**Аннотация**

Целью данной курсовой работы, посвященной изучению высокоуровневых методов обработки информации и программирования, является:

систематизация и углубление теоретических и практических знаний по специальности «Прикладная информатика в экономике»;

углубление навыков программирования на языках высокого уровня;

обобщение опыта применения объектно-ориентированного подхода к технологии программирования;

применение полученных знаний при создании программного продукта, выполняющего расчет значения функций, заданных таблично.

По сравнению с традиционным функциональным подходом, вместо последовательных этапов цикла жизни программы решение задачи в объектно-ориентированном подходе представляется в виде иерархии уровней, начиная с требований пользователя и кончая сопровождением готовой программы. Все уровни разрабатываются в единой среде программирования с использованием одного языка.

При создании программного продукта используются высокоуровневые методы обработки информации и программирования, умение программировать на языках высокого уровня, в частности, на языке Pascal, на основе которого создана интегрированная среда разработки Delphi7.

В первой главе рассматриваются объектно-ориентированный подход к реализации задачи программирования. Вторая глава является проектной, в ней описаны алгоритмы разработанного программного продукта, внутренние и внешние связи между модулями программы, их взаимодействие. В приложениях даны исходный текст программы и блок-схема. Курсовая работа выполнена на  листах.

Результатом выполнения курсовой работы является создание программного средства, которое по введенному значению аргумента выводит значение функции, для этого использована формула интерполяционного полинома Лагранжа.

**Введение**

Преимущество написания программного продукта на языке высокого уровня Pascal заключается в том, что язык позволяет четко реализовать идеи структурного программирования и структурной организации данных, этот язык является процедурным. То есть программа явно описывает действия, которые необходимо выполнить, а результат задается способом получения его при помощи некоторой процедуры – определенной последовательности действий. Основными средствами, применяемыми в этом языке (как и в других процедурных языках) являются величины, присваивания, циклы, процедуры. При построении процедурной программы необходимо ясно представлять, какие действия и в какой последовательности будут производиться при ее выполнении. В отличие от функциональной программы, для построения которой требуются скорее математические, чем алгоритмические мышления.

При разработке программного средства используется объектно-ориентированный подход к технологии программирования, который является инженерно-конструкторской технологией, концентрирующей внимание на процессе решения задачи. Объектно-ориентированный подход предполагает, что при разработке программы должны быть определены классы используемых в программе объектов и построены их описания, затем созданы экземпляры необходимых объектов и определено взаимодействие между ними.

Основная задача (разработать программный продукт, вычисляющий значение функции по заданному значению аргумента) выполняется с помощью средств вычислительной математики. В основе решения задачи лежит вычисление значение функции с помощью интерполяционного полинома Лагранжа. Основные направления выполнения программы:

создать модули, предлагающие поэтапное выполнение поставленной задачи;

создать модули, выполняющие вычисление правильно, быстро и корректно;

разработать интерфейс программы, совместимый с Windows-интерфейсом.

**Глава 1. Эволюция технологии программирования**

На ранних этапах развития программирования, когда программы писались в виде последовательностей машинных команд, какая-либо технология программирования отсутствовала. Первые шаги в разработке технологии состояли в представлении программы в виде последовательности операторов. Написанию последовательности машинных команд предшествовало составление операторной схемы, отражающей последовательность операторов и переходы между ними. Операторный подход позволил разработать первые программы для автоматизации составления программ – так называемые составляющие программы.

С увеличением размеров программ стали выделять их обособленные части и оформлять их как подпрограммы. Часть таких подпрограмм объединялась в библиотеки, из которых подпрограммы можно было включать в рабочие программы и затем вызывать из рабочих программ. Это положило начало процедурному программированию – большая программа представлялась совокупностью процедур-подпрограмм. Одна из подпрограмм являлась главной и с нее начиналось выполнение программы.

В 1958 году были разработаны первые языки программирования, Фортран и Алгол-58. Программа на Фортране состояла из главной программы и некоторого количества процедур – подпрограмм и функций. Программа на Алголе-58 и его последующей версии Алголе-60 представляла собой единое целое, но имела блочную структуру, включающую главный блок и вложенные блоки подпрограмм и функций. Компиляторы для Фортрана обеспечивали раздельную трансляцию процедур и последующее их объединение в рабочую программу, первые компиляторы для Алгола предполагали, что транслируется сразу вся программа, раздельная трансляция процедур не обеспечивалась.

Процедурный подход потребовал структурирования будущей программы, разделения ее на отдельные процедуры. При разработке отдельной процедуры о других процедурах требовалось знать только их назначение и способ вызова. Появилась возможность перерабатывать отдельные процедуры, не затрагивая остальной части программы, сокращая при этом затраты труда и машинного времени на разработку и модернизацию программ.

Следующим шагом в углублении структурирования программ стало так называемое структурное программирование, при котором программа в целом и отдельные процедуры рассматривались как последовательности канонических структур: линейных участков, циклов и разветвлений. Появилась возможность читать и проверять программу как последовательный текст, что повысило производительность труда программистов при разработке и отладке программ. С целью повышения структурности программы были выдвинуты требования к большей независимости подпрограмм, подпрограммы должны связываться с вызывающими их программами только путем передачи им аргументов, использование в подпрограммах переменных, принадлежащих другим процедурам или главной программе, стало считаться нежелательным.

Процедурное и структурное программирование затронули прежде всего процесс описания алгоритма как последовательности шагов, ведущих от варьируемых исходных данных к искомому результату. Для решения специальных задач стали разрабатываться языки программирования, ориентированные на конкретный класс задач: на системы управления базами данных, имитационное моделирование и т.д.

При разработке трансляторов все больше внимания стало уделяться обнаружению ошибок в исходных текстах программ, обеспечивая этим сокращение затрат времени на отладку программ.

Применение программ в самых разных областях человеческой деятельности привело к необходимости повышения надежности всего программного обеспечения. Одним из направлений совершенствования языков программирования стало повышения уровня типизации данных. Теория типов данных исходит из того, что каждое используемое в программе данное принадлежит одному и только одному типу данных. Тип данного определяет множество возможных значений данного и набор операций, допустимых над этим данным. Данное конкретного типа в ряде случаев может быть преобразовано в данное другого типа, но такое преобразование должно быть явно представлено в программе. В зависимости от степени выполнения перечисленных требований можно говорить об уровне типизации того или иного языка программирования. Стремление повысить уровень типизации языка программирования привело к появлению языка Паскаль, который считается строго типизированным языком, хотя и в нем разрешены некоторые неявные преобразования типов, например, целого в вещественное. Применение строго типизированного языка при написании программы позволяет еще при трансляции исходного текста выявить многие ошибки использования данных и этим повысить надежность программы. Вместе с тем строгая типизация сковывала свободу программиста, затрудняла применение некоторых приемов преобразования данных, часто используемых в системном программировании. Практически одновременно с Паскалем был разработан язык Си, в большей степени ориентированный на системное программирование и относящийся к слабо типизированным языкам.

Все универсальные языки программирования, несмотря на различия в синтаксисе и используемых ключевых словах, реализуют одни и те же канонические структуры: операторы присваивания, циклы и разветвления. Во всех современных языках присутствуют предопределенные (базовые) типы данных (целые и вещественные арифметические типы, символьный и, возможно, строковый тип), имеется возможность использования агрегатов данных, в том числе массивов и структур (записей). Для арифметических данных разрешены обычные арифметические операции, для агрегатов данных обычно предусмотрена только операция присваивания и возможность обращения к элементам агрегата. Вместе с тем при разработке программы для решения конкретной прикладной задачи желательна возможно большая концептуальная близость текста программы к описанию задачи. Например, если решение задачи требует выполнения операций над комплексными числами или квадратными матрицами, желательно, чтобы в программе явно присутствовали операторы сложения, вычитания, умножения и деления данных типа комплексного числа, сложения, вычитания, умножения и обращения данных типа квадратной матрицы. Решение этой проблемы возможно несколькими путями:

Построением языка программирования, содержащего как можно больше типов данных, и выбором для каждого класса задач некоторого подмножества этого языка. Такой язык иногда называют языком-оболочкой. На роль языка-оболочки претендовал язык ПЛ/1, оказавшийся настолько сложным, что так и не удалось построить его формализованное описание. Отсутствие формализованного описания, однако, не помешало широкому применению ПЛ/1 как в Западной Европе, так и в СССР.

Построением расширяемого языка, содержащего небольшое ядро и допускающего расширение, дополняющее язык типами данных и операторами, отражающими концептуальную сущность конкретного класса задач. Такой язык называют языком-ядром. Как язык-ядро были разработаны языки Симула и Алгол-68, не получившие широкого распространения, но оказавшие большое влияние на разработку других языков программирования.

Дальнейшим развитием второго пути явился объектно-ориентированный подход к программированию.

**Различия функционального и объектно-ориентированного подходов программирования**

Есть два принципиально отличающихся подхода к технологии программирования: инженерно-конструкторский и математический. Первый основан на том, что изготовление программного продукта по заданным требованиям суть итеративный процесс последовательной реализации алгоритма решения задачи. Второй подход основан на математическом доказательстве правильности программы и автоматическом (или полуавтоматическом) получении готовой программы по заданным спецификациям.

Объектно-ориентированный подход к технологии программирования – инженерно-конструкторский, в котором внимание концентрируется на самом процессе решения задачи. В данной статье авторами обобщается опыт использования объектно-ориентированной технологии при разработке среды объектно-ориентированного программирования и некоторых прикладных программ, работающих в этой среде.

Оценим преимущества и недостатки наиболее распространенной инженерной технологии – функциональной. Существует три главные проблемы функционального подхода:

Слабая формализация связей между этапами. Специализация этапов приводит к тому, что интерфейсы между ними описываются на различных языках с разной степенью формализации. Слабая формализация уменьшает надежность программы и, что самое неприятное, ее адекватность требованиям заказчика.

Отсутствие гибкости. Поэтапная последовательная схема жестко связана с нисходящим способом создания программы. Однако практика показывает, что процесс. создания. хоть сколько-нибудь сложной программы неизбежно оказывается циклическим, с возвратами для внесения изменений в предыдущие этапы. Такие возвраты связаны с большими затратами, поскольку вовлекают этапы с разными интерфейсами и разными исполнителями.

Плохое использование результатов предыдущих разработок. Нисходящее проектирование вместе со слабой формализацией языков спецификаций приводят к тому, что результаты проектирования, как и готовые программы и их части, практически невозможно использовать для других целей. Исключение составляют библиотеки языков программирования на этапе кодирования.

Принципиальное отличие от функциональной схемы состоит в том, что поэтапный цикл жизни программного продукта заменяется многоуровневым представлением процесса решения задачи, которое получается сочетанием нисходящего и восходящего способов. С одной стороны идет детализация верхних уровней, с другой стороны из нижних уровней собираются недостающие компоненты верхних уровней. Смежные и другие близлежащие уровни могут пересекаться, т.е. иметь общие компоненты. Таким образом, границы между уровнями размываются, их число становится произвольным и даже неопределенным, а сами уровни теряют свою специфику.

Возможность применения вышеуказанной схемы обеспечивают три единые для всех уровней средства: объектно-ориентированный язык, среда программирования, база данных (библиотека классов). Эти средства в явном виде составляют основу любой объектно-ориентированной технологии. Рассмотрим их основные характеристики.

 Объектно-ориентированный язык программирования основывается на понятии объекта как замкнутой независимой сущности, взаимодействующей с внешним миром через строго определенный интерфейс в виде перечня сообщений, которые объект может принимать. Объект обладает свойствами, поведением и состоянием. Объекты с одинаковыми свойствами и поведением объединяются в классы. Программа на объектно-ориентированном языке представляет собой совокупность описаний классов. Классы, в свою очередь, представляют собой описания свойств и поведения составляющих их объектов. Свойства представляются другими, как правило, более простыми объектами. Поведение описывается обменивающимися сообщениями объектами.

Объектно-ориентированным языкам присущи следующие характеристики:

абстрактные типы данных;

скрытие реализации внешнего интерфейса (инкапсуляция);

наследование свойств и поведения объектов;

динамическое связывание имени со значением;

полиморфизм имен сообщений;

автоматическое управление памятью.

Чисто объектно-ориентированные языки, такие как Smalltalk, Eiffel, обладают всеми перечисленными здесь свойствами. Широко распространены объектно-ориентированные расширения традиционных языков (Си++, Турбо-Паскаль), в которых присутствуют не все объектно-ориентированные черты.

Объектно-ориентированная база классов представляет собой иерархический набор (библиотеку) классов – строительных блоков для сборки новых уровней создаваемого программного продукта. Здесь хранятся классы «джентльменского набора» библиотек языков программирования, а также классы, описывающие понятия прикладных предметных областей.

Классы представляют собой накопленные знания о ранее изготовленных продуктах. База классов должна обеспечивать легкий поиск классов, их изучение, модификацию, включение новых классов. Язык и база классов объединяются в интегрированную объектно-ориентированную среду программирования, в которой все средства поддержки процесса разработки программ написаны на данном языке и входят на равных правах в состав базы классов. Типичными примерами таких средств являются графический многооконный интерфейс с пользователем, иерархические меню, компилятор, интерпретатор, отладчик, текстовый редактор.

Разработка программ по объектно-ориентированной технологии сводится к созданию новых классов, их тестированию и включению в состав объектно-ориентированной среды разработки.

Каждый отдельный класс, благодаря своей независимости, может разрабатываться и тестироваться отдельно. Проверенный класс включается в библиотеку и становится достоянием текущей и последующих разработок. Особенностью объектно-ориентированной технологии разработки программ является возможность быстрого прототипирования. В этом случае классы решаемой задачи описываются только своими основными свойствами и поведением, что позволяет сразу проверить главную идею решения задачи. Если она неверна, то может быть быстро изменена и проверена снова. При этом экономятся существенные затраты на программирование деталей.

Рассмотрим, каким образом в объектно-ориентированной технологии преодолеваются три перечисленных недостатка функционального подхода.

Формализация связей между этапами. Все этапы (уровни), начиная анализа программы, описываются на едином формализованном языке. Фактически четкой границы между уровнями.

Единственный неформализованный уровень – требования пользователя. Этот уровень вообще трудно формализуем, поскольку конечным пользователем или заказчиком обычно является непрофессионал в области программирования и ему необходимо предложить свободу в изложении требований к задаче. То, что требуется для последующих спецификаций, – перечень объектов, их свойств и взаимных действий – не выходит за рамки обычного описания проблемы на естественном языке. Переход от этого описания к спецификации программы делается вручную, но практически один в один с точностью до простого синтаксиса объектно-ориентированного языка.

Гибкость. Внесение изменений внутри объекта или класса совершенно не затрагивает его клиентов, т.е. тех объектов, которые его используют, разумеется, если внешний интерфейс объекта остается неизменным. Это фундаментальный принцип объектно-ориентированного программирования. Таким образом обеспечивается гибкость при программировании снизу-вверх.

В настоящее время объектно-ориентированный подход является одним из быстро развивающихся направлений в проектировании систем. Примером могут являться объектно-ориентированный анализ – методология разработки систем, предложенная Йорданом, объектно-ориентированное проектирование, объектно-ориентированное программирование, реализованное в многочисленных компиляторах C++, Object Pascal, Borland Pascal, Smalltalk.

**Объектно-ориентированный подход**

Основные идеи объектно-ориентированного подхода опираются на следующие положения:

программа представляет собой модель некоторого реального процесса, части реального мира;

модель реального мира или его части может быть описана как совокупность взаимодействующих между собой объектов;

объект описывается набором параметров, значения которых определяют состояние объекта, и набором операций (действий), которые может выполнять объект;

взаимодействие между объектами осуществляется посылкой специальных сообщений от одного объекта к другому. Сообщение, полученное объектом, может потребовать выполнения определенных действий, например, изменения состояния объекта;

объекты, описанные одним и тем же набором параметров и способные выполнять один и тот же набор действий, представляют собой класс однотипных объектов.

С точки зрения языка программирования класс объектов можно рассматривать как тип данного, а отдельный объект – как данное этого типа. Определение программистом собственных классов объектов для конкретного набора задач должно позволить описывать отдельные задачи в терминах самого класса задач (при соответствующем выборе имен типов и имен объектов, их параметров и выполняемых действий).

Таким образом, объектно-ориентированный подход предполагает, что при разработке программы должны быть определены классы используемых в программе объектов и построены их описания, затем созданы экземпляры необходимых объектов и определено взаимодействие между ними.

Понятия объекта и класса тесно связаны. Тем не менее, существует важное различие между этими понятиями. Класс – это абстракция существенных характеристик объекта. Класс – описание множеств объектов, которые разделяют одинаковые свойства, операции, отношения и семантику (смысл). Любой объект – просто экземпляр класса. Различают внутренне представление класса (реализацию) и внешнее представление (интерфейс).

Значительное увеличение сложности задач, решаемых с помощью компьютеров, приводит к увеличению размеров и сложности программ, что порождает трудности при их разработке и отладке. Рассмотрение любой сложной системы требует применения техники декомпозиции – разбиения на составляющие элементы. Известны две схемы декомпозиции: алгоритмическая декомпозиция и объектно-ориентированная декомпозиция.

В основе алгоритмической декомпозиции лежит разбиение по действиям – алгоритмам. Эта схема представления применяется в обычных программных средствах. Объектно-ориентированная декомпозиция обеспечивает разбиение по автономным лицам – объектам реального (или виртуального) мира. Эти лица (объекты) – более «крупные» элементы, каждый из них несет в себе и описания действий, и описания данных.

Объектно-ориентированное представление программного средства основывается на принципах абстрагирования, инкапсуляции, модульности и иерархической организации. Каждый из этих принципов не нов, но их совместное применение рассчитано на проведение объектно-ориентированной декомпозиции. Это определяет модификацию их содержания и механизмов взаимодействия друг с другом.

Аппарат абстракции – удобный инструмент для борьбы со сложностью реальных систем. Создавая понятие в интересах какой-либо задачи, мы отвлекаемся (абстрагируемся) от несущественных характеристик конкретных объектов, определяя только существенные характеристики. Абстрагирование сводится к формированию абстракций. Каждая абстракция фиксирует основные характеристики объекта, которые отличают его от других видов объектов и обеспечивают ясные понятийные границы. Абстракция концентрирует внимание на внешнем представлении объекта, позволяет отделить основное в поведении объекта от его реализации. Абстракцию удобно строить путем выделения обязанностей объекта.

Инкапсуляция и абстракция – взаимодополняющие понятия: абстракция выделяет внешнее поведение  объекта, а инкапсуляция содержит и скрывает реализацию, которая обеспечивает это поведение. Инкапсуляция достигается с помощью информационной закрытости. Обычно скрываются структура объектов и реализация их методов. Инкапсуляция является процессом разделения элементов абстракции на секции с различной видимостью. Инкапсуляция служит для отделения интерфейса абстракции от ее реализации.

В языках С++, Object Pascal, Ada 95 абстракция классов и объектов формируют логическую структуру системы. При производстве физической структуры эти абстракции помещаются в модули. В больших системах, где классов сотни, модули помогают управлять сложностью. Модули служат физическими контейнерами, в которых объявляются классы и объекты логической разработки.

Модульность определяет способность системы подвергаться декомпозиции на ряд сильно связанных и слабо сцепленных модулей. Общая цель декомпозиции на модули: уменьшение сроков разработки и стоимости программного средства за счет выделения модулей, которые проектируются и изменяются независимо. Каждая модульная структура должна быть достаточно простой, чтобы быть полностью понятной. Изменение реализации модулей должно проводиться без знания реализации других модулей и без влияния на их поведение.

Определение классов и объектов выполняется в ходе логической разработки, а определение модулей – в ходе физической разработки системы. Эти действия сильно взаимосвязаны, осуществляются итеративно. Допустим, в Ada 95 мощным средством обеспечения модульности является пакет.

Прекрасным дополнением к предыдущим принципам является иерархическая организация – формирование из абстракций иерархической структуры. Определением иерархии в проекте упрощаются понимание проблем заказчика и их реализация – сложная система становится обозримой человеком. Иерархическая организация задает размещение абстракция на различных уровнях описания системы.

Двумя важными инструментами иерархической организации в объектно-ориентрованных системах являются:

структура из классов(«is a»-иерархия);

структура из объектов(«part of»-иерархия).

Чаще всего «is a»-иерархическая структура строится с помощью наследования. Наследование определяет отношение между классами, где класс разделяет структуру или поведение, определенные в одном другом (единичное наследование) или в нескольких других (множественное наследование) классах. Другая разновидность иерархической организации – «part of»-иерархическая структура – базируется на отношении агрегации. Агрегация не является понятием, уникальным для объектно-ориентированных систем. Например, любой язык программирования, разрешающий структуры типа «запись», поддерживает агрегацию. И все же агрегация особенно полезна в сочетании с наследованием: агрегация обеспечивает физическую группировку логически связной структуры, а наследование позволяет легко и многократно использовать эти общие группы в других абстракциях.

Рассмотрим более пристально объекты – конкретные сущности, которые существуют во времени и пространстве.

Объект – это конкретное представление абстракции. Объект обладает индивидуальностью, состоянием и поведением. Структура и поведение подобных объектов определены в их общем классе. Термины «экземпляры класса» и «объект» взаимозаменяемы.

Индивидуальность – это характеристика объекта, которая отличает его от всех других объектов.

Состояние объекта характеризуется перечнем всех свойств объекта и текущими значениями каждого их этих свойств. Объекты не существуют изолированно друг от друга. Они подвергаются воздействию или сами воздействую на другие группы. Поведение характеризует то, как объект воздействует на другие объекты (или подвергается воздействию) в терминах изменений его состояния и передачи сообщений. Поведение объекта является функцией как его состояния, так и выполняемых им операций. Говорят, что состояние объекта представляет суммарный результат его поведения. Операция обозначает обслуживание, которое объект предлагает своим клиентам. Возможны пять видов операций клиента над объектом:

модификатор (изменяет состояние объекта);

селектор (дает доступ к состоянию, но не изменяет его);

итератор (доступ к содержанию объекта по частям, в строго определенном порядке);

конструктор (создает объект и инициализирует его состояние);

деструктор (разрушает объект и освобождает занимаемую им память).

В чистых объектно-ориентированных языках программирования операции могут объявляться только как методы – элементы классов, экземплярами которых являются объекты. Гибридные языки позволяют писать операции как свободные подпрограммы (вне классов). В общем случае все методы и свободные подпрограммы, ассоциированные с конкретным объектом, образуют протокол. Таким образом, протокол определяет оболочку допустимого поведения объекта и поэтому заключает в себе цельное (статическое и динамическое) представление объекта. Большой протокол полезно разделять на логические группировки, разделяющие пространство поведения объект, обозначают роли, которые может играть объект. С точки зрения внешней среды важное значение имеет такое понятие, как обязанности объекта. Обязанности означают обязательства объекта обеспечить определенное поведение. Обязанностями объекта являются все виды обслуживания, которые он предлагает клиентам. В мире объект играет определенные роли, выполняя свои обязанности.

Наличие у объекта внутреннего состояния означает, что порядок выполнения им операций очень важен. То есть объект может представляться как независимый автомат. Можно выделить активные и пассивные объекты. Активный объект имеет собственный канал (поток) управления, пассивный – нет. Активный объект автономен, он может проявлять свое поведение без воздействия со стороны других объектов. Пассивный объект, наоборот, может изменять свое состояние только под воздействием других объектов.

В поле зрения разработчика программного обеспечения находятся не объекты-одиночки, а взаимодействующие объекты, ведь именно взаимодействие объектов реализует поведение системы. Отношения между парой объектов основываются на взаимной информации о разрешенных операциях и ожидаемом поведении.

**Оценка сложности программных систем**

При конструировании объектно-ориентированных программных средств значительная часть затрат приходится на создание визуальных моделей. Очень важно корректно и всесторонне оценить качество этих моделей, сопоставив качеству числовую оценку. Решение данной задачи требует введения специального метрического аппарата. Такой аппарат развивает идеи классического оценивания сложных программных систем, основанного на метриках сложности, связности и сцепления. Вместе с тем он учитывает специфические особенности объектно-ориентированных решений. Объектно-ориентированные метрики вводятся с целью:

улучшить понимание качества продукта;

оценить эффективность процесса конструирования;

улучшить качество работы на этапе проектирования.

Для любого инженерного продукта метрики должны ориентироваться на его уникальные характеристики. С точки зрения метрик выделяют пять характеристик объектно-ориентированных систем: локализацию, инкапсуляцию, информационную закрытость, наследование и способы абстрагирования объектов. Эти характеристики оказывают максимальное влияние на объектно-ориентированные метрики.

Локализация фиксирует способ группировки информации в программе. В классических методах, где используется функциональная декомпозиция, информация локализуется вокруг функций. Функции в них реализуются как процедурные модули. В методах, управляемых данными, информация группируется вокруг структур данных. В объектно-ориентированной среде информация группируется внутри классов или объектов (инкапсуляция, как данных, так и процессов). Поскольку в классических методах основной механизм локализации – функция, программные метрики ориентированы на внутреннюю структуру или сложность функций (длина модуля, связность, цикломатическая сложность) или на способ, которым функции связываются друг с другом (сцепление модулей). Так как в объектно-ориентированной системе базовым элементом является класс, то локализация здесь основывается на объектах. Поэтому метрики должны применяться к классу (объекту) как к комплексной сущности. Кроме того, между операциями (функциями) и классами могут быть отношения не только «один-к-одному». Поэтому метрики, отображающие способы взаимодействия классов, должны быть приспособлены к отношениям «один-ко-многим», «многие-ко-многим».

Инкапсуляция – упаковка (связывание) совокупности элементов. Для классических программных средств примерами низкоуровневой инкапсуляции являются записи и массивы. Механизмом инкапсуляции среднего уровня являются подпрограммы. В объектно-ориентированных системах инкапсулируются обязанности класса, представляемые его свойствами, операциями и состояниями. Для метрик учет инкапсуляции приводит к смещению фокуса измерений с одного модуля на группу свойств и обрабатывающих модулей (операций). Кроме того, инкапсуляция переводит измерения на более высокий уровень абстракции. Напротив, классические метрики ориентированы на низкий уровень – количество булевых условий (цикломатическая сложность) и количество строк программы.

Информационная закрытость делает невидимыми операционные детали программного компонента. Другим компонентам доступна только необходимая информация. Качественные объектно-ориентированные системы поддерживают высокий уровень информационной закрытости. Таким образом, метрики, измеряющие степень достигнутой закрытости, тем самым отображают качество объектно-ориентированного проекта.

Наследование – механизм, обеспечивающий тиражирование обязанностей одного класса в другие классы. Наследование распространяется через все уровни иерархии классов. Стандартные программные средства не поддерживают эту характеристику. Поскольку наследование – основная характеристика объектно-ориентированных систем, на ней фокусируются многие объектно-ориентированные метрики (количество детей – потомков класса, количество родителей, высота класса в иерархии наследования).

Абстракция – это механизм, который позволяет проектировщику выделять главное в программном компоненте (как свойства, так и операции) без учета второстепенных деталей. По мере перемещения на более высокие уровни абстракции мы игнорируем все большее количество деталей, обеспечивая все общее представление понятия или элемента. По мере перемещения на более низкие уровни абстракции мы вводим все большее количество деталей, обеспечивая более точное представление понятия или элемента. Класс – это абстракция, которая может быть представлена на различных уровнях детализации и различными способами. Поэтому объектно-ориентированные метрики должны представлять абстракции в терминах измерений класса.

Классической мерой сложности внутренних связей модуля является связность, а классической мерой сложности внешних связей – сцепление.

Связность модуля – мера зависимости его частей, это внутренняя характеристика модуля. Чем выше связность модуля, тем лучше результат проектирования. Для измерения связности используют понятие силы связности (СС). Существует 7 типов связности:

Связность по совпадению (СС=0). В модуле отсутствуют явно выраженные внутренние связи.

Логическая связность (СС=1). Части модуля объединены по принципу функционального подобия. Например, модуль состоит из разных подпрограмм обработки ошибок. При использовании такого модуля клиент выбирает только одну из подпрограмм. Недостатками является сложное сопряжение и большая вероятность внесения ошибок при изменении сопряжения ради одной функции.

Временная связность (СС=3). Части модуля не связаны, но необходимы в один и тот же период работы системы. Недостаток: сильная взаимная связь с другими модулями, отсюда – сильная чувствительность к внесению изменений.

Процедурная связность (СС=5). Части модуля связаны порядком выполняемых ими действий, реализующих некоторый сценарий поведения.

Коммуникативная связность (СС=7). Части модуля связаны по данным (работают с одной и той же структурой данных).

Информационная (последовательная) связность (СС=9). Выходные данные одной части используются как входные данные в другой части модуля.

Функциональная связность (СС=10). Части модуля вместе реализуют одну функцию.

Этот метод функционален по своей природе, поэтому наибольшей связностью здесь объявлена функциональная связность. Вместе с тем одним из принципиальных преимуществ объектно-ориентированного подхода является естественная связность объектов. Максимально связанным является объект, в котором представляется единая сущность и в который включены все операции над этой сущностью. Следовательно, восьмой тип можно определить так:

Объектная связность. Каждая операция обеспечивает функциональность, которая предусматривает, что все свойства объекта будут модифицироваться, отображаться и использоваться как базис для предоставления услуг.

Высокая связность – желательная характеристика, так как она означает, что объект представляет единую часть в проблемной области, существует в едином пространстве. При изменении системы все действия над частью инкапсулируются в едином компоненте. Поэтому для производства изменения нет нужды модифицировать много компонентов. Если функциональность в объектно-ориентированной системе обеспечивается наследованием от суперклассов, то связность объекта, который наследует свойства и операции, уменьшается. В этом случае нельзя рассматривать объект как отдельный модуль – должны учитываться все его суперклассы. Системные средства просмотра содействуют такому учету. Однако понимание элемента, который наследует свойства от нескольких суперклассов, резко усложняется.

В классическом методе Л. Констентайна и Э. Йордана определены шесть типов сцепления, которые ориентированы на процедурное проектирование. Принципиальное преимущество объектно-ориентированного проектирования в том, что природа объектов приводит к созданию слабо сцепленных систем. Фундаментальное свойство объектно-ориентированного проектирования заключается в скрытости содержания объекта. Как правило, содержание объекта невидимо внешним элементам. Степень автономности объекта достаточно высока. Любой объект может быть замещен другим объектом с таким же интерфейсом. Тем не менее наследование в объектно-ориентированных системах приводит к другой форме сцепления. Объекты, которые наследуют свойства и операции, сцеплены с их суперклассами. Изменения в суперклассах должны проводиться осторожно, так как эти изменения распространяются во все классы, которые наследуют их характеристики.

Таким образом, сами по себе объектно-ориентированные механизмы не гарантируют минимального сцепления. Конечно, классы – мощное средство абстракции данных. Их введение уменьшило поток данных между модулями и, следовательно, снизило общее сцепление внутри системы. Однако количество типов зависимостей между модулями выросло. Появились отношения наследования, делегирования, реализации и т.д. Более разнообразным стал состав модулей в системе (классы, объекты, свободные функции и процедуры, пакеты). Отсюда вывод: необходимость измерения и регулирования сцепления в объектно-ориентированных системах обострилась.

Сцепление – мера зависимости модулей по данным, внешняя характеристика модуля, которую желательно уменьшать. Количественно сцепление измеряется степенью сцепления (СЦ). Выделяют 6 типов сцепления:

Сцепление по данным (СЦ=1). Модуль А вызывает модуль В. Все входные и выходные параметры вызываемого модуля – простые элементы данных.

Сцепление по образцу (СЦ=3). В качестве параметров используются структуры данных.

Сцепление по управлению (СЦ=4). Модуль А явно управляет функционированием модуля В (с помощью флагов или переключателей), посылая ему управляющие данные.

Сцепление по внешним ссылкам (СЦ=5). Модули А и В ссылаются на один и тот же глобальный элемент данных.

Сцепление по общей области (СЦ=7). Модули разделяют одну и ту же глобальную структуру данных.

Сцепление по содержанию (СЦ=9). Один модуль прямо ссылается на содержание другого модуля (не через точку входа). Например, коды их программ перемежаются друг с другом.

**Глава 2. Интегрированная среда разработки Delphi**

Курсовой проект был разработан в интегрированной среде разработки Delphi 7, далее последует описание данного программного изделия.

Прикладные программы, или приложения, Delphi создаются в интегрированной среде разработки. Пользовательский интерфейс этой среды служит для организации взаимодействия с программистом и включает в себя ряд окон, содержащих различные элементы управления. С помощью средств интегрированной среды разработчику удобно проектировать интерфейсную часть приложения, а также писать программный код и связывать его с элементами управления. В интегрированной среде разработки проходят все этапы создания приложения, включая отладку. Интегрированная среда разработки Delphi 7 представляет собой многооконную систему.

Язык Delphi реализует концепцию объектно-ориентированного программирования. Это означает, что функциональность приложения определяется набором взаимосвязанных задач, каждая из которых становится самостоятельным объектом. У объекта есть свойства (т.е. характеристики, или атрибуты), методы, определяющие его поведение, и события, на которые он реагирует. Одним из наиболее важных понятий объектно-ориентированного программирования является класс. Класс представляет собой дальнейшее развитие концепции типа и объединяет в себе задание не только структуры и размера переменных, но и выполняемых над ним операций. Объекты в программе всегда являются экземплярами того или иного класса.

Все вычисления курсовой программы построены на знании интерполяционного полинома Лагранжа (интерполяционный многочлен в форме Лагранжа). Именно с его помощью вычисляется значение функции при нескольких заданных пользователем значениях аргумента и некоторым другим входным данным. Интерполяционным многочленом называется многочлен https://uchil.net/images/27/34/5662734.png степени не выше n, который в узлах https://uchil.net/images/27/35/5662735.pngпринимает значения https://uchil.net/images/27/36/5662736.png.

Условия применения полинома:

Функция https://uchil.net/images/27/37/5662737.png непрерывна на интервале https://uchil.net/images/27/38/5662738.png и задана некоторыми своими значениями https://uchil.net/images/27/39/5662739.png для соответствующих значений аргумента https://uchil.net/images/27/40/5662740.png. Полином используется, когда необходимо найти значение этой функции в точке https://uchil.net/images/27/41/5662741.png.

Теорема:

Пусть заданы узлы https://uchil.net/images/27/42/5662742.png, среди которых нет совпадающих и значения https://uchil.net/images/27/36/5662736.png некоторой функции f(x) в этих узлах. Существует один и только один многочлен https://uchil.net/images/27/43/5662743.png степени не выше n, принимающий в узлах https://uchil.net/images/27/44/5662744.png значения https://uchil.net/images/27/45/5662745.png. Введем вспомогательные многочлены

https://uchil.net/images/27/46/5662746.png.

Это многочлен степени n, для которого выполняется равенство https://uchil.net/images/27/47/5662747.png.

Многочлен https://uchil.net/images/27/48/5662748.png называется полиномом Лагранжа.

При применении программы, названной «Интерполирование», пользователь в первом окне вводит количество аргументов, столько узлов https://uchil.net/images/27/49/5662749.png будет в дальнейшем использоваться программой. Во втором окне пользователь вводит значения аргументов и функций для построения полинома Лагранжа. В третьем окне задается значение аргумента, функцию которого необходимо найти. При нажатии кнопки «Вычислить», приводится в действие алгоритм программы, происходит вычисление и выдается результат и одновременно с этим процессом программа находит погрешность вычисления. Пользователь сам может оценить, насколько полученный в ходе вычислений результат расходится с правильным, используя погрешность.

**Характеристики модулей программного средства**

Связность внутри модулей

Модули 1, 2 и 4 имеют внутреннюю связность временную (части модуля не связаны, но необходимы в один и тот же период работы системы). Третий модуль имеет  тип связности – информативный последовательный, который говорит о том, что выходные данные одной части модуля используются как входные данные другой части модуля. Эта связи присутствует, потому что function Ln (k: Integer): Real;  function Pol ( ): Real; и function Delta ( ): Real; являются входными данными для Procedure TForm3/Button1Click (Sender: Tobject).

Сцепление модулей между собой

Модуль 1 сцеплен с модулем 2 по данным (тип сцепления первый, СЦ=1), то есть модуль 1 вызывает модуль 2, входные и выходные параметры – простые элементы данных.

Модуль 3 с модулем 2 сцеплен по содержанию, то есть модуль 3 прямо ссылается на содержание модуля 2. Модуль 3 с первым модулем не сцеплены, так как, хотя они оба сцеплены со вторым модулем, но каждый из них имеет разный тип сцепления, не имеющих ничего общего.

Четвертый модуль, являющийся сведениями о программе связан с модулями 1, 2 и 3 по данным, все три модуля вызывают четвертый.

**Заключение**

Во время выполнения курсовой работы был разработан программное средство, вычисляющее значение функции по заданному значению аргумента, выполняющегося с помощью средств вычислительной математики. В основе решения задачи, как было описано во второй части курсовой работы лежит вычисление интерполяционным полиномом Лагранжа. Было создано четыре модуля, три из которых являются активными, четвертый («О программе») может быть вызван с любого из трех главных модулей. Три основных модуля предлагают поэтапное выполнение задачи; в первом модуле вводятся начальные данные, во втором заполняется таблица со значениями функций и аргументов (это промежуточные точки, на основании которых и строится полином Лагранжа). Третий модуль является основным, использующим данные из второго и третьего модуля для вычислительного процесса. Интерфейс программы совместим с Windows-интерфейсом благодаря тому, что программа, в которой разрабатывалась курсовая работа, – Delphi7 – предоставляет такую возможность и поддерживает именно Windows-интерфейс. Все пункты курсового задания были выполнены, разработанное программное средство выполняет поставленную перед ним задачу.

1. Фокс Дж. Программное обеспечение и его разработка. М.:Мир, 1985. - 368 с.

2. Лисков Б., Гатэг Дж. Использование абстракций и спецификаций при разработке программ: Пер. с англ. - М.: Мир, 1999. - 424 с.

Иванов А.Г., Карпова А.В., Семик В.П., Филинов Ю.Е. Объектно-ориентированная среда программирования. Системы и средства информатики. Вып.2. М.: Наука, 1991.

Программирование. Специальный выпуск по объектно-ориентированному программированию. № 6, 1990.

Объектно-ориентированный подход технологии программирования/ Иванов А.Г., Пятницкий А.А, Филинов Ю.Е. – СПб.: Питер, 2003. – 443 с.: ил.

Макаров А.А. Лукашин Ю.П. Объектно-ориентированный подход в программировании. – М.: Финансы и статистика, 2000. – 356 с.