Компьютерные информационные технологии

Системы программирования

Кафедра информационных технологий

Дунько Элеонора Михайловна,

доцент, кандидат экономических наук

Вопросы для изучения

1. Этапы решения задач на компьютере
2. Понятие алгоритма
3. Инструментальные средства программирования
4. Технологии программирования
5. Программирование в среде приложений Microsoft Office (*сам*)


Программное обеспечение
Системное программное обеспечение
Прикладное программное обеспечение
Инструментальное программное обеспечение
Операционные системы
Сервисные системы
Системы технического обслуживания
Программные оболочки и среды
Утилиты
Прикладные программы пользователя
Пакеты прикладных программ (ППП)
Общего назначения
Методо-ориентированные
Проблемно-ориентированные
Интегрированные
Системы программирования
Инструментальные среды
Системы моделирования

1. Этапы решения задач на компьютере

1. Постановка задачи
2. Выбор метода решения и построение математической модели решения задачи
3. Разработка алгоритма решения задачи
4. Программирование
5. Отладка
6. Тестирование
7. Решение задачи на ЭВМ
8. Анализ полученного результата

Жизненный цикл программного продукта

1. Маркетинг рынка программных средств, спецификация требований к программному продукту.
2. Проектирование структуры программного продукта.
3. Программирование, тестирование, автономная и комплексная отладка программы.
4. Документирование программного продукта, подготовка эксплуатационной и технологической документации.
5. Выход на рынок программных средств, распространение программного продукта.
6. Эксплуатация программного продукта пользователями.
7. Сопровождение программного продукта.
8. Снятие программного продукта с продажи, отказ от сопровождения, вывод из эксплуатации.

2. Понятие алгоритма

Первоначально слово "алгоритм" в честь Аль Хорезми (узбекский математик, астроном, IX в.) звучало как «алгоризм»: система правил, с помощью которой можно получить решение задачи за конечное число шагов.

Например, правило определения остатка материала на конец месяца гласит, что необходимо взять величину остатка материала на начало данного месяца (входящий остаток), прибавить к нему количество поступившего на склад за месяц материала и вычесть количество отпущенного складом в этом же месяце материала.

Указанное правило можно записать с помощью символических обозначений так:

$$И = (В + П) – Р,$$

где *И* – остаток на конец месяца;
В – входящий остаток;
П – поступление на склад;
Р – отпуск (расход) со склада.

2. Понятие алгоритма

Начиная с 1974 года, в СССР был введен государственный стандарт (ГОСТ 19.004–80) определения алгоритма:

алгоритм *– точное предписание, определяющее вычислительный процесс, ведущий от начальных данных к искомому результату.*

Реализация алгоритма тесно связана с умением применять его к конкретным исходным данным. Такой процесс называется *алгоритмическим процессом*. Он заключается в переработке данных по правилам, которые указываются алгоритмом.

При решении практических задач *процесс алгоритмизации* сводится к следующему:

- выделение автономных участков;
- определение порядка выполнения выделенных участков;
- запись содержания каждого участка;
- проверка правильности выбранного алгоритма.

2.1 Свойства алгоритма

Конечность (финитность) означает, что алгоритм должен заканчиваться после конечного числа шагов.
Определенность (детерминированность) состоит в четкости образующих алгоритм указаний, их полной понятности и полной однозначности, не допускающей никакого произвола.
Массовость – это возможность применять алгоритм к решению всех задач одного типа, которые отличаются друг от друга различными исходными данными.
Результативность заключается в том, что определяемый алгоритмом процесс ведет от исходных данных к результату, который является именно решением задачи.
Дискретность, т.е. возможность разбиения алгоритмического процесса на отдельные этапы, этих этапов — на отдельные шаги и т.д. вплоть до элементарных операций, понятных для исполнения человеку или машине.

2.2 Алгоритм для ЭВМ

Алгоритм, записанный в форме, воспринимаемой вычислительной машиной, называется **программой** (ГОСТ 19.004–80).

Оператор – это совокупность символов, указывающих операцию и значение или местонахождение ее операндов (ГОСТ 19.004–80).

Операнд – это объект (данное), над которым выполняет операцию команда вычислительной машины (ГОСТ 19.004–80). **Машинная команда** – это оператор, опознаваемый и выполняемый техническими средствами вычислительной машины (ГОСТ 19.004–80).

Процесс составления программы называется **программированием.**

Программирование – это также раздел прикладной математики, разрабатывающий методы использования вычислительных машин для реализации алгоритмов (ГОСТ

2.2 Алгоритм для ЭВМ

С точки зрения применения ЭВМ, ***алгоритм*** *– это правило, указывающее действия, в результате цепочки которых от исходных данных мы приходим в искомому результату.*

Если алгоритм сводит решение поставленной задачи к арифметическим действиям над числами, то такой алгоритм принято называть ***численным алгоритмом***.

Если предписание о способе действия относится к логическим условиям, то такие алгоритмы называют ***логическими***, например, алгоритм поиска путей в конечном лабиринте, игры в шахматы, перевода.

На практике, особенно при решении экономических задач, численные алгоритмы переплетаются с логическими.

2.3 Способы записи алгоритмов

1. ***Словесная форма записи*** *с помощью общепринятых изобразительных средств: алгебраических символов и словесного текста.*

Преимущества: позволяет каждому составителю использовать привычные для него обозначения; пояснениями можно отметить с любой степенью детализации различные особенности алгоритма.

Недостатки: не является строго формальным; описание слишком длинное, громоздкое и не наглядное.

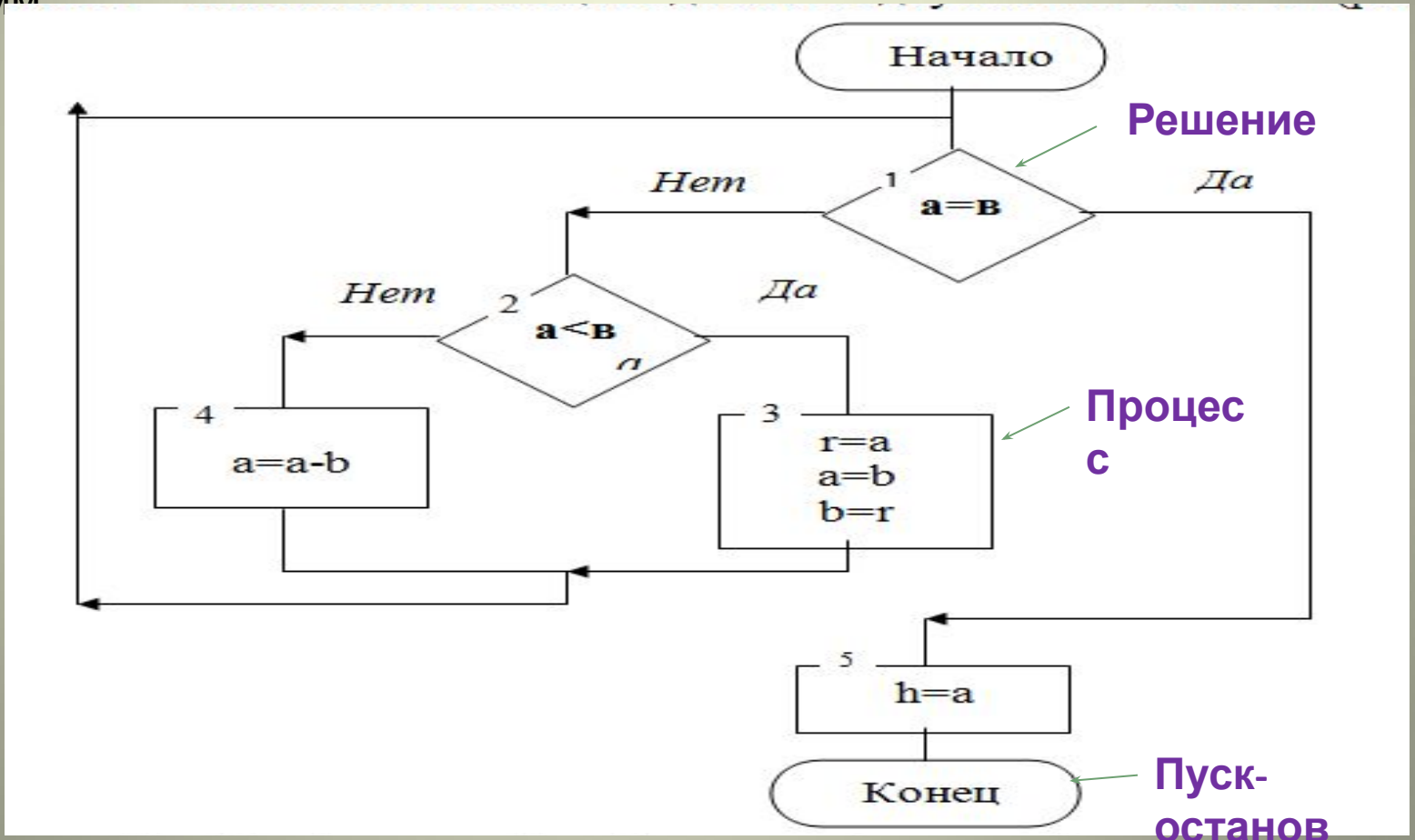
2. ***Блок-схема*** *- графическое изображение логической структуры алгоритма, в котором каждый этап процесса переработки данных представляется в виде геометрических фигур (блоков).*

3. ***Языки программирования*** *– это языки, предназначенные для записи программ и данных* (ГОСТ 19781–74).

2.3 Способы записи алгоритмов

Условные графические изображения, используемые для составления блок-схем, называют ***символами****.*

Перечень символов, их наименование, отображаемые ими функции, форма и размеры определяет ГОСТ 19428–74 "Обработка данных и программирование. Схемы алгоритмов и программ" и ГОСТ 19.003–80 "Схемы алгоритмов и программ"

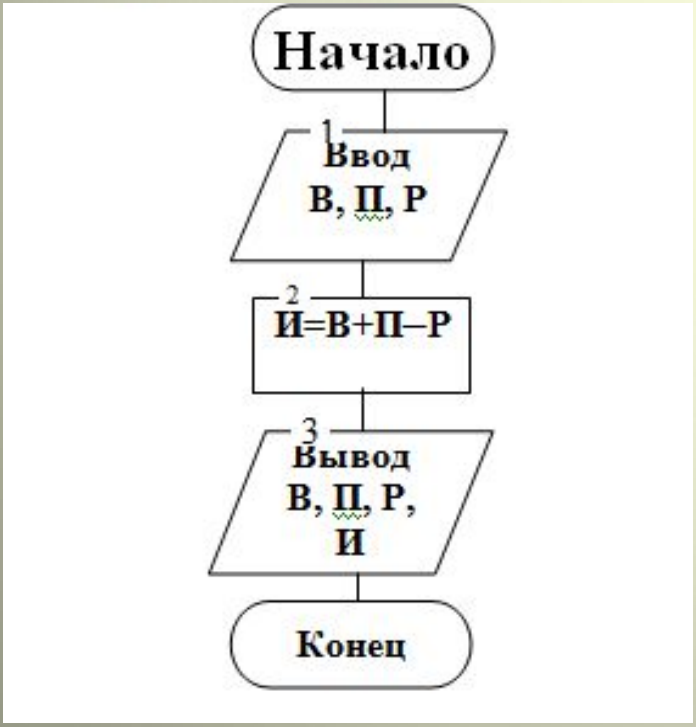


2.4 Правила построения блок-схем

- для связи блоков использовать только вертикальные и горизонтальные линии;
- обязательно снабжать стрелками те линии, которые направлены вверх или влево; на других линиях стрелки не обязательны;
- помнить, что линии не должны пересекаться; во избежание пересечений использовать символы-соединители;
- при разветвлении не забывать писать слова “Да” и “Нет”, чтобы показать, в каком случае выбирается то или иное условие;
- всякая линия должна быть направлена к какому-либо блоку;
- из всех символов, кроме ромба, должна выходить только одна линия;
- не нужна излишняя детализация, а следует лишь

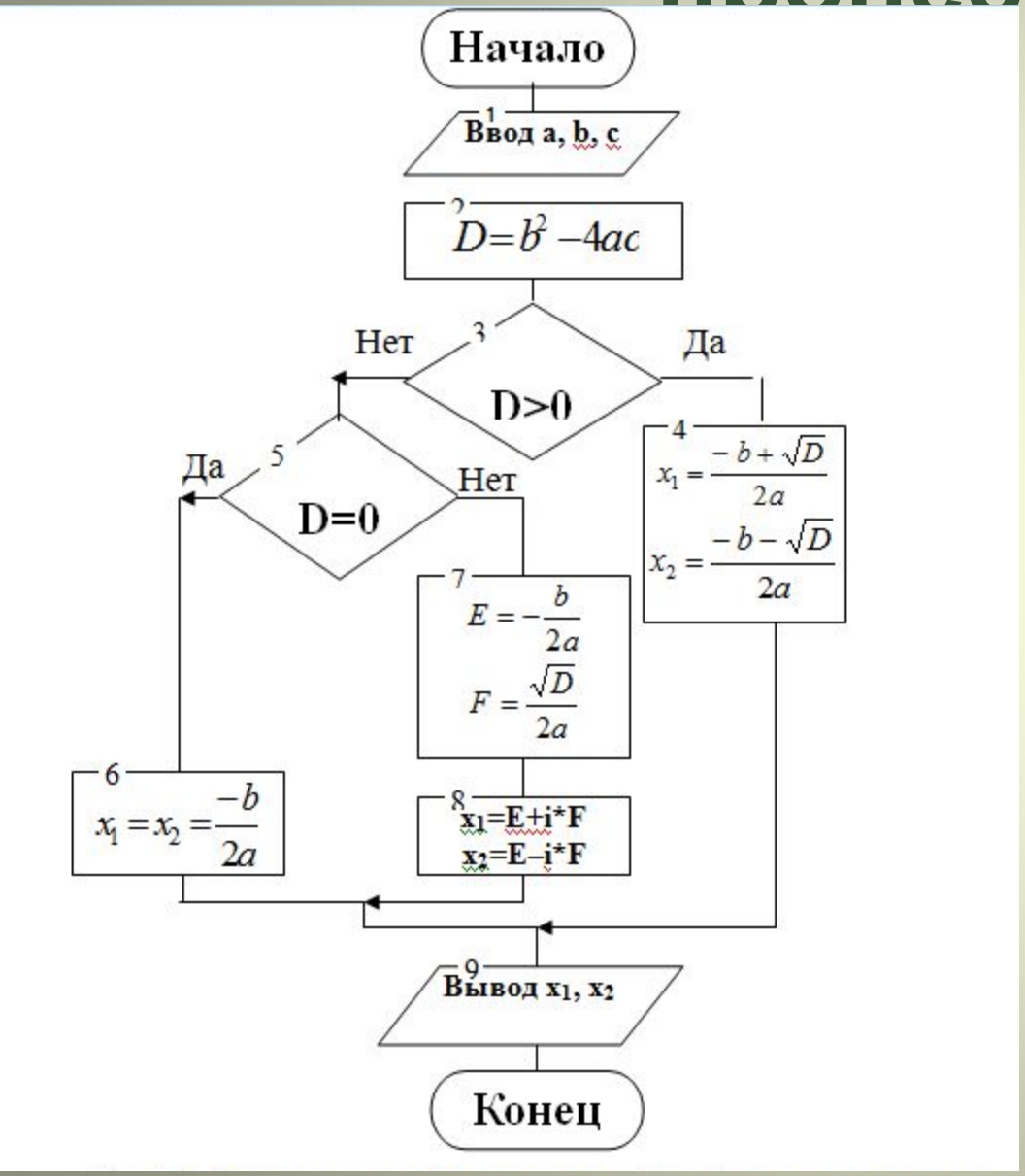
2.5 Типы алгоритмических процессов

- ***Линейные процессы:*** вычисление алгебраического выражения, образующегося при помощи четырех арифметических действий. Составление алгоритма здесь состоит в определенной последовательности выполнения арифметических операций.



Линейный *характер следования характеризуется тем, что после к-го оператора выполняется (к+1)-й, затем (к+2)-й и т.д. с шагом, равным +1, пока не встретится оператор конца, после которого все вычисления прекращаются.*
Для линейных процессов характерно, что направление вычислений не зависит от значения исходных данных и получаемых в результате решения задачи промежуточных результатов.

2.5 Типы алгоритмических процессов

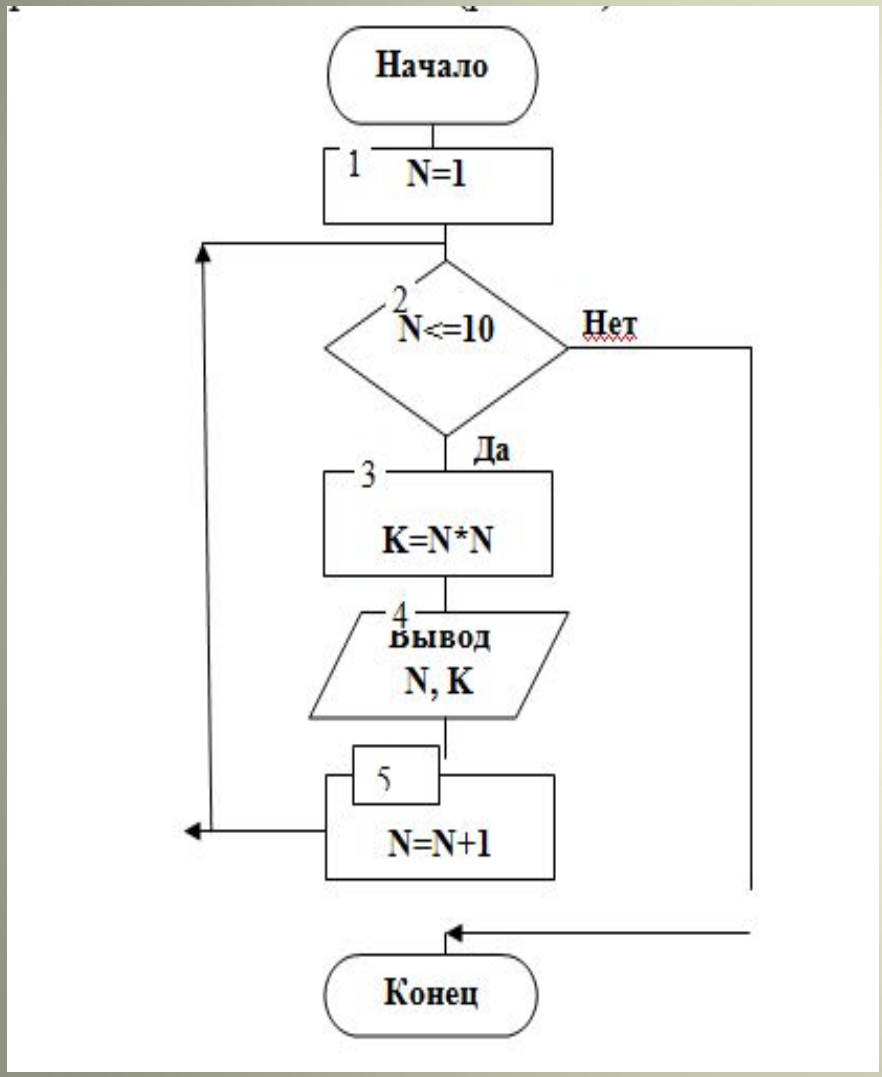


Ветвящиеся процессы.

Вычислительные процессы, в которых в зависимости от значения некоторого признака проводятся вычисления по одному из нескольких возможных направлений, называются **ветвящимися** (разветвляющимися).

Ветвящийся процесс называется **простым**, если в нем имеется две ветви, и **сложным**, если процесс разветвляется более чем на две ветви.

2.5 Типы алгоритмических процессов



Циклические процессы. В процессе решения реальных задач часто некоторые участки в них повторяются несколько раз. *Такой многократно повторяющийся участок принято называть* ***циклом.*** *Если внутри цикла содержатся другие циклы, то такой цикл называется* ***кратным,*** *или сложным, в противном случае –* ***простым.***

В зависимости от расположения проверки окончания цикла циклические процессы делят на ***циклы с предусловием*** *и* ***циклы с постусловием.***

3. Инструментальные средства программирования

Системой программирования называется комплекс программ, предназначенный для автоматизации программирования задач на ЭВМ.

Система программирования **освобождает** проблемного пользователя или прикладного программиста **от необходимости написания программ решения своих задач** на неудобном для него языке машинных команд и предоставляют им возможность использовать специальные языки более высокого уровня.

Для каждого из таких языков, называемых входными или исходными, система программирования имеет программу, осуществляющую автоматический перевод (трансляцию) текстов программы с входного языка на язык машины.

3.1 Функции систем программирования

- **контроль правильности записи программ** на входных языках и выдачу информации о наличии, месте и характере ошибок;
- **общее распределение памяти и описание глобальных переменных**, используемых многими подпрограммами исполняемой задачи;
- **трансляция отдельных частей или всей программы**, написанной на входном языке, в промежуточный язык низкого уровня или машинные коды;
- **автоматическая стыковка подпрограмм** внутри отдельно протранслированных частей общей программы;
- **выпуск сопровождающей технической документации**: распечаток программ на входном и машинном языках, сведений о распределении памяти и др.

Реализацию этих функций в системах программирования осуществляю следующие средства:

- Языки программирования;
- Трансляторы;
- Отладчики;
- Библиотеки стандартных модулей (подпрограмм).

3.2 Языки программирования и их классификация

- ***Язык программирования*** *– это набор символов (цифр, букв, специальных знаков) и система правил образования (синтаксис) и правил истолкования (семантика) конструкций из этих символов, с помощью которых описывается алгоритм решения задачи.*

Классификация ЯП **по степени зависимости от ЭВМ:**

3.2 Языки программирования и их классификация

Машинно-зависимые языки относятся к языкам низкого уровня, то есть они ближе всего к компьютеру:

Машинные – *это внутренние языки конкретных машин*, их еще называют машинными кодами. То есть если семантика языка программирования реализована в конкретной машине, то такой язык называется машинным.

Машинно-ориентированные языки *связаны с определенной машиной и отражают ее структуру.* Операторы этих языков – это те же машинные команды, но записанные мнемоническими кодами, а в качестве операндов используются не конкретные адреса, а символические имена.

Достоинства: написанные на них программы занимают меньше места в памяти и работают быстрее.

Недостаток – в их оптимизации под аппаратную архитектуру конкретного компьютера и отсутствии стандартизации в технологии написания на них программ.

Примеры: *Ассемблер, Fortran, Cobol, Lisp, Prolog, Лого* и др.

3.2 Языки программирования и их классификация

Машинно-независимые языки ориентированы не на систему команд машины, а на систему операторов, характерных для записи определенного класса алгоритмов.

Машинно-независимые языки относятся к языкам высокого уровня, их называют еще алгоритмическими языками.

Под алгоритмическим языком *понимается связанная синтаксической структурой система обозначений и терминов, содержащая сведения о том, какие действия, над какими данными, и в какой последовательности надо выполнить, чтобы решить поставленную задачу.*

Достоинства :

- независимость от аппаратного обеспечения;
- высокая эффективность труда разработчиков программ за счет укрупнения команд (одна команда на этом языке равняется нескольким командам на языке низкого уровня);
- возможность широкого использования ранее написанных программ.

Примеры: *Бейсик, Си, Паскаль, APL, Алгол*-68 и др.

3.2 Языки программирования и их классификация

Машинно-независимые языки делятся на:

- ***Процедурно-ориентированные языки*** *предназначены для описания различных классов алгоритмов с помощью стандартного набора процедур*: Бейсик, Си, Паскаль и др.

  Сегодня автоматизация программирования привела к развитию ***объектно-ориентированных языков,*** *нацеленных на работу с объектами в зависимости от происходящих событий: Delphi, Java, VBA* и др.

- ***Проблемно-ориентированные языки*** *предназначены для описания не алгоритмов, а задач в терминах их предметной области и и*спользуются для записи задач в терминологии потребителя, поэтому алфавит этих языков – это термины тех отраслей науки и техники, для которых составляется программа**:** генераторы отчетов: *APT, STRESS* (США) и др.

3.2 Языки программирования и их классификация

Классификация ЯП **по уровням представления**:

- ***Эталонный язык*** *является основой ЯП.* Он используется для создания различных вариантов языка.
- *Если эталонному языку придается вид, пригодный для машинного применения, то он называется* ***Языком конкретных представлений.***
- ***Язык для публикаций*** *служит для распространения программ среди пользователей*. Язык публикаций является производным от эталонного языка и отличается от него некоторыми видоизменениями, связанными с удобством печати алгоритмов.

3.2 Языки программирования и их классификация

Классификация ЯП **по области применения**:

- Языки для описания вычислительных задач;
- Языки для описания экономических задач;
- Языки символьного преобразования;
- Языки отладки;
- Языки параллельного программирования;
- Языки нейро-лингвистического программирования.

3.3 Основные понятия языка программирования

Язык программирования имеет иерархическую структуру.

Обычно в нем выделяют четыре уровня:

1. **Основные символы** (алфавит)
2. **Слова**
3. **Выражения**
4. **Предложения** (*операторы*)

3.3 Основные понятия языка программирования

1. Основные символы – *неделимые знаки*, с помощью которых создаются сложные образования.

2. Слова – *сочетания символов* алфавита, имеющие в языке определенный смысл.

3. Выражения – *сочетания групп слов*.

4. Оператор (предложение) задает описание некоторой части вычислительного процесса.

3.3 Основные понятия языка программирования

1. Основные символы – *неделимые знаки*, с помощью которых создаются сложные образования.

2. Слова – *сочетания символов* алфавита, имеющие в языке определенный смысл.

3. Выражения – *сочетания групп слов*.

4. Оператор (предложение) задает описание некоторой части вычислительного процесса.

- оператор присваивания,
- условные операторы,
- операторы циклов.

Примеры в Бейсике:

```
IF   A=B   THEN  Y=A^2   ELSE   Y=A*X1
     FOR     I=0     TO   10    STEP   2
```

3.4 Трансляторы

- Перевод исходной программы на машинный язык совершается автоматически с помощью специальной программы ***транслятора***.
- В процессе трансляции происходит и выявление некоторых ошибок.
- Выдаваемые транслятором данные составляют выходную, или объектную программу (*объектный модуль*).
- В зависимости от функционального назначения транслятор может быть либо компилятором, либо интерпретатором.

Компилятор *- транслирующая программа, обеспечивающая перевод с языка программирования на машинный язык без одновременного выполнения получаемой программы.*

3.5 Отладчики

После написания программы на любом языке необходимо провести ее отладку.

Частично автоматизировать этап отладки можно. В этом случае можно выдавать на печать обширную информацию о работе программы.

Выдачу нужной для обнаружения ошибок информации можно организовать при помощи специальных отладочных программ – **отладчиков**.

Сегодня существует набор отладочных программ различного функционального назначения.

3.6 Библиотеки модулей (подпрограмм)

Методология программирования совершенствуется: происходит переход от разработки языков системного уровня: Бейсик, Паскаль и др., – к *языкам описания сценариев*: Perry Tel и др.

Языки системного программирования позволяют разрабатывать программы «с нуля».

***Языки описания сценариев* позволяют связывать готовые программы**

4. Технологии программирования

Технология (от греческого: *techne* – искусство, + *logos* мастерство, учение) – совокупность знаний о методах:

- обработки,
- изготовления,
- изменения состояния, свойств, формы сырья, материала или полуфабриката,

– в процессе производства.

Начало технологии программирования положено в середине 20-го века.

4. Технологии программирования

Начало технологии программирования положено в середине 20-го века.

Возникновение этого направления обусловлено рядом причин:

- **Увеличение** *парка компьютеров* **и** *объема ПО*.
- *Программы большого объема* **и сложной структуры трудно читаемы и понимаемы.**
- *Рост стоимости разработки ПО*.
- **Специфические** *особенности процесса программирования*.

4. Технологии программирования

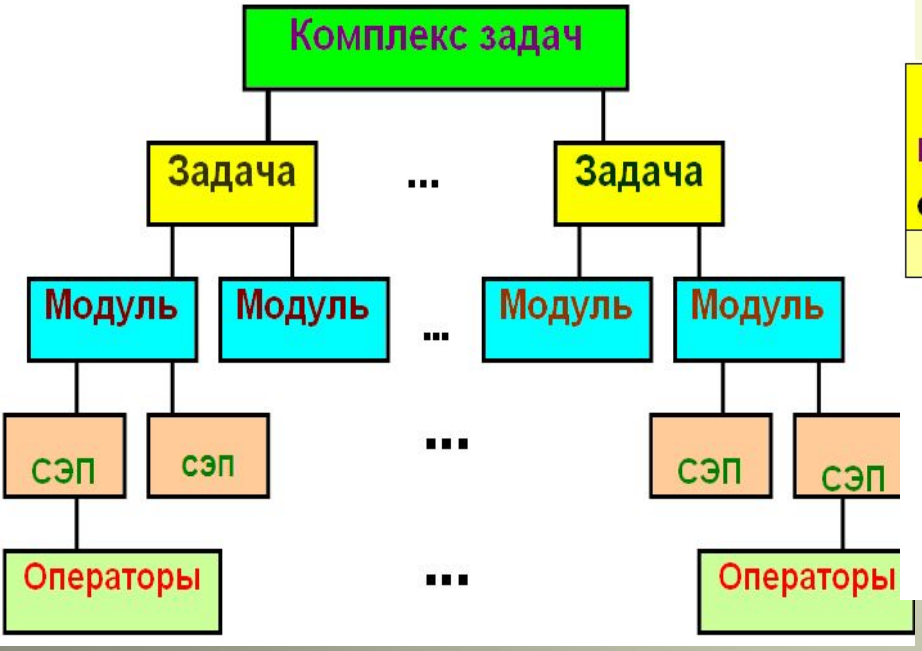
Организация программирования *в соответствии с принципами индустриальных методов* **получила название структурного подхода.**

Структурный подход нацелен на:
1. *Упрощение и унификацию структуры ПО* **– структуризация ПО;**
2. *Унификацию* **и стандартизацию** *технической документации* **на ПО;**
3. *Организацию* **в соответствии со стандартными правилами** *процессов разработки ПО.*

4. Технологии программирования

1. Важным в процессе упрощения и унификации структуры ПО является *упорядочение сверху вниз.*

Что предполагает начинать разработку с верхнего уровня иерархии – нисходящая обработка.



Унификация внутренней структуры блоков (узлов) предполагает стандартное построение тела каждого узла.

Рекомендуется использовать три структуры:

- ***Линейную*** – **простую последовательность;**
- ***Распределительную*** – **выбор, или ветвление;**
- ***Циклическую*** – **повторение.**

4. Технологии программирования

2. Унификация и стандартизация технической документации на ПО достигаются внедрением *стандартной формы спецификации ПО* – единой системы документов, ведущихся в процессе разработки, эксплуатации и модификации ПО.

4. Технологии программирования

3. Организация процесса разработки и внедрения ПО предполагает соблюдение основных правил:

- разработку ПО и его элементов осуществлять *в последовательности сверху вниз;*
- для выполнения работ создавать *специализированные группы стандартного состава* с распределением функций между членами группы;
- *техническая документация должна разрабатываться и корректироваться параллельно* с разработкой и корректировкой элементов ПО.

4. Технологии программирования

1. Метод восходящего проектирования
2. Восходящее программирование
3. Метод расширения ядра
4. Компьютерный дарвинизм
5. Объектно-ориентированное программирование
6. Сборочное программирование
7. Императивное программирование
8. Модульное сборочное программирование
9. Аспектно-ориентированное сборочное программирование
10. Объектно-ориентированное сборочное программирование
11. Компонентное сборочное программирование
12. Логическое программирование
13. Синтезирующее программирование
14. Диаграмма функционального моделирования

4. Технологии программирования

1. Метод восходящего проектирования – вначале определяются вспомогательные модули, которые потребуются для проектируемой программы.

2. Восходящее программирование (Программирование "снизу вверх") – методика разработки программ, когда крупные блоки собираются из ранее созданных мелких блоков.

Восходящее программирование начинается с разработки ключевых процедур и подпрограмм, которые постоянно модифицируются.

4. Технологии программирования

3. Метод расширения ядра – метод восходящего программирования, при котором внимание уделяется выявлению множества вспомогательных модулей, а не определению функции всей программы.

4. Компьютерный дарвинизм – подход, основанный на принципе восходящей разработки при интенсивном тестировании.

Подход состоит из трех основных процессов: макетирования, тестирования и отладки.

4. Технологии программирования

5. Объектно-ориентированное программирование – программа рассматривается как набор дискретных объектов, содержащих наборы структур данных и процедур, взаимодействующих с другими объектами.

6. Сборочное программирование – программа собирается посредством повторного использования известных фрагментов программ.

4. Технологии программирования

7. Императивное программирование – характеризуется принципом последовательного изменения состояния вычислителя пошаговым образом.

8. Модульное сборочное программирование – разновидность сборочного программирования, основанная на процедурах и функциях методологии структурного императивного программирования.

4. Технологии программирования

9. Аспектно-ориентированное сборочное программирование – разновидность сборочного программирования, основанная на сборке полнофункциональных приложений из многоаспектных компонентов, инкапсулирующих различные варианты реализации.

10. Объектно-ориентированное сборочное программирование – разновидность сборочного программирования:

- основанная на методологии объектно-ориентированного программирования и
- предполагающая распространение библиотек классов в виде исходного кода или упаковку классов в динамически компонуемую библиотеку.

4. Технологии программирования

11. Компонентное сборочное программирование – объектно-ориентированное сборочное программирование, основанное на распространении классов в бинарном виде и предоставление доступа к методам класса через строго определенные интерфейсы.

Компонентное сборочное программирование поддерживают технологические подходы **COM**, **CORBA**, **Net**.

4. Технологии программирования

12. Логическое программирование – программирование в терминах фактов и правил вывода, с использованием языка, основанного на формальных исчислениях.

13. Синтезирующее программирование – программирование, предполагающее синтез программы по ее спецификации.

4. Технологии программирования

14. Диаграмма функционального моделирования (Structured analysis and design technique) – инструмент разработки функциональных спецификаций в виде:

- диаграмм,
- фрагментов текста,
- глоссария,

– связанных перекрестными ссылками.

4. Подходы к разработке программных комплексов

1. Императивный подход

Программа – неструктурный набор команд.

Примеры: Fortran, C

2. Модульный (структурный) подход

Программа – описание действий, которые надо осуществить.

При этом:

- задача разбивается на подзадачи;
- составляется структурная схема задач;
- осуществляется реализация.

Примеры: Haskell, Prolog.

4. Подходы к разработке программных комплексов

2.1. Функциональный подход

Программа – функция с одним или несколькими аргументами.

Достоинства:

+ программа моделируется путем агрегирования математических функций;

+ память компьютера распределяется автоматически.

Недостатки:

–структура программ нелинейная;

–эффективность реализации невысокая.

Примеры: LISP, SML.

2.2. Логический подход

Программа – совокупность правил, или логических высказываний.

Достоинства:

+ высокий уровень машинной независимости;

+ возможность откатов.

Недостатки:

– специфичность класса решаемых задач;

– сложность реализации для систем реального времени.

Примеры: Prolog, Mercury.

4. Подходы к разработке программных комплексов

3. Объектно-ориентированный подход

Программа – описание объектов, их свойств и методов их обработки.

Достоинства:

+ близость к предметной области;

+ поддержка механизмов изменения объектов;

+ использование библиотек объектов и методов.

Недостатки:

– трудности формализации объектов;

– трудности тестирования программы.

Примеры: C++, C#; Visual Basic, Eiffel, Oberon.

4. Подходы к разработке программных комплексов

4. Подход сценариев (скриптов):

Программа – совокупность возможных сценариев обработки данных.

Достоинства:

+ интуитивная ясность описаний;
 близость к предметной области;
 высокая степень абстракции.

Недостаток – сложность тестирования и верификации программ.

Примеры: MS Visual Studio.NET, VBScript, PowerScript, LotusScript, JavaScript.

5. Подход поддержки параллельных вычислений

Программа – совокупность описаний процессов, которые могут выполняться в действительности псевдопараллельно.

Достоинства:

 применяются в системах реального времени;
 обрабатывают большие массивы информации, поступающей от одновременно работающих пользователей.

Недостаток – высокая стоимость разработки ПО.

Примеры: Ada, Modula-2, Oz.

Спасибо за внимание!