Содержание:

Введение

Изобретение языка программирования высшего уровня позволило нам общаться с машиной, похоже, даже понимать её. Действительно, если мы обратим внимание на темпы роста и развития новейших технологий в области программирования, то можно предположить, что в ближайшем будущем, человеческие познания в этой сфере, помогут произвести на свет языки, умеющие принимать, обрабатывать и передавать информации в виде мысли, слова, звука или жеста. Такие языки могут по праву носить имя «высочайших». Возможно, концепция решения этого вопроса проста, а ближайшее будущее этого проекта уже не за горами, и в этот момент, где-нибудь горбится молодой, никем не признанный специалист и разрабатывает новейшую систему искусственного интеллекта, которая наконец-то позволит человеку, с помощью своих машинных языков, вести диалог с компьютером на «ТЫ».

Цель курсовой работы: изучение методов современного программирования. Формирование представления об основных понятиях и подходах технологии программирования, приемах обеспечения технологичности программных продуктов, разработке пользовательских интерфейсов, тестирование программных продуктов, отладке программного обеспечения.

Задачи курсовой работы:

1. Рассмотреть виды технологии программирования.
2. представить основную информацию о методах современного программирования.
3. Проанализировать преимущества/ недостатки рассмотренных методов современного программирования.

1.Понятие технологии программирования.

Технологией программирования называют совокупность методов и средств, используемых в процессе разработки программного обеспечения. Как любая другая технология, технология программирования представляет собой набор

технологических инструкций, включающих:

- указание последовательности выполнения технологических операций;
- перечисление условий, при которых выполняется та или иная операция;
- описания самих операций, где для каждой операции определены исходные данные, результаты, а также инструкции, нормативы, стандарты, критерии и методы оценки и т. п. (*рис.1)*

Рис.1. *Структура описания технологической операции*

Кроме набора операций и их последовательности, технология также определяет способ описания проектируемой системы, точнее модели, используемой на конкретном этапе разработки.

Различают технологии, используемые на конкретных этапах разработки или для решения отдельных задач этих этапов, и технологии, охватывающие несколько этапов или весь процесс разработки. В основе первых, как правило, лежит ограниченно применимый метод, позволяющий решить конкретную задачу. В основе вторых обычно лежит базовый метод или подход (парадигма), определяющий совокупность методов, используемых на разных этапах разработки, или методологию. [1]

Исторически в развитии программирования можно выделить несколько принципиально отличающихся методологий.

Изначально понятие технологии как таковой — это 60-е годы прошлого столетия — это период «стихийного» программирования. В этот период отсутствовало понятие структуры программы, типов данных и т.д. Вследствие этого код получался запутанным, противоречивым. Программирование тех лет считалось искусством. Конец 60-х — кризис в программирование.

Выход из этого кризиса — переход к структурной парадигме программирования. Структурный подход к программированию представляет собой совокупность рекомендуемых технологических приемов, охватывающих выполнение всех этапов разработки программного обеспечения. В основе структурного подхода лежит декомпозиция (разбиение на части) сложных систем с целью последующей реализации в виде отдельных небольших подпрограмм. С появлением других принципов декомпозиции (объектного, логического и т.д.) данный способ получил название процедурной декомпозиции.

Другим базовым принципом структурного программирования является использование при составлении программ только базовых алгоритмических структур, запрет на использование оператора GOTO.[2]

Структурный подход требовал представления задачи в виде иерархии подзадач простейшей структуры. Проектирование осуществлялось «сверху-вниз» и подразумевало реализацию общей идеи, обеспечивая проработку интерфейсов подпрограмм. Одновременно вводились ограничения на конструкции алгоритмов, рекомендовались формальные модели их описания, а также специальный метод проектирования алгоритмов — метод пошаговой детализации.

Поддержка принципов структурного программирования была заложена в основу так называемых процедурных языков программирования. Как правило, они включали основные «структурные» операторы передачи управления, поддерживали вложение подпрограмм, локализацию и ограничение области «видимости» данных. Среди наиболее известных языков этой группы стоит назвать PL/1, ALGOL-68, Pascal, C.

2.Состав и назначение инструментария технологии программирования

Инструментарий технологии программирования – это совокупность программ и программных комплексов, обеспечивающих технологию разработки, отладки и внедрения создаваемых программных продуктов. Пользователями этого класса программного обеспечения являются системные и прикладные программисты.

2.1 Инструментарий технологии программирования

Инструментарий технологии программирования делится на два больших класса инструментальных средств:

1. Инструментальные средства для создания отдельных приложений (программ):

- Локальные средства (языки программирования, системы программирования, инструментальные среды пользователя);

- Интегрированные среды разработки программ – среды, основное назначение которых – повышение производительности труда программистов за счет автоматизации создания кодов программ, обеспечивающих интерфейс пользователя графического типа, а также автоматизации разработки запросов и отчетов (например, Delphi).

1. Инструментальные средства для создания информационных систем и технологий (CASE- технология) – средства, поддерживающие полный цикл проектирования сложной информационной системы или технологии от исследования объекта автоматизации до оформления проектной и прочей документации на информационную систему или технологию. Они позволяют вести коллективную работу над проектом за счет возможности работы в локальной сети, экспорта – импорта любых фрагментов проекта, организации управления проектом.

2.2 Языки программирования

- Машинно-ориентированные (язык С) – объединяет идеи ассемблера и алгоритмического языка. Программы компактны и работают очень быстро.
- Универсальные (Turbo Pascal, Basic, Fortran) - приближены максимально, насколько это возможно, к естественному английскому языку: название каждой команды – английское слово; -Функциональные. Применяются, как

правило, для машинного моделирования той или иной проблематики. Имеют в составе:

- проблемно-ориентированные (GPSS) - моделируют систему с помощью последовательности событий. Применяются, в частности, при проектировании вычислительных комплексов;
- бъектно-ориентированные (Forth) - имеют встроенные средства для моделирования новых объектов программирования;
- логико-ориентированные (Prolog) - отдельно описываются правила предметной области, по которым затем выводятся новые факты. [5]

Системы программирования включают:

- Интегрированную среду разработчика программы. Основное назначение инструментария данного вида – повышение производительности труда программистов, автоматизация создания кодов программ, обеспечивающих интерфейс пользователя графического типа, разработка приложений для архитектуры клиент-сервер, запросов и отчетов;
- Транслятор – программу, переводящую исходный текст во внутреннее представление компьютера;
- Отладчик – программу для трассировки и анализа выполнения прикладных программ. Позволяет отслеживать выполнение программы в пооператорном режиме, идентифицировать место и вид ошибок в программе, наблюдают за изменением значений переменных, выражений и т.д.;
- Компоновщик – программа для подготовки прикладной программы к работе в конкретных адресах основной памяти компьютера;
- справочные системы.[3]

Инструментальная среда пользователя – это специальные программные средства, встроенные в пакет прикладных программ:

1. Библиотеки функций, процедур, объектов и методов обработки;

2. Макрокоманды;

3. Программные модули-вставки;

4. Конструкторы экранных форм и отчетов;

5. Языки запросов высокого уровня;

6. Клавишные макросы;

7. Языковые макросы;

8. Генераторы приложений;

9. Языки манипулирования данными;[2]

Основные алгоритмические структуры.

Понятие алгоритма

Алгоритмом называется точное предписание, определяющее последовательность действий исполнителя, направленных на решение поставленной задачи. В роли исполнителей алгоритмов могут выступать люди, роботы, компьютеры. Последовательность действий, которую необходимо выполнить над исходными данными, чтобы достичь поставленной цели, так же принято считать алгоритмом.

Можно утверждать, что алгоритмы – это способ фиксации и передачи знаний, накопленных человечеством, это богатство культуры, науки и техники. Роль алгоритмов в жизни человека весьма многогранна и не сводится только к обработке информации. Однако в процессе обработки информации алгоритмы играют первостепенную роль. Алгоритмы обладают важнейшим качеством – исполнение одного того же алгоритма в одних и тех же условиях различными людьми (в общем случае – исполнителями), как правило, приводит к одинаковым результатам. Следовательно, можно утверждать, что алгоритма обладают (точнее, должны обладать) некоторыми свойствами, которые обеспечивают этот эффект.[6]

Используются различные способы записи алгоритмов. Широко распространен словесный способ записи: это записи рецептов приготовления различных блюд в кулинарной книге, инструкции по использованию технических устройств, правила правописания и многие другие. Наглядно представляется алгоритм языком блок-схем. Компьютер «понимает» только алгоритмы, которые заданы в виде двоичных машинных кодов. Однако этот «естественный» для компьютеров, обладающий всеми свойствами способ записи алгоритмов, очень сложен для использования человеком. Поэтому в информатике применяется ряд специальных способов, языков задания, записи алгоритмов. Которые, во-первых, призваны обеспечить соответствие алгоритма всем необходимым требованиям, во-вторых, приспособлены для их использования как человеком, так и – после специальной обработки – компьютером. Такие искусственные языки, использующие для записи алгоритмов и обеспечивающие им наличие всех необходимых свойств, называются **алгоритмическими языками.**

3. Понятие исполнителя алгоритма

Понятие исполнителя невозможно определить с помощью какой-либо формализации. Исполнителем может быть человек, группа людей, робот, станок, компьютер, язык программирования и т.д. Важнейшим свойством, характеризующим любого из этих исполнителей, является то, что исполнитель умеет выполнять некоторые команды. Так, исполнитель-человек умеет выполнять такие команды как «встать», «сесть», «включить компьютер» и т.д., а исполнитель-язык программирования Бейсик - команды PRINT, END, LIST и другие аналогичные. Вся совокупность команд, которые данный исполнитель умеет выполнять, называется системой команд исполнителя (СКИ).

На *рис.2.* рассмотрен исполнитель-робот, работа которого состоит в собственном перемещении по рабочему полю (квадрату произвольного размера, разделенному на клетки) и перемещении объектов, в начальный момент времени находящихся на «складе» (правая верхняя клетка).[4]

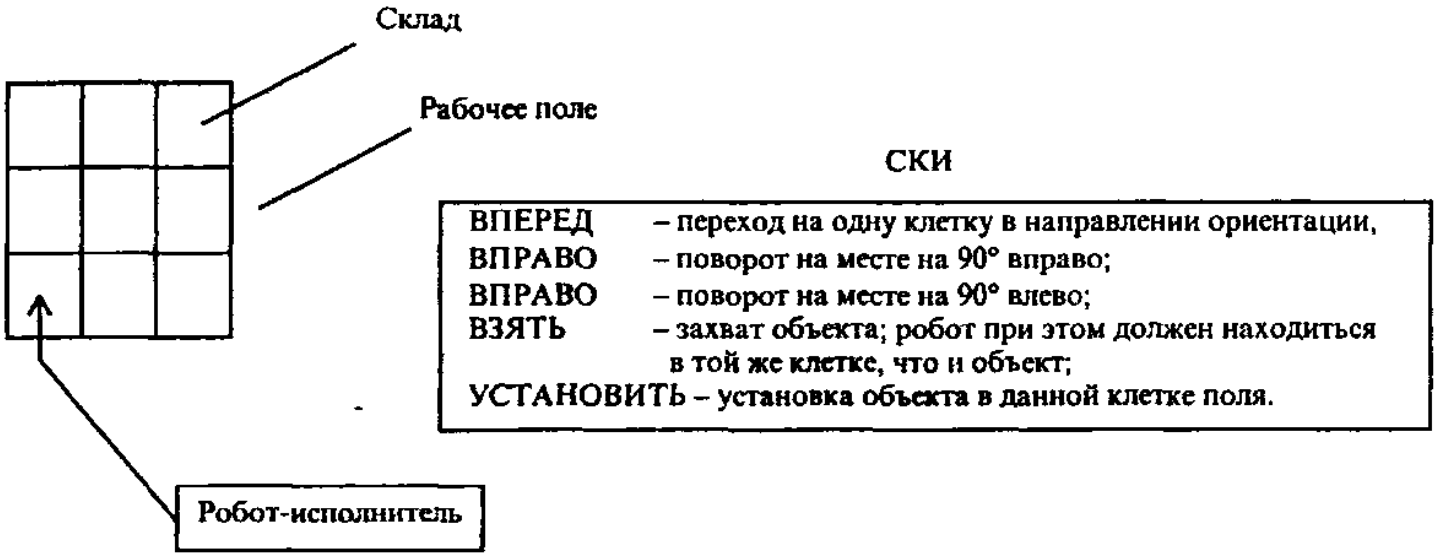

Рис.2 Исполнитель-робот.

Одно из принципиальных обстоятельств состоит в том, что исполнитель не вникает в смысл того, что он делает, но получает необходимый результат. В таком случае говорят, что исполнитель действует формально, т.е. отвлекается от содержания поставленной задачи и только строго выполняет некоторые правила, инструкции.

Это - важная особенность алгоритмов. Наличие алгоритма формализует процесс решения задачи, исключает рассуждение исполнителя. Использование алгоритма дает возможность решать задачу формально, механически исполняя команды алгоритма в указанной последовательности. Целесообразность предусматриваемых алгоритмом действий обеспечивается точным анализом со стороны того, кто составляет этот алгоритм.

Введение в рассмотрение понятия «исполнитель» позволяет определить алгоритм как понятное и точное предписание исполнителю совершить последовательность действий, направленных на достижение поставленной цели. В случае исполнителя-робота мы имеем пример алгоритма «в обстановке», характеризующегося отсутствием каких-либо величин. Наиболее же распространенными и привычными являются алгоритмы работы с величинами - числовыми, символьными, логическими и т.д.[8]

Свойства алгоритмов.

Алгоритм должен быть составлен таким образом, чтобы исполнитель, в расчете на которого он создан, мог однозначно и точно следовать командам алгоритма и эффективно получать определенный результат. Это накладывает на записи алгоритмов ряд обязательных требований, суть которых вытекает, вообще говоря, из приведенного выше неформального толкования понятия алгоритма. Сформулируем эти требования в виде перечня свойств, которым должны удовлетворять алгоритмы, адресуемые заданному исполнителю.

При составлении и записи алгоритма необходимо обеспечить, чтобы он обладал рядом свойств:

- однозначностью (детерминированностью) - любое действие алгоритма должно быть строго и недвусмысленно определено в каждом случае;
- дискретностью – разбиение алгоритма на ряд отдельных законченных действий (шагов);
- конечностью - каждое действие в отдельности и алгоритм в целом должны иметь возможность завершения;
- массовостью - возможность применения данного алгоритма для решения целого класса задач с разными исходными данными;
- результативностью - алгоритм должен приводить к правильному результату для всех допустимых входных значениях.[7]

Способы записи алгоритма.

- Графическая запись (блок-схемы)
- Словесная запись (псевдокоды)
- Язык программирования

Словесная форма записи алгоритма представляет собой описание на естественном языке последовательных этапов обработки данных. Словесный способ не имеет широкого распространения, так как такие описания строго не формализуемы, допускают неоднозначность толкования отдельных предписаний. Алгоритм, записанный с помощью псевдокода, представляет собой полуформализованное описание на условном алгоритмическом языке, включающее как основные элементы языка программирования, так и фразы естественного языка, общепринятые математические обозначения и другие.

Графическая форма записи, называемая также схемой алгоритма, представляет собой изображение алгоритма в виде последовательности связанных между собой функциональных блоков, каждый из которых соответствует выполнению одного или нескольких действий. Графическая запись является более компактной и наглядной по сравнению со словесной. В схеме алгоритма каждому типу действий соответствует геометрическая фигура. Фигуры соединяются линиями переходов, определяющими очередность выполнения действий.

Графическая форма записи, называемая также структурной схемой или блок-схемой алгоритма, представляет собой изображение алгоритма в виде последовательности связанных между собой функциональных блоков, каждый из которых соответствует выполнению одного или нескольких действий.[11]

В дальнейшем мы будем использовать блок-схемы алгоритмов. Они позволяют представить алгоритмы в более наглядном виде, это дает возможность анализировать их работу, искать ошибки в их реализации и т.д. В блок-схемах всегда есть начало и конец, обозначаемые эллипсами, между ними - последовательность шагов алгоритма, соединенных стрелками.[10]

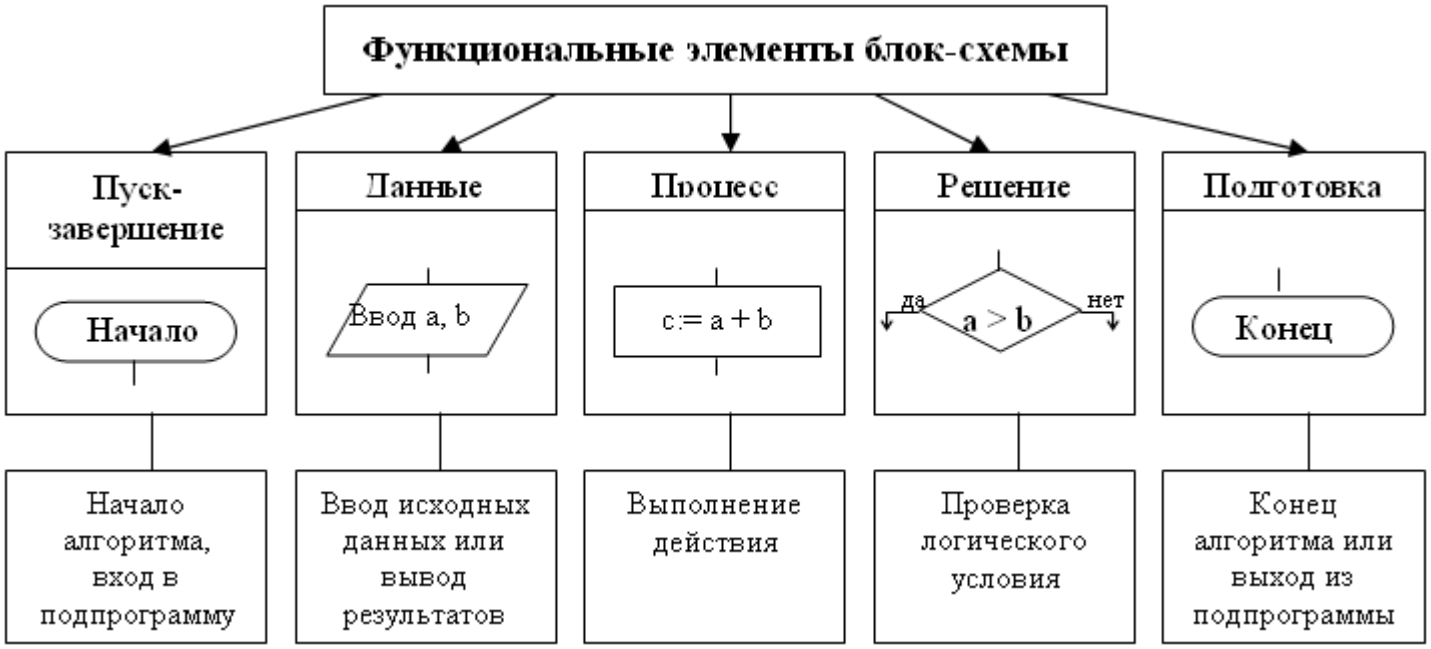

Рис.3 Функциональные элементы блок-схемы

Шаги бывают безусловными (изображаются прямоугольниками, параллелограммами) и условными (изображаются ромбами). Из ромба всегда выходят две стрелки - одна означает дальнейший путь, в случае выполнения условия (обозначается обычно словом "да" или "+"), другая - невыполнение (словом "нет" или "-"). Ввод с клавиатуры или вывод на экран значения выражения изображается параллелограммом. Команда, выполняющая обработку действий (команда присваивания), изображается в прямоугольнике. Пример блок-схемы представлен на *рис. 4*

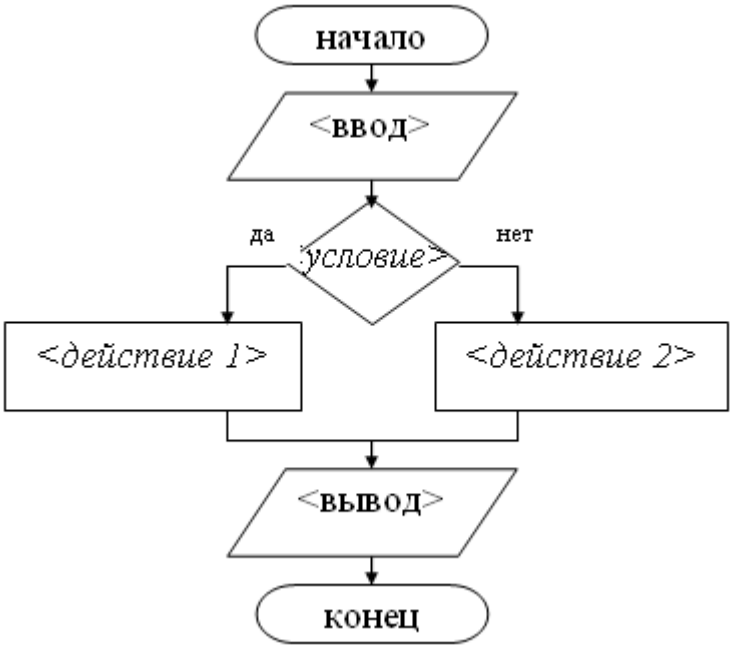

Рис.4 блок-схема

Виды алгоритмов

Алгоритмические структуры можно разделить на три основных вида:

- Линейные

Линейные алгоритмы — это набор команд, выполняемых последовательно во времени, друг за другом

- Разветвленные

Разветвленные алгоритмы - содержит хотя бы одно условие, в результате которого обеспечивается переход на один из двух возможных шагов.

- Циклические

Циклический алгоритм – это алгоритм, предусматривающий многократное повторение одного и того же действия над новыми данными.[11]

Линейный алгоритм

Линейным называется алгоритм, блоки которого расположены последовательно один за другим, нет условий и повторений.

Как правило, алгоритмы линейной структуры состоят из трех частей: ввод исходных данных, вычисления результатов по формулам, вывод значений результатов. Это самые простой алгоритм. Пример на *рис. 5*

Рис.5 линейная структура алгоритма.

Порядок выполнения блоков, как уже было сказано, определяют линии потока. По схеме видно, что при вводе совершенно различных исходных данных (по смыслу задачи, конечно, s>0) все действия этого алгоритма будут выполняться всегда, и порядок их выполнения никогда не изменяется.

Для отладки линейного алгоритма достаточно сравнить результаты его исполнения с результатами ручного решения задачи.

Разветвляющийся алгоритм

Разветвляющийся алгоритм (ветвление) обеспечивает выбор между двумя альтернативами. Выполняется проверка, а затем выбирается один из путей.

Подобная структура называется также «ЕСЛИ – ТО – ИНАЧЕ», или «развилка». Каждый из путей (ТО или ИНАЧЕ) ведет к общей точке слияния, так что выполнение

программы продолжается независимо от того, какой путь был выбран.

Может оказаться, что для одного из результатов проверки ничего предпринимать не надо. В этом случае можно применять только один обрабатывающий блок.[9]

Рис 6. Разветвляющаяся структура алгоритма

Рис 7. Структура «неполное ветвление»

Циклический алгоритм

Циклический алгоритм(Цикл) содержит некоторую последовательность операций, выполняемую многократно. Основной блок цикла, тело цикла, производит требуемые вычисления. Остальные блоки организуют циклический процесс: устанавливают начальные и новые значения данных, проверяют условия окончания или продолжения циклического процесса.

Различают два типа структур цикла: *цикл с параметром* или *с повторением и цикл с условием.* Циклический алгоритм позволяет компактно описать большое число одинаковых вычислений над разными данными для получения необходимого результата.

Циклы с параметром используют тогда, когда количество повторов тела цикла заранее известно. Пример представлен на *рис.8*

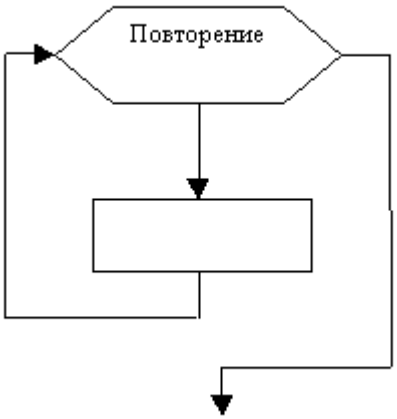

Рис 8. Структура цикла с параметром.

Циклы с условием используются тогда, когда число повторений заранее неизвестно, но задано условие окончания цикла. Причем, если условие окончания цикла проверяется перед выполнением тела цикла, то такие циклические структуры называют циклами *с предусловием* («Выполнять пока») Пример представлен на *рис. 9* [12]

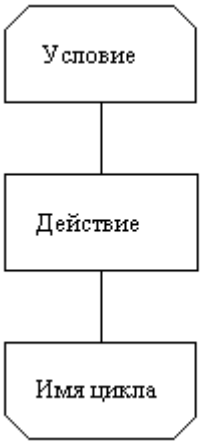

Рис.9.Структура цикла с предусловием.

А если проверка условия происходит после выполнения тела цикла – циклами *с постусловием* («Выполнять до тех пор, пока не». *Пример представлен на рис.10*

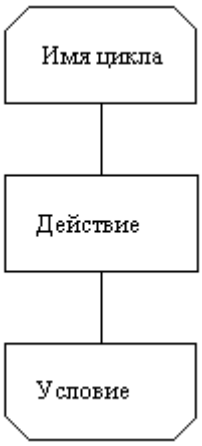

Рис 10. Структура цикла с постусловием

Методы технологии программирования.

Термины методов технологии программирования.

Основные определения:

Программа — завершенный продукт, пригодный для запуска своим автором на системе, на которой он был разработан.

Программный продукт — программа, которую любой человек может запускать, тестировать, исправлять и развивать. Такая программа должна быть написана в обобщенном стиле, тщательно оттестирована и сопровождена подробной

документацией. (С учетом модной в настоящее время концепции авторских прав, здесь необходимо уточнить – любой человек, имеющий разрешение работать с исходными текстами программ)

Программный комплекс — набор взаимодействующих программ, согласованных по функциям и форматам, точно определенным интерфейсам, и вкупе составляющих полное средство для решения больших задач.[13]

Жизненный цикл программного обеспечения – это весь период его разработки и эксплуатации, начиная с момента возникновения замысла и заканчивая прекращением ее использования.

Методология программирования – совокупность методов, применимых в жизненном цикле программного обеспечения и объединенных общим философским подходом.[14]

Технология программирования изучает технологические процессы и порядок их прохождения – стадии (с использованием знаний, методов и средств).

Процесс — совокупность взаимосвязанных действий, преобразующих некоторые входные данные в выходные. Процессы состоят из набора действий, а каждое действие из набора задач. Вертикальное измерение отражает статические аспекты процессов и оперирует такими понятиями, как рабочие процессы, действия, задачи, результаты деятельности и исполнители.[16]

Стадия — часть действий по созданию программного обеспечения, ограниченная некоторыми временными рамками и заканчивающаяся выпуском конкретного продукта, определяемого заданными для данной стадии требованиями. Стадии состоят из этапов, которые обычно имеют итерационный характер. Иногда стадии объединяют в более крупные временные рамки, называемые фазами. Итак, горизонтальное измерение представляет время, отражает динамические аспекты процессов и оперирует такими понятиями, как фазы, стадии, этапы, итерации и контрольные точки.

Технологический подход определяется спецификой комбинации стадий и процессов, ориентированной на разные классы программного обеспечения и на особенности коллектива разработчиков.[15]

4. Методы программирования

4.1. Модульное программирование

Модульное программирование является развитием и совершенствованием процедурного программирования и библиотек специальных программ. Основная черта модульного программирования – стандартизация интерфейса между отдельными программными единицами. Модуль – это отдельная функционально-законченная программная единица, которая структурно оформляется стандартным образом по отношению к компилятору и по отношению к объединению ее с другими аналогичными единицами и загрузке. Как правило, каждый модуль содержит паспорт, в котором указаны все основные его характеристики: язык программирования, объем, входные и выходные переменные, их формат, ограничения на них, точки входа, параметры настройки и т.д. Объем модуля обычно не превышает 1000 команд ЭВМ или операторов языка программирования. В противном случае модуль становится громоздким и трудным к восприятию и использованию.[19]

Модульное программирование — это искусство разбиения задачи на некоторое число различных модулей, умение широко использовать стандартные модули путем их параметрической настройки, автоматизация сборки готовых модулей из библиотек, банков модулей.

Основные концепции модульного программирования:

- каждый модуль реализует единственную независимую функцию;
- каждый модуль имеет единственную точку входа и выхода;
- размер модуля по возможности должен быть минимизирован;
- каждый модуль может быть разработан и закодирован различными членами бригады программистов и может быть отдельно протестирован;
- вся система построена из модулей;
- модуль не должен давать побочных эффектов;
- каждый модуль не зависит от того, как реализованы другие модули.[15]

При таком подходе сложная система разделяется на несколько частей, одновременно создаваемых различными программистами. Каждый модуль реализует единственную функцию. Размер модуля невелик, поэтому тестирование управляемо и может быть проведено тщательным образом. После кодирования и тестирования всех модулей происходит их интеграция, и тестируется вся система.

При сопровождении тестируется и отлаживается только тот модуль, который плохо работает. Очевидны преимущества в облегчении написания и тестирования

программ, уменьшается стоимость их сопровождения.

Основные характеристики программного модуля

Не всякий программный модуль способствует упрощению программы. Выделить хороший с этой точки зрения модуль является серьезной творческой задачей. Для оценки приемлемости выделенного модуля используются некоторые критерии. Так, Хольт предложил следующие два общих таких критерия: [14]

1. размер модуля;
2. прочность модуля;
3. сцепление с другими модулями;
4. рутинность модуля (независимость от предыстории обращений к нему).
5. Размер модуля.

Размер модуля измеряется числом содержащихся в нем операторов или строк. Модуль не должен быть слишком маленьким или слишком большим. Маленькие модули приводят к громоздкой модульной структуре программы и могут не окупать накладных расходов, связанных с их оформлением. Большие модули неудобны для изучения и изменений, они могут существенно увеличить суммарное время повторных трансляций программы при отладке программы. Обычно рекомендуются программные модули размером от нескольких десятков до нескольких сотен операторов.

1. Прочность модуля

Прочность модуля — это мера его внутренних связей. Чем выше прочность модуля, тем больше связей он может спрятать от внешней по отношению к нему части программы и, следовательно, тем больший вклад в упрощение программы он может внести. Для оценки степени прочности модуля Майерс предлагает упорядоченный по степени прочности набор из семи классов модулей. Самой слабой степенью прочности обладает модуль, прочный по совпадению. Это такой модуль, между элементами которого нет осмысленных связей. Такой модуль может быть выделен, например, при обнаружении в разных местах программы повторения одной и той же последовательности операторов, которая и оформляется в отдельный модуль. Необходимость изменения этой последовательности в одном из контекстов может привести к изменению этого модуля, что может сделать его использование в других контекстах ошибочным. Такой класс программных модулей не рекомендуется для использования. Вообще говоря, предложенная Майерсом упорядоченность по степени прочности классов модулей не бесспорна. Однако, это

не очень существенно так, как только два высших по прочности класса модулей рекомендуются для использования. Эти классы я рассмотрю подробнее. [19]

Функционально прочный модуль — это модуль, выполняющий (реализующий) одну какую-либо определенную функцию. При реализации этой функции такой модуль может использовать и другие модули. Такой класс программных модулей рекомендуется для использования.

1. Сцепление модуля.

Сцепление модуля — это мера его зависимости по данным от других модулей. Характеризуется способом передачи данных. Чем слабее сцепление модуля с другими модулями, тем сильнее его независимость от других модулей. Для оценки степени сцепления Майерс предлагает упорядоченный набор из шести видов сцепления модулей. Худшим видом сцепления модулей является сцепление по содержимому. Таким является сцепление двух модулей, когда один из них имеет прямые ссылки на содержимое другого модуля (например, на константу, содержащуюся в другом модуле). Такое сцепление модулей недопустимо. Не рекомендуется использовать также сцепление по общей области - это такое сцепление модулей, когда несколько модулей используют одну и ту же область памяти. Такой вид сцепления модулей реализуется, например, при программировании на языке ФОРТРАН с использованием блоков COMMON. Единственным видом сцепления модулей, который рекомендуется для использования современной технологией программирования, является параметрическое сцепление (сцепление по данным по Майерсу) - это случай, когда данные передаются модулю либо при обращении к нему как значения его параметров, либо как результат его обращения к другому модулю для вычисления некоторой функции. Такой вид сцепления модулей реализуется на языках программирования при использовании обращений к процедурам (функциям).[13]

В таблице 1 приведены характеристики различных типов сцепления по экспертным оценкам. Допустимыми считают первые три типа сцепления, так как использование остальных приводит к резкому ухудшению технологичности программ.

Таблица 1

Характеристики типов сцепления

Тип сцепления	Сцепление, балл	Устойчивость к ошибкам других модулей	Наглядность (понятность)	Возможность изменения	Вероятность повторного использования
По данным	1	Хорошая	Хорошая	Хорошая	Большая
По образцу	3	Средняя	Хорошая	Средняя	Средняя
По управлению	4	Средняя	Плохая	Плохая	Малая
По общей области	6	Плохая	Плохая	Средняя	Малая
По содержимому	10	Плохая	Плохая	Плохая	Малая

1. Рутинность модуля.

Рутинность модуля — это его независимость от предыстории обращений к нему. Модуль называется рутинным, если результат (эффект) обращения к нему зависит только от значений его параметров (и не зависит от предыстории обращений к нему). Модуль называется зависящим от предыстории, если результат (эффект) обращения к нему зависит от внутреннего состояния этого модуля, изменяемого в результате предыдущих обращений к нему. Майерс не рекомендует использовать зависящие от предыстории (непредсказуемые) модули, так как они провоцируют появление в программах хитрых (неуловимых) ошибок. Однако такая рекомендация является неконструктивной, так как во многих случаях именно зависящий от предыстории модуль является лучшей реализаций информационно прочного модуля.[12]

Разновидности модулей.

Существуют три основные разновидности модулей:

- "Маленькие" (функциональные) модули, реализующие, как правило, одну какую-либо определенную функцию. Основным и простейшим модулем практически во всех языках программирования является процедура или функция.
- "Средние" (информационные) модули, реализующие, как правило, несколько операций или функций над одной и той же структурой данных (информационным объектом), которая считается неизвестной вне этого модуля. Примеры "средних" модулей в языках программирования:

a) задачи в языке программирования Ada;

b) кластер в языке программирования CLU;

c) классы в языках программирования C++ и Java.

- "Большие” (логические) модули, объединяющие набор "средних" или "маленьких" модулей. Примеры "больших" модулей в языках программирования:

a) модуль в языке программирования Modula-2;

b) пакеты в языках программирования Ada и Java.

При разработке модульного программирования может использоваться метод восходящей разработки или метод нисходящей разработки. *(рис. 11)*[19]

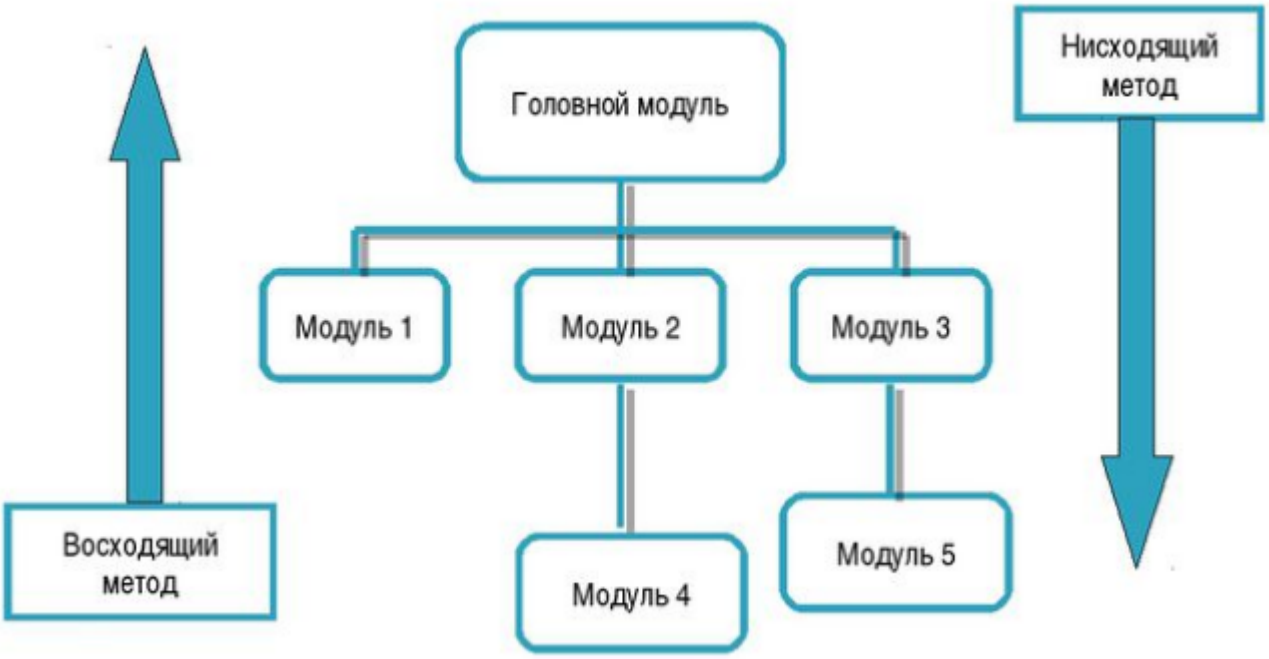

Рис.11. Структура метода восходящей, нисходящей разработки.

Выводы по 1 части.

- Модульное программирование связано с проектированием сверху вниз и заключается в том, что программа разбивается на части, которые называются модулями и разрабатываются по отдельности.

4.2. Структурное программирование

Структурное программирование основано главным образом на теоретическом аппарате теории рекурсивных функций. Программа рассматривается как частично-рекурсивный оператор [21] над библиотечными подпрограммами и исходными операциями. Структурное программирование базируется также на теории доказательств, прежде всего на естественном выводе. Структура программы соответствует структуре простейшего математического рассуждения, не использующего сложных лемм и абстрактных понятий.

Средства структурного программирования в первую очередь включаются во все языки программирования традиционного типа и во многие нетрадиционные языки.

Основные принципы структурного программирования:

- сложная задача разбивается на простые, функционально управляемые задачи, каждая задача имеет один вход и один выход; управляющий поток программы состоит из совокупности элементарных функциональных подзадач;
- управляющие структуры просты, т. е. логическая задача должна состоять из минимальной, функционально полной совокупности достаточно простых управляющих структур;
- программа разрабатывается поэтапно, на каждом этапе решается ограниченное число точно поставленных задач.

Свойство структурности операторов состоит в том, что каждый оператор имеет один вход и один выход. Программа, построенная из структурных операторов, называется структурированной.[19]

Фундаментом структурного программирования является теорема о структурировании, которая устанавливает, что как бы сложна ни была задача, схема соответствующей программы всегда может быть представлена с использованием ограниченного числа элементарных управляющих структур.

Если технологию разработки алгоритмов «сверху - вниз» совместить с использованием только структурных схем, то получится новая технология, которая называется структурным программированием 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 не только с помощью трех простых операторов, но и с помощью средств, более точно отражающих 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 (занимающих до 50 операторов — критический порог для быстрого понимания цели подпрограммы), 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.[24]

Нисходящее проектирование

Наличие подпрограмм позволяет вести проектирование и разработку приложения сверху вниз — такой подход называется нисходящим проектированием. 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.[21]

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. Например, если имеется необходимость вычисления сложной математической функции, то выделяется отдельная подпрограмма такого вычисления, но реализуется она временно одним оператором, который просто присваивает заранее выбранное значение (например, 5). Когда все приложение будет написано и отлажено, тогда можно приступить к реализации этой функции.[20]

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.

Процедуры и функции

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).

В Си++ понятия «процедура» нет — там имеются только функции, а если никакого значения функция не вычисляет, то считается, что она возвращает значение типа «никакое» (void)

Параметры подпрограмм

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.[24]

Управление последовательностью вызова подпрограмм

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.

Элементарные (базовые) структуры могут соединяться между собой, образуя более сложные структуры, по тем же самым элементарным схемам. Виды основных управляющих структур алгоритма приведены на *рис. 12.*

- Структура типа «следование» (*рис.12 а*) – образуется последовательностью действий, S1, S2, …, Sn, следующих одно за другим: выполнить S1; выполнить S2; …; выполнить Sn. В линейном

вычислительном процессе все операции выполняются последовательно в порядке их записи. Типовым примером такого процесса является стандартная вычислительная схема, состоящая из трех этапов:

1) ввод исходных данных;

2) вычисление по формулам;

3) вывод результата.

- Структура типа «ветвление» (ЕСЛИ – ТО – ИНАЧЕ) *(рис.12 б)* – обеспечивает в зависимости от результата проверки условия P, принимающего одно из двух логических значении Да (True) или Нет (False), выбор одного из альтернативных путей работы алгоритма: если P, то выполнить S1, иначе выполнить S2. Каждый из путей ведет к общему выходу, так что работа алгоритма будет продолжаться независимо от того, какой путь будет выбран.
- Структура типа «цикл с предусловием» (*рис.12 в*) – обеспечивает многократное выполнение действия S в зависимости от того, какое значение принимает логическое условие P: до тех пор, пока P, выполнять S. Выполнение цикла прекращается, когда условие P не выполняется.[21]

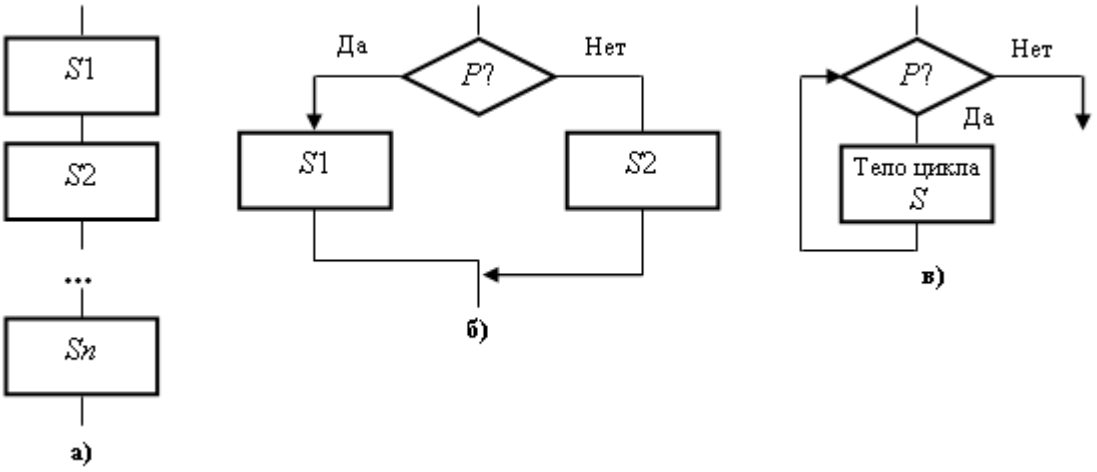

Рис.12 Базовые алгоритмические структуры.

В расширенный комплект элементарных алгоритмических структур дополнительно входят следующие управляющие конструкции:

- Структура типа «сокращенное ветвление» (ЕСЛИ – ТО) (*рис. 13 а*) – если результат проверки условия P принимает значение Да (True), то выполняется действие S; в противном случае это действие пропускается и управление передается следующей структуре: если P, то выполнить S1.

В языке Pascal такая структура имеет следующий формат:

If P Then S;

- Структура типа «выбор – иначе» (*рис. 13 б*) являются расширенным вариантом структуры типа ЕСЛИ – ТО – ИНАЧЕ. Здесь проверяемое условие P может принимать не два логических значения, а несколько порядковых значений, например, 1, 2, …, n. Если P = i, то будет выполняться действие Si.

Если же значение P будет выходить из диапазона допустимых значений, то выполняется действие S (в укороченном варианте «выбор» никакого действия не производится и управление передается к следующей структуре.

- Case P Of
- S1;
- S2;
- …
- N: Sn
- Else S
- End
- . Структура типа «цикл с постусловием» (*рис. 13 в*) – обеспечивает многократное выполнение действия S до тех пор, пока не выполняется условие P.

В языке Pascal такая структура имеет следующий формат:

- Repeat
- S
- Until P;
- Структура типа «цикл с параметром» (*рис. 13* г) – обеспечивает заранее определенное многократное выполнение действия S. При этом здесь последовательно выполняются следующие типовые операции:

1. Задание начального значения используемого параметра цикла (например, если переменной цикла является i, то ей присваивается значение i1, т.е. i:=i1);
2. выполнение действий S, предусмотренных в теле цикла;
3. изменение параметра цикла, который обеспечивает вычисление результата с новыми начальными данными (например, если параметр цикла i изменяется с шагом i3, i:=i+ i3);
4. проверка текущего значения параметра цикла с заданным конечным значением (i)

5. переход к повторению тела цикла, если параметр цикла не превысил конечного значения, иначе — выполнение следующих действий или вывод результата.[22]

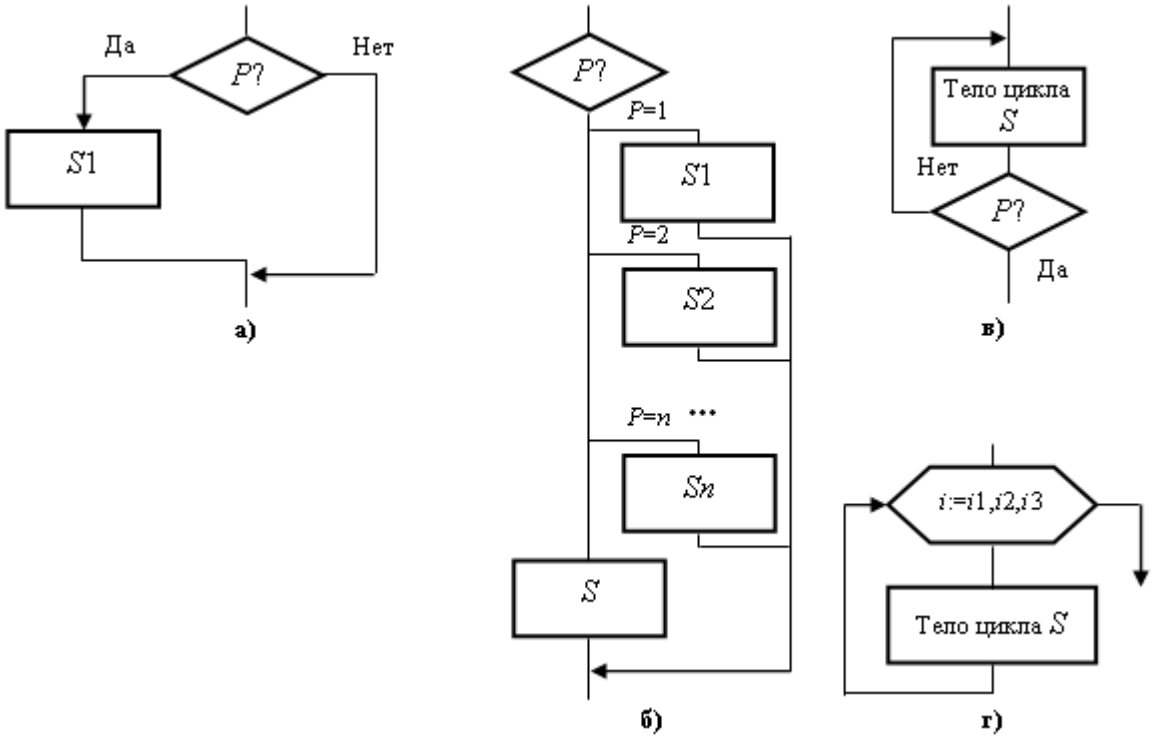

Рис 13. Дополнительные алгоритмические структуры.

Выводы по 2 части.

- Структурное программирование (кодирование) — это метод написания программ, имеющих определенную структуру. Он основан на использовании небольшого набора структурных операторов, правильность которых легко проанализировать и установить. При этом одни операторы состоят из других, вложенных в них.
- В основе структурного проектирования лежат принципы последовательной декомпозиции задачи и целенаправленного ее структурирования на отдельные составляющие. Методы структурного проектирования представляют собой комплекс технических и организационных принципов системного проектирования программных продуктов.
- Постепенное уточнение проекта называется методом нисходящего проектирования (пошаговой детализации или проектирования сверху вниз). Средства достижения целей на предыдущем уровне превращаются в цели на нижнем.

4.3. Объектно-ориентированное программирование

Концепция объектно-ориентированного программирования 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. В этом состоит главное отличие ООП от процедурного программирования, где отдельно определённые структуры данных передаются в процедуры (функции) в качестве параметров. Таким образом, объектно-ориентированная программа состоит из объектов – отдельных фрагментов кода, обрабатывающего данные, которые взаимодействуют друг с другом через определённые интерфейсы.[23]

Объектно-ориентированный язык программирования должен обладать следующими свойствами:

1. абстракции – формальное о качествах или свойствах предмета путем мысленного удаления некоторых частностей или материальных объектов;
2. инкапсуляции – механизма, связывающего вмести 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 иерархической классификации;
4. полиморфизма – свойства, позволяющего использовать один и тот же интерфейс для общего класса действий.[25]

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, то есть устранение всех вспомогательных промежуточных материалов, использовавшихся при проектировании;
- кодирование, отладка, компоновка и тестирование.

Объектно-ориентированное программирование позволяет программисту моделировать объекты определённой предметной области путем программирования их содержания и поведения в пределах класса. Конструкция «класс» обеспечивает механизм инкапсуляции для реализации абстрактных типов данных. Инкапсуляция как бы скрывает и подробности внутренней реализации типов, и внешние операции и функции, допустимые для выполнения над объектами этого типа.[24]

Объекты в объектно-ориентированном программировании:

Базовым в объектно-ориентированном программировании является понятие объекта. Объект имеет определённые свойства. Состояние объекта задаётся значениями его признаков. Объект «знает», как решать определённые задачи, то есть располагает методами решения. Программа, написанная с использованием ООП, состоит из объектов, которые могут взаимодействовать между собой.

В отличие от типа «запись», объектный тип содержит не только поля, описывающие данные, но также процедуры и функции, описания которых содержится в описании объекта. Эти процедуры и функции называют методами. Методам объекта доступны его поля. Следует отметить, что методы и их параметры определяются в описании объекта, а их реализация даётся вне этого описания, в том мест программы, которое предшествует вызову данного метода. В описании объекта фактически содержаться лишь шаблоны обращения к методам, которые необходимы компилятору для проверки соответствия количества параметров и их типов при обращении к методам.[23]

Выводы по 3 части.

Объектно-ориентированное программирование имеет целый ряд преимуществ:

- четко выраженная модульность построения программ и использование абстрактных типов данных.
- динамическая связь между программными модулями увеличивает гибкость языка программирования и позволяет вводить (определять) новые типы данных без необходимости модификации всей программы.
- возможность многократного использования одних и тех же модулей.

Вместе с тем объектно-ориентированному программированию присущи и некоторые недостатки. Одним из наиболее существенных недостатков является замедление выполнения программ, обусловленное механизмом динамической связи. Это вызвано активным обменом информацией между программными модулями. Другим недостатком является сложность разработки соответствующих трансляторов по сравнению с трансляторами традиционных языков программирования. Однако преимущества объектно-ориентированного программирования, несомненно, перевешивают недостатки, что делает его перспективным средством разработки сложных программных систем.

4.4. Логическое программирование

Центральным понятием в логическом программировании является отношение. Программа представляет собой совокупность определений отношений между объектами (в терминах условий или ограничений) и цели (запроса). Процесс выполнения программы трактуется как процесс общезначимости логической формулы, построенной из программы по правилам, установленным семантикой используемого языка. Результат вычисления является …побочным продуктом этого процесса. В реляционном программировании нужно только специфицировать факты, на которых алгоритм основывается, а не определять последовательность шагов, которые требуется выполнить. Это свидетельствует о декларативности языка логического программирования. Она метко выражена в формуле Р. Ковальского: «алгоритм = логика + управление».

Языки логического программирования характеризуются:

- высоким уровнем;
- строгой ориентацией на символьные вычисления;
- возможностью инверсных вычислений, то есть переменные в процедурах не делятся на входные и выходные;
- возможной логической неполнотой, поскольку зачастую невозможно выразить в программе определенные логические соотношения, а также невозможно получить из программы все правильные выводы.

Логические программы, в принципе, имеют небольшое быстродействие, так как вычисления осуществляются методом проб и ошибок, поиском с возвратами к предыдущим шагам.[26]

Языки логического программирования

- Чистый Пролог

Логические программы, исполняемые с помощью вычислительной модели Пролога, называются программами на чистом Прологе. Чистый Пролог представляет собой приближённую реализацию вычислительной модели логического программирования на последовательной машине. Конечно, данная реализация не является единственно возможной. Она является, однако, одной из наилучших с практической точки зрения – совмещения свойств абстрактного интерпретатора с возможностью эффективного выполнения.

При построении на основе абстрактного интерпретатора некоторого интерпретатора для конкретного языка программирования необходимо принять два решения. Во-первых, следует ограничить произвол в выборе редуцируемой цели в резольвенте, т.е. уточнить метод расписания. Во-вторых, нужно реализовать недетерминированный выбор предложения программы, используемого в редукции.

Существуют разные языки программирования, использующие различные способы выбора. Грубо говоря, они делятся на два класса. Пролог и его расширения основаны на последовательном выполнении. Другие языки, такие, как Parlog, Параллельный Пролог и GHC, основаны на параллельном выполнении. Последовательные языки отличаются от параллельных методом реализации недетерминизма. Отличие Пролога от его версий состоит в методе выбора редуцируемой цели.[29]

Выполнение программ на Прологе заключается в работе абстрактного интерпретатора, при которой вместо произвольной цели выбирается самая левая цель, а недетерминированный выбор предложения заменяется последовательным поиском унифицируемого правила и механизма возврата.

Другими словами, в Прологе используется стековый метод расписания. В Прологе резольвента используется как стек – для редукции выбирается верхняя цель, производные цели помещаются в стек резольвенты.

В дополнение к методу стека Пролог моделирует недетерминированный выбор редуцирующего правила с помощью последовательного поиска и механизма возврата. При попытке редуцировать цель выбирается первое предложение, заголовок которого унифицируем с данной целью. Если не существует правила, унифицируемого с выбранной целью, то вычисление восстанавливается на стадии

последнего сделанного выбора и выбирается следующее унифицируемое предложение.[28]

Сравнение с традиционными языками программирования

Язык программирования характеризуется присущими ему механизмами управления и обработки данных. Пролог как универсальный язык программирования можно рассматривать и с этих точек зрения.

При успешном выполнении вычисления средства управления программ на языке Пролог подобны средствам управления в обычных процедурных языках. Обращение к некоторой цели соответствует вызову процедуры, порядок целей в теле правила соответствует последовательности операторов. Точнее, правило A = B1 ,B2 ,…,Bn можно рассматривать как определение процедуры:

Procedure A

Call B1

Call B2

Call Bn

End

Рекурсивный вызов цели в Прологе в последовательности действий и в реализации подобен тому же вызову в обычных рекурсивных языках. Различие возникает при реализации возврата. В обычных языках, если вычисление не может быть продолжено (например, все ветви в операторе case ложны), возникает ошибка выполнения. В Прологе вычисление просто возвращается к последнему выбору и делается попытка продолжить вычисления по новому пути.[30]

Структуры данных, которыми оперируют логические программы, - термы – соответствуют общим структурам записей в обычных языках программирования. Пролог использует очень гибкую систему организации структур данных. Подобно языку Лисп, Пролог является бестиповым языком, не содержащим объявления данных.

Другие особенности использования структур данных в языке Пролог связаны с природой логических переменных. Логические переменные соотносятся с объектами, а не с ячейками памяти. Если переменной сопоставлен конкретный

объект, то эта переменная уже никогда не может ссылаться на другой объект. Иными словами, логическое программирование не поддерживает механизм деструктивного присваивания, позволяющий изменять значение инициализированной переменной.

В логическом программировании обработка данных полностью заключена в алгоритме унификации. В унификации реализованы:

- однократное присваивание,
- • передача параметров,
- • размещение записей,
- • доступ к полям записей для одновременных чтения/записи.[27]

Порядок правил

Два синтаксических понятия, несущественные в логических программах, важны при создании программ на Прологе. В каждой процедуре должен быть принят порядок правил, или порядок предложений. Кроме того, в теле каждого предложения должен быть определён порядок целей. Последствия этих решений могут оказаться колоссальными: эффективность программирования на Прологе может измениться в десятки раз. В крайних и тем не менее распространённых случаях корректные логические программы вообще не приведут к решению вследствие не завершающегося вычисления.[30]

Порядок правил определяет порядок поиска решений.

Изменение порядка правил в процедуре приводит к перестановке ветвей в любом дереве поиска цели, использующей данную процедуру. Обход дерева поиска происходит в глубину. Поэтому перестановка ветвей дерева изменяет порядок обхода дерева и порядок нахождения решений. Этот эффект очевиден при использовании фактов для нахождения ответов на экзистенциальный вопрос.

Порядок, в котором находятся ответы на вопросы при работе рекурсивной программы, определяется порядком предложений.

Порядок предложений в программах на общепринятом Прологе важнее, чем порядок предложений в программах на чистом Прологе.[29]

Проблема завершения программ.

Используемый в Прологе принцип обхода дерева в глубину приводит к серьёзным проблемам. Если дерево поиска цели относительно некоторой программы содержит бесконечную ветвь, то вычисление не завершится. Пролог может не найти решение цели, даже если существует конечное вычисление, решающее вопрос.

Бесконечные вычисления появляются при использовании рекурсивных правил. Рекурсивные правила, в которых рекурсивная цель является первой целью в теле правила, называются левыми рекурсивными правилами. Левые рекурсивные правила в Прологе приносят немало хлопот. В случае несоответствующих аргументов использование этих правил приводит к бесконечным вычислениям.[26]

Лучшее решение этой проблемы – отказаться от использования левой рекурсии. В общем случае невозможно избавиться от всех появлений левой рекурсии. Однако соответствующий анализ позволяет определить вопросы, решение которых относительно рекурсивных программ приводит к результату. Другой, не всегда замечаемый случай, который заведомо приводит к не завершающимся вычислениям – это порочный круг. Ввиду порочного круга дерево поиска обязательно содержит бесконечную ветвь.

- Язык логического программирования KL0.

KL0 (от англ. “kernel-language version 0” – ядро-язык версии 0) – язык, в основу которого положено расширение языка логического программирования Пролог. Среди особенностей, новых в KL0 по отношению к Прологу, можно выделить:

- ◦ более гибкую структуру управления.
  ◦ многопроцессовость
  ◦ операции с побочным эффектом
  ◦ машинно-ориентированные операции.

К наиболее существенным механизмам Пролога, не поддерживаемым в KL0, относятся:

- ◦ средства управления базой данных
  ◦ средства управления таблицей имён.

Так как KL0 мало, чем отличается от Пролога, ограничимся лишь рассмотрением типов данных.

К базовым типам языка относятся:

- символы,
- целые и действительные числа
- строки
- др.

Символы в основном предназначены для представления символьных атомов Пролога и, как правило, никак не связаны ни со строками символов, используемыми для текстуального представления программ, ни с определениями предикатов, в которых символы задают имена предикатов. Такие атрибуты при необходимости могут быть приписаны символам средствами ESP[1] . KL0 более прост и не поддерживает подобных механизмов. В этом отношении он только обеспечивает структуры данных прямого доступа и стандартную функцию хеширования для доступа к определениям атрибутов в хеш-таблице. Символы в KL0 можно проверять только на идентичность.[30]

Целые и действительные числа введены для эффективного выполнения арифметических операций. Арифметические операции в KL0 не обладают свойством двойственности: сложение и вычитание, здесь различные предикаты. Аппаратно поддерживаются только числа фиксированной длины, определяемой разрядной сеткой. Целые числа произвольной длины (bignums), действительные числа произвольной точности и, возможно, рациональные числа могут быть реализованы с помощью обработчика исключений. Исключение возбуждается, если операндами встроенных арифметических предикатов (машинных инструкций в традиционном смысле) являются, например, нечисловые данные. Обработчик исключения может проверить аргументы и, если они соответствуют ожидаемым, выполнить предписанные операции; в противном случае вызывается обработчик ошибок. После обработки исключения дальнейшее выполнение программы может быть возобновлено.[32]

Выводы по 4 части.

- Логическое программирование хорошо подходит для решения проблем, для работы с формальными и естественными языками, для баз данных, запросных и экспертных систем и для других дискретных не вычислительных задач. Пользователя привлекает ясность, содержательность программ и их нетехнический характер. В программе не нужно описывать, каким образом решается задача. Достаточно описания самой задачи и того, что желательно узнать.

- Однако логическое программирование с использованием лишь хорновских предложений было бы слишком узконаправленным. Поэтому, кроме этого, используются другие методы программирования. Некоторые задачи по своему характеру процедурные, и программировать их чисто декларативными языками непрактично. Нужны более развитые типы данных. Пролог и логическое программирование непрерывно расширяются, охватывая все новые методы программирования и формы изображения именно в направлении процедурного и объектно-ориентированного программирования, а также в направлении параллельных вычислений.

4.5. Функциональное программирование

Функциональным называется программирование при помощи функций в математическом их понимании. Функциональное программирование основано на следующей идее: в результате каждого действия возникает значение, которое может быть аргументом следующего действия. Программы строятся из логически расчлененных определений функций. Каждое определение функции состоит из организующих вычисления управляющих структур и из вложенных, в том числе вызывающих самих себя (рекурсивных) вызовов функций.

Язык LISP (LISt Processing) – язык программирования высокого уровня, разработан в 1961 году Дж. Маккарти. В основе Лиспа лежит функциональная модель вычислений, ориентированная прежде всего на решение задач нечислового характера.[31]

Основы функционального программирования:

- Вызов функций является единственной разновидностью действий,

выполняемых в функциональной программе,

- В алгоритмических языках программа является последовательностью операторов, вызовов процедур в соответствии с алгоритмом. В функциональном программировании программа состоит из вызовов функций *(рис. 14)* и описывает то, что нужно делать и что собой представляет результат решения, а не как нужно действовать для получения результата.

Рис.14 Структура функциональной программы.

- Основными методами программирования являются суперпозиция функций и рекурсия.
- В алгоритмических языках с именем переменной связана некоторая область памяти, соответствие строго сохраняется в течение всего времени выполнения программы. В функциональном программировании переменная обозначает только имя некоторой структуры, имена символов, переменных, списков, функций и других объектов не закреплены предварительно за какими-либо типами данных. В ФП одна и та же переменная в различные моменты времени может представлять различные объекты.[32]
- В языках функционального программирования программа и обрабатываемые ею данные имеют единую списочную форму представления.
- Функциональное программирование предполагает наличие функционалов – функций, аргументы и результаты которых могут быть функциями. Всякий язык функционального программирования предполагает наличие ядра, называемого строго функциональным языком.

Требования к функциональному языку

- Всякая функция должна однозначно определять результат по любому набору аргументов.
- Отсутствует оператор присваивания.
- Переменная обозначает только имя структуры. 4. В языке присутствуют функционалы.

Применение языков функционального программирования.

- Системы автоматизированного проектирования.
- Программирование игр.
- Математическая лингвистика.
- Реализация ленивых вычислений.[33]

Выводы по 5 части.

Основные преимущества языков функционального программирования:

- Краткость программы.
- Функциональные программы поддаются формальному анализу легче своих аналогов на алгоритмических языках за счет использования математической функции в качестве основной конструкции.
- Возможность реализации с параллельной архитектурой

Заключение

Анализ рассмотренных методов систематического и теоретического программирования показывает постоянное их развитие, совершенствование, пополнение новыми возможностями и возможностью объединения. Классическим примером объединения является язык UML, в результате объектно-ориентированное программирование обогатилось новыми возможностями, которые удовлетворили многих пользователей визуальным и наглядным моделированием программных систем на основе разнообразных диаграмм. Каждый метод программирования может развиваться как самостоятельно, так и в сообществе с другими. Как считают многие специалисты в области информатики, перспективными среди рассмотренных методов программирования являются методы модульного и ориентированного программирования.

Список использованной литературы

1. Хомский Н. Аспекты теории синтаксиса. – МГУ, 2006.
2. Гладкий А.В. Формальные грамматики и языки. – М.: Наука, 1973.
3. Льюис Ф., Розенкранц Д., Стирнз Р. Теоретические основы проектирования компиляторов. – М.: Мир, 2008.
4. Агальцев В.П. Информатика для экономистов: учебник / В.П. агальцев В.М. Титов. – М.: Форум, Инфра-М, 2011. – 448 с.

5. Информатика: учебник / Под ред. проф. Н.В. Макаровой – 3-е изд. – М.: Финансы и статистика, 2000. – 768 с.
6. Волков В.Б. Информатика: учеб. для вузов /В.Б. Волков Н.В. Макарова. – СПб.: Питер, 2011. – 576 с.
7. Гагарина Л.Г. Технология разработки программного обеспечения / Л.Г. ГагаринаЕ.В. КокореваБ.Д. Виснадул. – М.: Форум, Инфра-М, 2007. – 400 с.
8. Жоголев Е.А. Технология программирования / Е.А. Жоголев. – М.: Научный мир, 2004. – 216 с.
9. Иванова Г.С. Технология программирования: учебник / Г.С. Иванова. - 2-е изд. - М.: МГТУ им. Н.Э. Баумана, 2002. – 416 с.
10. Информатика: Базовый курс: учеб. для вузов / под ред. С.В. Симоновича. – 3-е изд. – СПб.: Питер, 2011. – 640 с.
11. Информатика: учебник / Соболь Б.В. [и др.] -3-е изд., доп. и перераб. - Ростов н/Д: Феникс, 2007. — 446 с.
12. Меняев М.Ф. Информатика и основы программирования: учеб. пособие / М.Ф. Меняев. – 3-е изд.стер. – М.: Омега-Л, 2007. – 458 с.
13. Методы программирования: учеб. пособие / Н.И. Минаков [и др.] -2-е изд. – М.: Вузовская книга, 2000. – 280 с.
14. Окулов С.М. Основы программирования / С.М. Окулов. – М.: ЮНИМЕДИАСТАЙЛ, 2002. – 424 с.
15. Фельдман С.Системное программирование на персональном компьютере / С. Фельдман.– 2-е изд. – М.: Новый издательский дом, 2004. – 512 с.
16. Шелест В. Д. Программирование: учеб. пособие / В. Д. Шелест. – СПб.: БХВ-Петербург, 2002. – 592 с.
17. Гамма Э., Хелм Р., Джонсон Р., Влиссидес Дж. Приемы объектно-ориентированного проектирования. Паттерны проектирования. –

СПб: Питер, 2001. – 368 с.

1. Чернецки К., Айзенекер У. Порождающее программирование. Методы, инструменты, применение. – Изд. дом «Питер».– М., С-Пет.–2005. – 730 с.
2. Римский Г.В. Структура и функционирование системы автоматизации модульного программирования // Программирование. – 1980. –

№2. – С.31–38.

1. Лаврищева Е.М., Грищенко В.Н.Структурное программирование. – Киев. – Наукова Думка, 1991. – 213с.

2. 7. Грищенко В.Н., Лаврищева Е.М. Методы и средства структурного программирования // Кибернетика и системный анализ. – 2003.

– №1. – С.39-55.

1. Лингер Р., Миллс Х., Уитт Б. Теория и практика структурного программирования.–М.: Мир, 1982.
2. Жоголев Е.А. Технология программирования. М: Научный мир, 2004
3. Борисенко В.В. Основы программирования: Учебное пособие, М: Интернет-университет информационных технологий, 2009
4. Грищенко В.Н., Лаврищева Е.М. Объектно -ориентированное программирование. Состояние, направления и перспективы развития – М.: 2007. – С.67–69.
5. Яковсон Айвар. Мечты о будущем программирования. Открытые системы. – М.: 2005. – №12. – С.59–63.
6. Летичевский А.А, Маринченко В.Г. Объекты в системе алгебраического программирования // Кибернетика и системный анализ. – 1997.

– №2. – С.160–180.

1. Летичевский А.А., Капитонова Ю.В., Волков В.А., Вышемирский В.В., Летичевский А.А. (мол.). Инсерционное программирование //

Там же.– 2003. – №1. – С.12-32.

1. Редько В.Н. Экспликативное программирование: ретроспективы и перспективы // Проб. программирования. – 1998. – №2. – С. 22– 41.
2. 18. Никитченко Н.С. Композиционно-номинативный подход к уточнению понятия программы // Там же. – 1999. – №1. – С.16–31.
3. Хендерсон П. Функциональное программирование. Применение и реализация. М.: Мир, 1983.
4. Бердж В. Методы рекурсивного программирования. М.: Машиностроение, 1983.
5. Летичевский А.А., Капитонова Ю.В. Доказательство теорем в математической информационной среде // Там же. – 1998. – №.4. –

С. 3 – 12