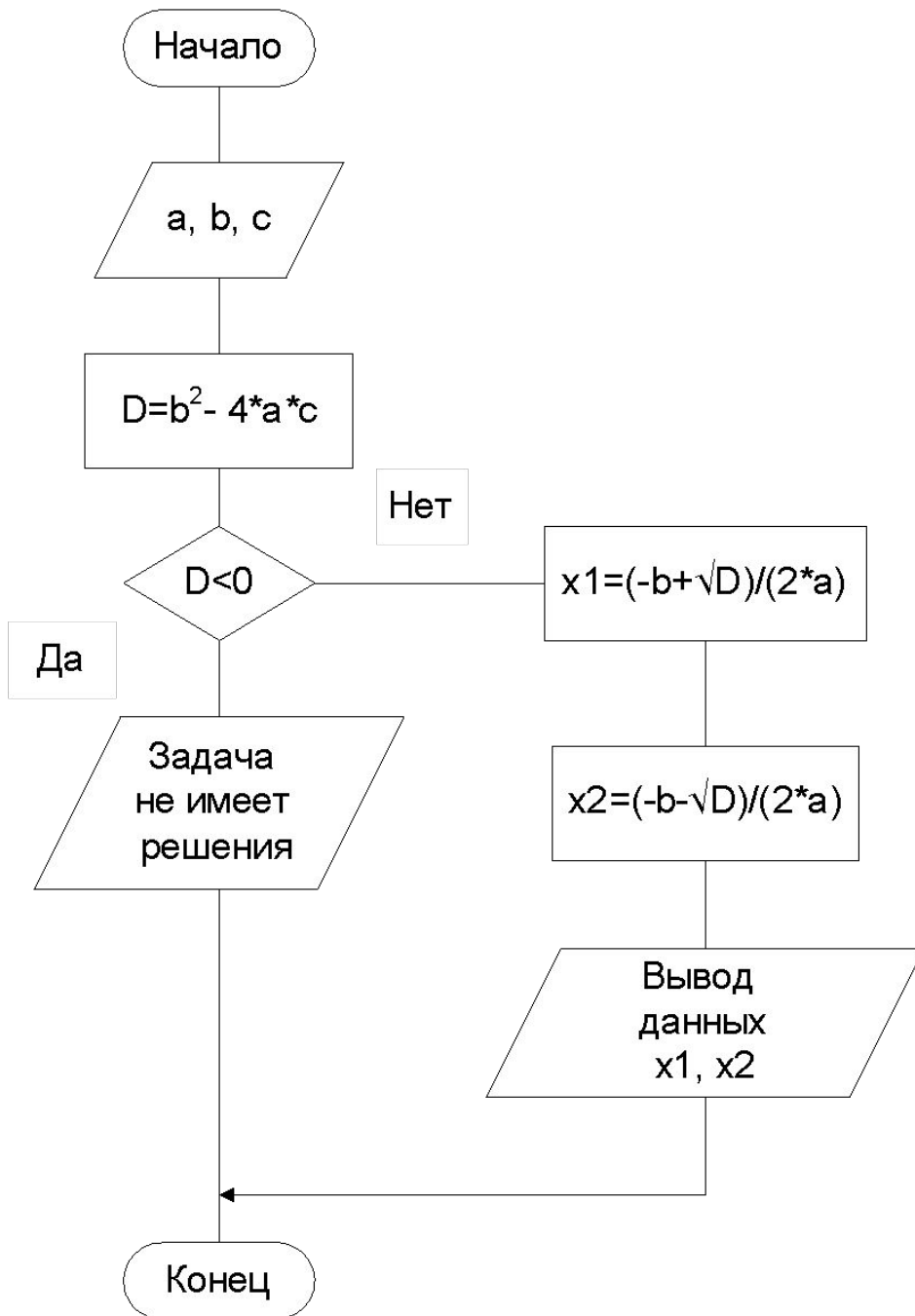
- **Вместо одного символа с соответствующим текстом могут быть использованы несколько символов с перекрытием изображения, каждый из которых содержит описательный текст (использование или формирование нескольких носителей данных или файлов, производство множества копий печатных отчетов).**
- **Когда несколько символов представляют упорядоченное множество, это упорядочение должно располагаться от переднего (первого) к заднему (последнему).**

# Словесное описание алгоритма

Рассмотрим пример: необходимо найти корни квадратного уравнения:  $a \cdot x^2 + b \cdot x + c = 0$  ( $a \neq 0$ ):

- 1) Начало
- 2) Ввод данных  $a, b, c$
- 3) Вычислить  $D = b^2 - 4 \cdot a \cdot c$
- 4) Если  $D < 0$ , то задача не имеет решения, перейти к 8
- 5)  $x_1 = (-b + \sqrt{D}) / (2 \cdot a)$
- 6)  $x_2 = (-b - \sqrt{D}) / (2 \cdot a)$
- 7) Вывод данных  $x_1, x_2$
- 8) Конец





Представим  
графическое  
описание алгоритма  
решения ранее  
представленной  
задачи

# Разновидности структур алгоритмов

- **Линейные**
- **Разветвляющиеся**
- **Циклические**

# **Линейный вычислительный процесс**

- **Линейный вычислительный процесс – это процесс, в котором направление вычислений является единственным.**

# **Разветвляющийся вычислительный процесс**

- **Разветвляющийся вычислительный процесс – это процесс, в котором направление вычислений определяется некоторыми условиями**

# **Циклический вычислительный процесс**

- **Циклический вычислительный процесс – это процесс, в котором отдельные участки вычислений выполняются многократно.**
- **Участок схемы, многократно повторяемый в ходе вычислений, называется циклом. При повторениях обычно используются новые значения исходных данных.**

## **В соответствии с взаимным расположением циклов :**

- *Простые* – циклы, не содержащие внутри себя другие циклы
- *Сложные* - циклы, содержащие внутри себя другие циклы
- *Вложенные* (внутренние) – циклы, входящие в состав других циклов (цикл в цикле)
- *Внешние* – циклы, не являющиеся составной частью других циклов, но содержащие в своем составе внутренние циклы.

# В зависимости от месторасположения условия выполнения цикла

- *Циклы с предусловием*
- *Циклы с постусловием*

# В соответствии с видом условия выполнения

- *Циклы с параметром*
- *Итерационные циклы*



# Структурное программирование

- При создании средних по размеру приложений (несколько тысяч строк исходного кода) используется *структурное программирование*, идея которого заключается в том, что структура программы должна отражать структуру решаемой задачи, чтобы алгоритм решения был ясно виден из исходного текста.
- 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.

- Идеи структурного программирования появились в начале *70-годов* в компании IBM, в их разработке участвовали известные ученые:
  - *Э. Дейкстра,*
  - *Х. Милс,*
  - *Э. Кнут,*
  - *С. Хоор.*

# **Заповеди структурного программирования**

- 1. нисходящее проектирование;**
- 2. пошаговое проектирование;**
- 3. структурное проектирование  
(программирование без goto);**
- 4. одновременное проектирование алгоритма и  
данных;**
- 5. модульное проектирование;**
- 6. модульное, нисходящее, пошаговое  
тестирование.**

Структурное программирование основано на модульной структуре программного продукта и типовых *управляющих структурах* алгоритмов обработки данных различных программных модулей.

# Типы управляющих структур:

- **последовательность;**
- **альтернатива (условие выбора);**
- **цикл.**

**две методики (стратегии) разработки программ, относящиеся к структурному программированию:**

- программирование «сверху вниз»;**
- программирование «снизу вверх».**

# **Программирование «сверху вниз», или нисходящее программирование**

**– это методика разработки программ, при которой разработка начинается с определения целей решения проблемы, после чего идет последовательная детализация, заканчивающаяся детальной программой.**

- **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.**
- **В данном случае программа конструируется иерархически - сверху вниз: от главной программы к подпрограммам самого нижнего уровня, причем на каждом уровне используются только простые последовательности инструкций, циклы и условные разветвления.**



- **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. Когда все приложение будет написано и отлажено, тогда можно приступить к реализации этой функции.**

# **Программирование «снизу вверх», или восходящее программирование**

**– это методика разработки программ, начинающаяся с разработки подпрограмм (процедур, функций), в то время когда проработка общей схемы не закончилась.**

**Такая методика является менее предпочтительной по сравнению с нисходящим программированием так как часто приводит к нежелательным результатам, переделкам и увеличению времени разработки.**

**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.**

- *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.

- *Подпрограммы* активизируются только в момент их вызова. Операторы, которые находятся внутри подпрограммы, выполняются, только если эта подпрограмма явно вызвана.
- 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.

- *Подпрограммы могут быть вложенными* – допускается вызов подпрограммы не только из главной программ, но и из любых других программ.
- В некоторых языках программирования допускается вызов подпрограммы из себя самой. Такой прием называется *рекурсией* и опасен тем, что может привести к зацикливанию – *бесконечному самовывозу*.

# **Достоинства структурного программирования:**

- повышается надежность программ (благодаря хорошему структурированию при проектировании, программа легко поддается тестированию и не создает проблем при отладке);**
- повышается эффективность программ (структурирование программы позволяет легко находить и корректировать ошибки, а отдельные подпрограммы можно переделывать (модифицировать) независимо от других);**
- уменьшается время и стоимость программной разработки;**
- улучшается читабельность программ.**

**Т. о., технология структурного программирования при разработке серьезных программных комплексов, основана на следующих принципах:**

- программирование должно осуществляться сверху вниз;**
- весь проект должен быть разбит на модули (подпрограммы) с одним входом и одним выходом;**
- подпрограмма должна допускать только три основные структуры – последовательное выполнение, ветвление (if, case) и повторение (for, while, repeat).**



**Т. о., технология структурного  
программирования при разработке серьезных  
программных комплексов, основана на  
следующих принципах:**

- недопустим оператор передачи управления в любую точку программы (goto);**
- документация должна создаваться одновременно с программированием в виде комментариев к программе.**

**Структурное программирование эффективно используется для решения различных математических задач, имеющих алгоритмический характер.**