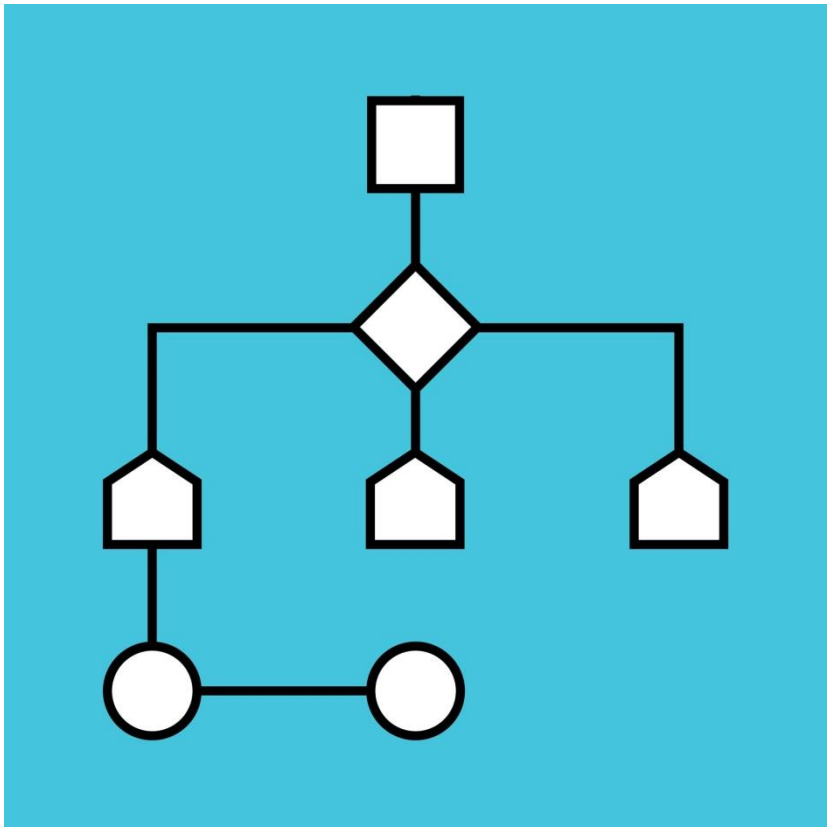
Частное профессиональное образовательное учреждение
«СЕВЕРО-КАВКАЗСКИЙ КОЛЛЕДЖ ИННОВАЦИОННЫХ ТЕХНОЛОГИЙ»
(ЧПОУ «СККИТ»)

ОСНОВЫ АЛГОРИТМИЗАЦИИ И ПРОГРАММИРОВАНИЯ

КУРС ЛЕКЦИЙ

09.02.06 Сетевое и системное администрирование



Пятигорск

Тема 1. Основы алгоритмизации, языки и системы программирования

1. Понятие алгоритма

Алгоритм – это точное и понятное предписание исполнителю совершить последовательность действий, направленных на решение поставленной задачи. Название "*алгоритм*" произошло от латинской формы имени среднеазиатского математика аль-Хорезми - Algorithmi.

Исполнитель алгоритма - это некоторая абстрактная или реальная (техническая, биологическая или биотехническая) система, способная выполнить действия, предписываемые алгоритмом. Исполнителя характеризуют: среда, элементарные действия, система команд, отказы. *Среда* (или обстановка) - это "место обитания" исполнителя. Каждый исполнитель может выполнять команды только из некоторого строго заданного списка - *системы команд* исполнителя. Для каждой команды должны быть заданы условия применимости (в каких состояниях среды может быть выполнена команда) и описаны результаты выполнения команды. После вызова команды исполнитель совершает соответствующее *элементарное действие*. *Отказы* исполнителя возникают, если команда вызывается при недопустимом для нее состоянии среды.

В информатике универсальным исполнителем алгоритмов является компьютер.

2. Свойства алгоритмов

Можно выделить следующие *основные свойства алгоритмов:*

1) *Понятность* для исполнителя - т.е. исполнитель алгоритма должен знать, как его выполнять.

2) *Дискретность* (прерывность, раздельность) - т.е. алгоритм должен представлять процесс решения задачи как последовательное выполнение простых или ранее определенных шагов.

3) *Определенность* - т.е. каждое правило алгоритма должно быть четким, однозначным и не оставлять места для разночтений.

4) *Результативность* (или конечность). Это свойство состоит в том, что алгоритм должен приводить к решению задачи за конечное число шагов.

5) *Массовость* - означает, что алгоритм решения задачи разрабатывается в общем виде, т.е. он должен быть применим для некоторого класса задач, различающихся лишь исходными данными. При этом исходные данные могут выбираться из некоторой области, которая называется *областью применимости алгоритма*.

3. Формы представления алгоритмов

Наиболее распространенными *формами представления алгоритмов* являются: словесная, графическая, псевдокоды и программная.

1) *Словесная форма* записи представляет собой описание последовательных этапов обработки данных на естественном языке (например, на русском).

Пример. Записать алгоритм нахождения наибольшего общего делителя (НОД) двух натуральных чисел.

Алгоритм: 1) задать два числа; 2) если числа равны, то взять любое из них в качестве ответа и остановиться, в противном случае продолжить выполнение алгоритма; 3) определить большее из чисел; 4) заменить большее из чисел разностью большего и меньшего из чисел; 5) повторить алгоритм с шага 2.

Описанный алгоритм применим к любым натуральным числам и должен приводить к решению поставленной задачи за конечное число шагов.

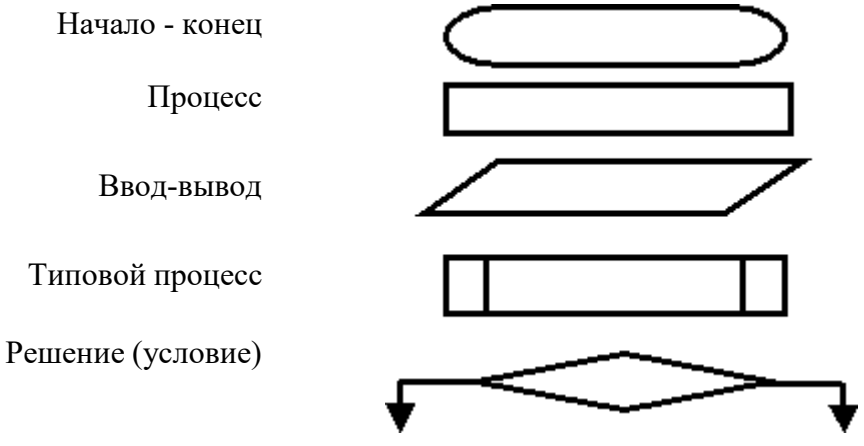
Словесный способ не имеет широкого распространения, поскольку такие описания:

а) строго не формализуемы;

б) страдают многословностью записей;

в) допускают неоднозначность толкования отдельных предписаний.

2) *Графический способ* представления алгоритмов является более компактным и наглядным по сравнению со словесным. При графическом исполнении алгоритм изображается в виде последовательности связанных между собой функциональных блоков, каждый из которых соответствует выполнению одного из действий. Такое графическое представление называется схемой алгоритма или ***блок-схемой***. В блок-схеме каждому типу действий соответствует геометрическая фигура, называемая *блочным символом.* Блочные символы соединяются *линиями переходов*, определяющими очередность выполнения действий.



Подробнее об этом ниже…

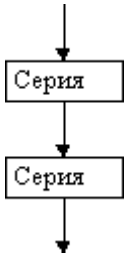
3) *Псевдокод* представляет собой систему обозначений и правил, предназначенную для единообразной записи алгоритмов. Он занимает промежуточное место между естественным и формальным языками.

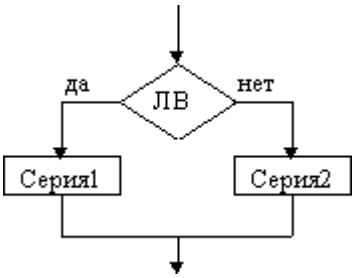
С одной стороны, он близок к обычному естественному языку, поэтому алгоритмы могут на нем записываться и читаться как обычный текст. С другой стороны, в псевдокоде используются *служебные слова* и математическая символика, что приближает запись алгоритма к общепринятой математической записи. Служебные слова выделяются в печатном тексте жирным шрифтом, а в рукописном тексте подчеркиваются для того, чтобы их можно было отличить от остального текста.

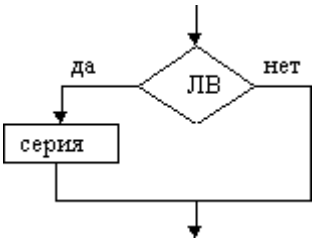
Пример. 1) задать два числа x и y; 2) ЕСЛИ x=y, ТО НОД=x и КОНЕЦ; 3) ЕСЛИ x>y, ТО x=x-y, ИНАЧЕ y=y-x; 4) ПЕРЕЙТИ в пункт 2.

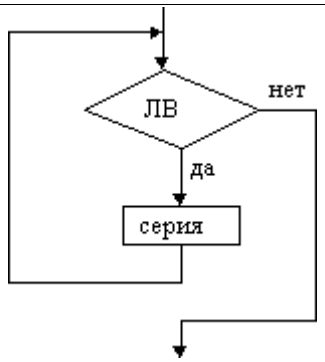
4) *Программная форма* представляет собой тексты программ, написанных на различных языках программирования.

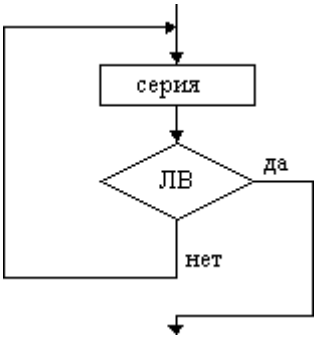
Ниже приведены графические обозначения на блок-схемах.

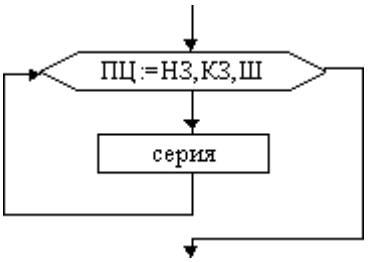
<table>
<tr>
<td> 

Структура “следование”</td>
<td> 

Полная развилка</td>
<td> 

Неполная развилка</td>
</tr>
<tr>
<td> 

Цикл с предусловие
(цикл ПОКА)</td>
<td> 

Цикл с постусловием (цикл ДО)</td>
<td> 

Цикл с параметром</td>
</tr>
</table>

На схемах СЕРИЯ обозначает один или несколько любых операторов; УСЛОВИЕ есть логическое выражение (ЛВ) (если его значение ИСТИНА, переход происходит по ветви ДА, иначе — по НЕТ). На схеме цикла с параметром использованы обозначения: ПЦ — параметр цикла, НЗ — начальное значение параметра цикла, КЗ — конечное значение параметра цикла, Ш — шаг изменения параметра цикла.

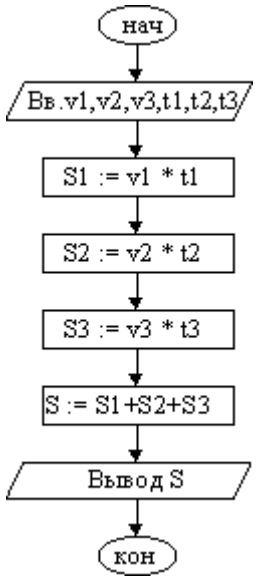
Начало и конец алгоритма на блок-схемах обозначают овалом, вводимые и выводимые переменные записываются в параллелограмме.

Виды алгоритмов

Линейные алгоритмы

Простейшие задачи имеют линейный алгоритм решения. Это означает, что он не содержит проверок условий и повторений.

Пример 1. Пешеход шел по пересеченной местности. Его скорость движения по равнине 1 км/ч, в гору —2 км/ч и под гору — 3 км/ч. Время движения соответственно 1, 2 и 3 ч. Какой путь прошел пешеход?



1. Ввести v1, v2, v3, t1, t2, t3.
2. S1 := v1 * t1.
3. S2 := v2 * t2.
4. S3 := v3 * t3.
5. S := S1 + S2 + S3.
6. Вывести значение S.
7. Конец.

Для проверки работоспособности алгоритма необходимо задать значения входных переменных, вычислить конечный результат по алгоритму и сравнить с результатом ручного счета.

Разветвленный алгоритм

1. Задачи на ветвление

Разветвляющимся называется такой **алгоритм**, в котором в зависимости от истинности или ложности заданного условия выбирается один из нескольких возможных вариантов (путей) вычислительного процесса. Каждый такой вариант называется **ветвью алгоритма**. Возможных вариантов может быть два или три.

Ветвление - такая форма организации действий, при которой в зависимости от выполнения или невыполнения некоторого условия совершается либо одна, либо другая последовательность действий.

Признаком разветвляющегося алгоритма является наличие операций проверки *условия*.

Условие – это выражение логического типа. Оно может включать в себя константы, имена переменных, арифметические операции, операции отношения, логические операции, скобки. Условие может быть истинным или ложным, то есть принимать одно из двух значений: ИСТИНА или ЛОЖЬ.

Различают два вида условий: *простые* и *составные*.

Простым условием называется выражение, составленное из двух арифметических выражений или двух текстовых величин, связанных одним из знаков: <, ≤ , >, ³, =, ¹. Такое условие часто называют операцией отношения.

Составными условиями называются условия, состоящие из нескольких простых и соединенные знаками логических операций И, ИЛИ, НЕ.

ПРИМЕРЫ составных условий.

(A > 5) И (B < 2)

(X = Y) ИЛИ (50 £ C – 1)

НЕ («собака» = «конура») И («кошка» = D)

Составное условие вида «X И Y» истинно тогда и только тогда, когда истинны оба условия X и Y, в остальных случаях – ложно.

Составное условие вида «X ИЛИ Y» истинно тогда, когда истинно хотя бы одно условие X или Y.

Условие вида «НЕ X» истинно, если X ложно, и наоборот.

Для обозначения операции проверки условия в блок-схеме алгоритма используется блок, изображаемый ромбом, внутри которого указывается проверяемое условие. Этот блок имеет один вход и два или три выхода.

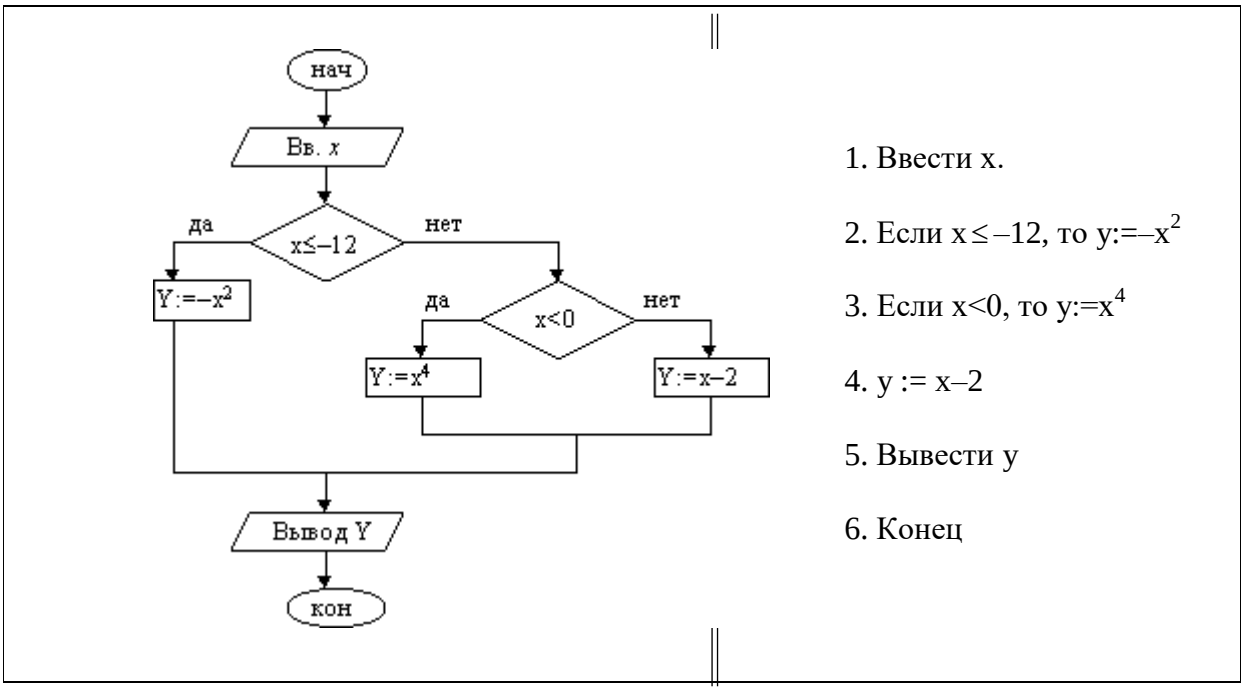
Если выходов два, то они обозначаются ДА и НЕТ. Если условие истинно, то вычислительный процесс «идет» по ветви ДА; если ложно – по ветви НЕТ.

Если выходов три (это чаще всего бывает, когда в условии переменная или выражение сравниваются с нулем), то они обозначаются: >0, <0, =0.

При написании разветвляющихся программ используется условный оператор.

Пример 1. Вычислить значение функции

$$y = \begin{cases} -x^2 \text{ при } x \le -12, \\ x^4 \text{ при } -12 < x < 0, \\ x - 2 \text{ при } x \ge 0. \end{cases}$$



При тестировании алгоритмов с развилкой необходимо подбирать такие исходные данные, чтобы можно было проверить все ветви. В приведенном выше примере должно быть по крайней мере три тестовых набора.

Циклические алгоритмы

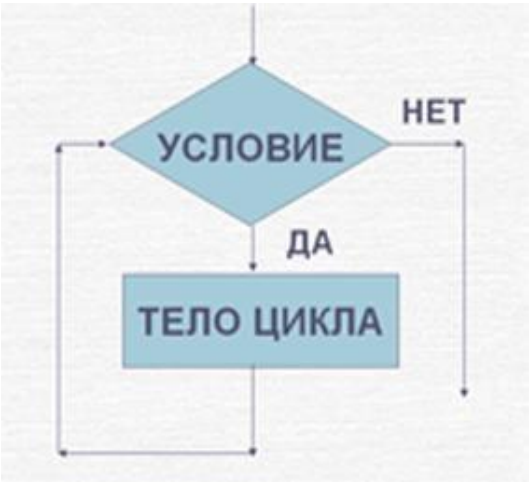
Циклическим называют алгоритм, в котором получение результата обеспечивается многократным выполнением одних и тех же операций. Иными словами, циклическим называют алгоритм, в котором определенная последовательность действий повторяется либо заданное количество раз, либо до тех пор, пока не выполнится некоторое условие. Повторяющаяся последовательность действий называется телом цикла.

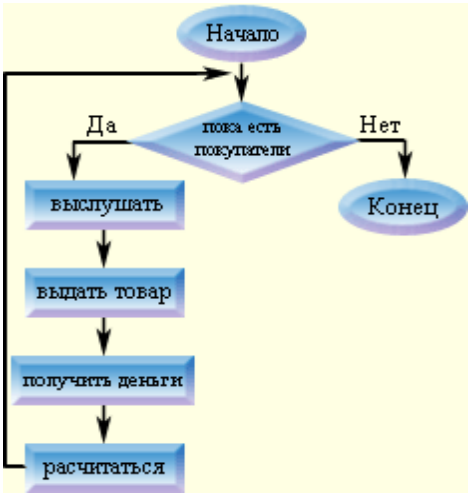
Цикл - такая форма организации действий, при которой одна и та же последовательность действий (тело цикла) совершается несколько раз (или ни разу) до тех пор, пока выполняется некоторое условие.

Цикл с предусловием (цикл-пока)

Выполнение начинается с проверки условия, поэтому этот цикл и называется циклом с предусловием. Переход к выполнению серии действий (телу цикла) осуществляется тогда, когда условие истинно, в противном случае происходит выход из цикла. Условие в данном случае называется условием продолжения цикла. Возможен такой вариант, что тело цикла не выполнится ни разу. Для того, чтобы цикл не повторялся бесконечное число раз, в теле цикла необходимо изменять значение параметра цикла так, чтобы за конечное количество «шагов» условие стало ложным.

Блок-схема **цикла с предусловием**

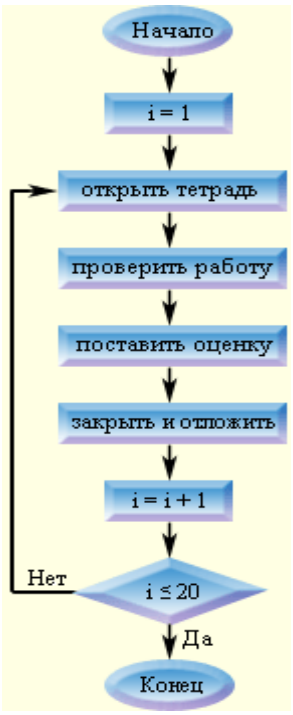




Цикл с постусловием (цикл-до)

Цикл с постусловием начинается с выполнения тела цикла и лишь затем проверяется условие. Тело цикла обязательно выполнится хотя бы один раз. Если условие ложно, то повторяется выполнение тела цикла, если истинно – то осуществляется выход из цикла. Поэтому данное условие называют условием прекращения цикла. Чтобы не было зацикливания, в теле цикла необходимо изменять значение параметра цикла так, чтобы за конечное количество «шагов» условие стало истинным.

Блок-схема **цикла с постусловием**



Если какие-либо операторы необходимо выполнить несколько раз, то их не переписывают каждый раз заново, а организуют цикл.

Итак, алгоритм – это точно определенное описание способа решения задачи в виде последовательности действий. Такое описание называется *формальным*, а процесс – *формализацией*. Для представления алгоритма в виде, понятном компьютеру, служат *языки программирования*. Изучение программирования разумно начинать собственно с разработки алгоритмов, не акцентируя первоначально внимания на записи алгоритма на том или ином языке программирования. После написания программы с помощью *трансляторов* либо переводится в *машинный код* (последовательность чисел, с которой работает процессор), либо исполняется.

Тема 2. Основные элементы языка. Управляющие операторы языка.

Структурированные типы данных. Символьные типы данных

Систему программирования *ТурбоПаскаль* называют *интегрированной средой программирования*, так как она объединяет в себе возможности: *редактора текстов, компилятора, компоновщика, отладчика.*

Процесс поиска ошибок в программе называется **тестированием**, процесс устранения ошибок - **отладкой**. С помощью языка программирования создается не готовая программа, а ее текст, описывающий разработанный алгоритм. Для получения работающей программы, необходимо перевести этот текст в машинный код (для этого служат программы - **компиляторы**) и затем использовать отдельно от исходного текста. Либо сразу выполнять команды языка**,** указанные в программе, с помощью **интерпретаторов** (**п**ри большом объеме повторяющихся вычислений программа может работать медленно). *Компиляторы* же сразу преобразуют весть текст программы (исходный код) - транслируют на машинный язык. Работают быстрее, чем интерпретаторы. В реальных системах программирования перемешаны технологии и компиляции, и интерпретации.

Разные типы процессоров имеют разные наборы команд. Если язык программирования ориентирован на конкретный тип процессора и учитывает его особенности, то он называется **языком программирования низкого уровня**. **Языком** самого **низкого уровня** является **язык ассемблера**, который представляет каждую команду машинного кода с помощью символьных условных обозначений, называемых мнемониками. **Языки** высокого **уровня** понятнее человеку. В них не учитываются особенности компьютерных архитектур.

В начале 50-х годов появился первый язык *ассемблера*. К концу 50-х годов начали появляться языки программирования более высокого уровня, такие как Lisp, Fortran, ALGOL. В них уже не было точного соответствия между языковыми конструкциями и машинными командами. Преобразование строк исходного кода в последовательности двоичных команд осуществлялось компилятором. Со временем их число пополнилось языками PL /1, Pascal, C, C++, Java. Все они менее эффективно используют аппаратуру по сравнению с языками ассемблера, но позволяет быстрее создавать приложения. В результате им удалось практически полностью вытеснить языки ассемблера при создании крупных приложении.

<table>
<tr><th colspan="4">Языки программирования высокого уровня</th></tr>
<tr><td>Неструктурные (BASIC-первый по популярности, Fortran-
1-ый компилятор)</td><td>Структурные (Pascal, Algol-комприлятор)</td><td>логические (Prolog, Simula)</td><td>функциональные (Lisp, ЛОГО)</td></tr>
<tr><td colspan="2">объектно-ориентированные (Smalltalk, C++, Object Pascal, Delphi, Java)</td><td>Языки программирования баз данных (SQL, Oracle)</td><td>Языки программирования для Интернет (HTML, Perl, Xml)</td></tr>
</table>

Тема 3. Модульное программирован. Рекурсия. Визуальнособытийно управляемое программирования . Разработка оконного приложения

Рассмотрим классификацию по стилю программирования. Стиль - совокупность правил, лежащих в основе синтаксиса и семантики языка программирования. Различают следующие стили:

1. неструктурный;
2. структурный;
3. объектно-ориентированный;
4. логический;
5. функциональный.

Неструктурное программирование допускает использование в явном виде команды безусловного перехода (в большинстве языков GOTO). Типичные представители неструктурных языков - ранние версии Бейсика и Фортрана. Данный стиль вызван особенностями выполнения машиной программы в кодах и унаследован от программ на языке ассемблера, поскольку там команда перехода является обязательной. Однако в языках высокого уровня наличие команды перехода влечет за собой массу серьезных недостатков: программа превращается в "спагетти" с бесконечными переходами вверх-вниз, ее очень трудно сопровождать и модифицировать. Фактически неструктурный стиль программирования не позволяет разрабатывать большие проекты. Ранее широко практиковавшееся первоначальное обучение программированию на базе неструктурного языка (обычно Бейсика) приводило к огромным трудностям при переходе на более современные стили.

Структурное программирование **–** процесс разработки алгоритмов с помощью блок-схем.

Подпрограмма – набор 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.

Структурный стиль был разработан в середине 60-х - начале 70-х гг. В его основе лежат две идеи: задача разбивается на большое число мелких подзадач, каждая из которых решается своей

процедурой или функцией (декомпозиция задачи). При этом проектирование программы идет по принципу сверху вниз: сначала определяются необходимые для решения программы модули, их входы и выходы, а затем уже эти модули разрабатываются. Такой подход вместе с локальными именами переменных позволяет разрабатывать проект силами большого числа программистов.

Как доказал Э. Дейкстра, любой алгоритм можно реализовать, используя лишь три управляющие конструкции: последовательное выполнение, ветвление и цикл. Данное обстоятельство позволяет при наличии соответствующих операторов исключить из языка команду безусловного перехода GOTO.

Принципы структурного программирования были реализованы в языке Алгол, но наибольшую популярность завоевал язык Паскаль, созданный в 1970 швейцарским ученым Н. Виртом. Паскаль получил широчайшее распространение и может считаться образцовым языком программирования, наиболее популярным и сейчас (например, в версии Delphi фирмы Imprise).

Логическое программирование представляет собой попытку возложить на программиста только постановку задачи, а поиски путей ее решения предоставить транслятору. Логические языки (Пролог, Симула) имеют специальные конструкции для описания объектов и связей между ними. Например, если дано:

```
БРАТЬЯ ИМЕЮТ ОДНОГО ОТЦА
ДЖОН - ОТЕЦ ДЖЕКА
МАЙК - БРАТ ДЖЕКА
```

то система логического программирования должна сделать вывод:

```
ДЖОН - ОТЕЦ МАЙКА.
```

Хотя работы по логическому программированию ведутся с 50-х гг., в настоящее время данное направление несколько потеряло свою актуальность в связи с отсутствием реальных результатов, поскольку большинство реализованных на принципах логического программирования систем оказались практически непригодными.

Объектно-ориентированное программирование

В начале 80-х годов возникло новое направление, основанное на понятии объекта. *Объект* – это совокупность *свойств* (структур данных, характеристик для этого объекта), *методов* обработки (подпрограммы изменений свойств) и *событий*, на которые данный объект может реагировать и которые приводят, как правило, к изменению свойств объекта.

Объектно-ориентированное (ОО) программирование, разработанное в середине 70-х гг. Керниганом и Риччи и реализованное в ОО-версиях языков Си и Паскаль, представляет собой отображение объектов реального мира, их свойств (атрибутов) и связей между ними при помощи специальных структур данных. Структурное программирование подразумевает наличие ряда встроенных структур данных: целых, вещественных и строковых переменных, массивов, записей -

при помощи которых и производится отображение свойств объектов реального мира. При объектно-ориентированном подходе для объекта создается своя структура данных (класс), содержащая как свойства объекта (поля), так и процедуры для управления объектом (методы).

Например, рассмотрим простейший объект - точку на экране. Она имеет как минимум три поля (координаты и цвет) и методы вроде "ChangeColor" (поменять цвет), "MoveXY" (переместиться в точку x, y) и т.д. Объектно-ориентированный подход является в настоящее время доминирующим и позволяет сократить время разработки и увеличить надежность больших проектов. Однако программы в данном стиле отличаются громоздким синтаксисом; в целом идеология объектно-ориентированного подхода весьма неочевидна, часто воспринимается с трудом (особенно это характерно для языка Си, который и в своем первоначальном виде отличается крайне неудобочитаемым синтаксисом), и переход к его использованию труден или невозможен для большого числа программистов.

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*.

Объектно-ориентированное программирование базируется на трех концепциях – *инкапсуляции, наследовании и полиморфизме.* Объединение данных с методами в одном классе (типе) называется *инкапсуляцией*. Помимо объединения, инкапсуляция позволяет ограничивать доступ к данным объектов и реализации методов классов. В результате появляется возможность использования классов в приложениях на основе только описаний этих классов.

Важная характеристика класса – возможность создания на его основе новых классов с *наследованием* всех его свойств и методов добавления собственных. Класс, не имеющий предшественника, называется *базовым*. Например, класс «животное» имеет свойства «название», «размер», методы «идти» и «размножаться». Созданный на его основе класс «кошка» наследует все эти свойства и методы, к которым добавляется свойство «окраска» и метод «пить». Наследование позволяет создавать новые классы, повторно используя уже готовый исходный код и не тратя времени на его переписывание.

В большинстве случаев методы базового класса у классов-наследников приходится переопределять – объект класса «кошка» выполняет метод «идти» совсем не так, как объект класса «амеба». Такое свойство объектов переопределять методы наследуемого класса и корректно их использовать называется *полиморфизмом*.

В основе ***функционального стиля*** лежит понятие функции как "черного ящика". *Основная идея* - задача разбивается на группы, которые дробятся на элементарные функции, определенное сочетание которых реализует решение подзадач. Процесс дробления на элементарные функции заканчивается тогда, когда каждая из полученных функций выполняет некоторую логически

отдельную операцию и результат работы данной функции может служить входом для другой функции. Язык реализации Лисп.

HTML (Hyper Text Markup Lanquage) является стандартным языком, предназначенным для создания гипертекстовых документов в среде WWW. HTML - документы могут просматриваться различными браузерами (специальными программами, интерпретирующими такого рода гипертекстовые документы), наиболее известным из которых является Internet Explorer. В отличие от документов, например текстового процессора Microsoft Word, документы в формате HTML не организованы по принципу WYSIWYG (What You See Is You Get) — что видишь, то и получишь [при выводе на печать или монитор]. Когда документ создан с использованием HTML, браузер должен интерпретировать HTML для выделения различных элементов документа и их первичной обработки с целью дальнейшего отображения в виде, задуманном автором. Данная технология не является исключительной особенностью HTML, она, например, используется в языке, предназначенном для верстки математических текстов TeX (LaTeX), автором которого является Д. Кнут. Использование HTML позволяет форматировать документы для их представления с использованием заголовков, шрифтов, линий и других стандартных элементов на любой системе, предназначенной для их просмотра.

Большинство документов имеют стандартные элементы, например, такие, как заголовки, параграфы или списки. Используя **тэги** (команды) HTML, можно обозначать данные элементы, обеспечивая браузеры информацией для их отображения, сохраняя в целом общую структуру и информационную полноту документов. В большинстве случаев (например, при использовании различных текстовых редакторов) автор строго определяет внешний вид документа. В случае HTML (основываясь на возможностях браузера) можно в определенной степени управлять внешним видом (но не его содержимым). HTML позволяет отметить, где в документе должен быть заголовок или абзац, при помощи тэгов HTML, а затем предоставляет браузеру интерпретировать эти тэги. Например, один браузер может распознавать тэг начала абзаца и представлять документ в нужном виде, а другой, не имеющий такой возможности, представит документ в одну строку. Пользователи не которых браузеров имеют также настраивать размер и вид шрифта, цвет и другие параметры, влияющие на отображение документа. Основное преимущество HTML заключается в том, что документ может быть просмотрен на браузерах различных типов и на различных платформах.

Технология подготовки и решения задач на ЭВМ

Решение задач включает следующие основные этапы, часть из которых осуществляется без участия ЭВМ.

1. ***Постановка задач***
   a. сбор информации о задаче;
   b. формулировка условия задачи;
   c. определение конечных целей;
   d. описание данных
- Выбор методов решения задач

Чтобы решить задачу необходима точная постановка задачи и правильный выбор метода решения задачи. Постановка задачи сводится, как правило, к математической форме описания условий задачи по схеме:

Задача (словесное описание).
Дано (перечисление исходного).
Требуется (перечисление требуемого).
Связь (зависимость между исходным и требуемым).
При (условия допустимости исходного).

Выбор метода решения должен обеспечить получение требуемых результатов для любых допустимых исходных данных.

2. ***Анализ и исследование задачи, модели:***
   a. анализ существующих аналогов;
   b. анализ технических и программных средств;
   c. разработка математической модели;
   d. разработка структур данных.

3. ***Построение алгоритма:***
   a. выбор формы записи алгоритма (блок-схема, табличная и др.);
   b. запись алгоритма.

4. ***Программирование:***
   a. выбор языка программирования;
   b. выбор способа представления данных;
   c. запись алгоритма на выбранном языке;
   d. выбор тестов и методов тестирования.

5. ***Тестирование и отладка:***
   a. синтаксическая отладка;
   b. отладка семантики и логической структуры;
   c. тестовые расчеты и анализ результатов тестирования;
   d. совершенствование программы.

Тестирование - процесс выполнения программ с целью обнаружения факта наличия ошибок.

Отладка программы - этап разработки компьютерной программы, в процессе которого происходят обнаружение, локализация и устранение явных ошибок в программе.

Обычно отладка выполняется на контрольных примерах с известными результатами.
При отладке важно помнить:

- лучше использовать простые тестовые данные;
- ошибки разделять и устранять поочерёдно,
- не вносить в программу сразу несколько изменений;
- не следует считать причиной ошибок транслятор.

Два этапа процесса тестирования:

- проверка в нормальных условиях;
- проверка в экстремальных условиях

Альфа-тестирование - тестирование готового программного продукта на специально созданных задачах.

Бета-тестирование (англ. beta testing) — интенсивное использование почти готовой версии продукта (как правило, программного или аппаратного обеспечения) с целью выявления максимального числа ошибок в его работе для их последующего устранения перед окончательным выходом (выпуском) продукта на рынок, к массовому потребителю.

В отличие от альфа-тестирования, проводимого силами штатных разработчиков или тестировщиков, бета-тестирование предполагает привлечение добровольцев из числа обычных будущих пользователей продукта, которым рассылается упомянутая предварительная версия продукта (так называемая бета-версия). Такими добровольцами (их называют бета-тестерами) обычно движет любопытство к новому продукту — любопытство, ради удовлетворения которого они вполне согласны мириться с возможностью испытать последствия еще не найденных (а потому и не исправленных) ошибок.

6. ***Анализ результатов решения задачи*** и уточнение в случае необходимости математической модели с повторным выполнением этапов 2 - 5.

7. ***Сопровождение программы:***
    a. доработка программы для решения конкретных задач;
    b. составление документации к решенной задаче, к математической модели, к алгоритму, к программе, к набору тестов, к использованию.

Сопровождение программного изделия - процесс модификации существующей программы для ЭВМ, обусловленный необходимостью устранения выявленных в ней ошибок и/или изменения ее функциональных возможностей.