ВВЕДЕНИЕ

Курсовая работа - это самостоятельный исследовательский труд студента, выполненный на основе использования научной, учебной, методической литературы и прочих учебно-методических материалов. При подготовке и написании курсовой работы перед студентами ставятся следующие задачи:

* Приобрести навыки самостоятельной работы с литературой по выбранной тематике;
* Систематизировать и закрепить теоретические знания по дисциплине;
* Развить навыки решения практических задач;
* Уметь связно излагать материал, касающийся выбранной проблематики;
* Самостоятельно производить необходимые виды анализа для разработки программ.

Курсовая работа выполняется каждым студентом в соответствии с полученным индивидуальным заданием и представляется на рецензирование с последующей ее устной защитой и оценкой.

Целью курсовой работы является закрепление знаний и навыков, полученных в процессе изучения дисциплины. Получение практических навыков по реализации программных средств на языке программирования Си.

Программирование можно рассматривать как искусство, науку, ремесло. Программирование — это искусство получения ответов от машины. Для этого в узком смысле нужно составить специальный код для технического устройства, а в широком — разработать программы на языках программирования, т. е. не просто составить код, а выполнить интеллектуальную работу по составлению высокоразумных программ для решения различных задач во всех сферах человеческой деятельности.

Программирование — процесс описания последовательности действий решения задачи средствами конкретного языка программирования и оформление результатов описания в виде программы. Эта работа требует точности, аккуратности и терпения. Команды машине должны формулироваться абсолютно четко и полно, не должны содержать никакой двусмысленности.

Уровень программирования определяется четырьмя взаимосвя­занными факторами развития: возможностями компьютеров, теори­ей и языками, искусством и технологией программирования.

1. История и этапы развития программирования

С глубокой древности известны попытки создать устройства, ус­коряющие и облегчающие процесс вычислений. Еще древние греки и римляне применяли приспособление, подобное счетам, — абак. Такие устройства были известны и в странах Древнего Востока. В XVM в. немецкие ученые В. Шиккард (1623), Г.Лейбниц (1673) и французский ученый Б. Паскаль (1642) создали механические вы­числительные устройства — предшественники всем известного арифмометра. Вычислительные машины совершенствовались в те­чение нескольких веков. Но при этом не применялось понятие «программа и программирование».

В 1854 г. английский математик Джордж Буль опубликовал кни­гу «Законы мышления», в которой развил алгебру высказываний —Булеву алгебру. На ее основе в начале 80-х гг. XIX в. построена тео­рия релейно-контактных схем и конструирования сложных дискрет­ных автоматов. Алгебра логики оказала многогранное влияние на развитие вычислительной техники, являясь инструментом разработ­ки и анализа сложных схем, инструментом оптимизации большого числа логических элементов, из многих тысяч которых состоит со­временная ЭВМ.

Идеи Ч. Бэббиджа реализовал американский ученый Г. Холлерит, который с помощью построенной счетно-аналитической маши­ны и перфокарт за три года обработал результаты переписи населе­ния в США по состоянию на 1890 г. В машине впервые было ис­пользовано электричество. В 1896 г. Холлеритом была основана фирма по выпуску вычислительных перфорационных машин и пер­фокарт.

В 1936 г. английский математик А. Тьюринг ввел понятие ма­шины Тьюринга, как формального уточнения интуитивного понятия алгоритма. Ученый показал, что любой алгоритм в некотором смысле может быть реализован на машине Тьюринга, а следователь­но, доказывал возможность построения универсальной ЭВМ. И та, и другая машины аналогично могут быть снабжены исходными данными решаемой задачи и программой ее решения. Машину Тьюринга можно считать как бы идеализированной моделью универ­сальной ЭВМ.

В 40-х гг. XX в. механическая элементная база вычислительных машин стала заменяться электрическими и электронными устройствами. Первые электромеханические машины были созданы в Германии К. Цузе (Ц-3, 1941 г.) и в США под руководством профессора Гарвардского университета Г. Айкена (МАРК-1, 1944 г.). Первая электронная машина создана в США группой инженеров под руководством доктора Пенсильванского университета Дж. Мочли и аспиранта Дж. Экксрта (ЭНИАК — электронный числовой интегратор и калькулятор, 1946 г.). В 1949 г. в Англии была построена EDSAC — первая машина, обладающая автоматическим программным управ­лением, внутренним запоминающим устройством и другими необхо­димыми компонентами современных ЭВМ.

Логические схемы вычислительных машин были разработаны в конце 1940-х гг. Дж. фон Нейманом, Г. Гольдстайном и А. В. Берксом. Особый вклад в эту работу внес американский математик Джон фон Нейман, принимавший участие в создании ЭНИАК. Он предложил идею хранения команд управления и данных в машин­ной памяти и сформулировал основные принципы построения современных ЭВМ. ЭВМ с хранимой программой оказались более быстродействующими и гибкими, чем ранее созданные.

В 1951 г. в США было налажено первое серийное производство электронных машин УНИВАК (универсальная автоматическая вы­числительная машина). В это же время фирма IBM начала серий­ный выпуск машины IBM/701.

В СССР первыми авторами ЭВМ, изобретенной в декабре 1948 г., являются И. С. Брук и Б. И. Рамеев. А первая советская ЭВМ с сохраняющейся программой создана в 1951 г. под руково­дством С. А Лебедева (МЭСМ — малая электронная счетная маши­на). В 1953 г. в Советском Союзе начался серийный выпуск машин, первыми их которых были БЭСМ-1, «Стрела».

С появлением цифровых программно-управляемых машин родилась новая область прикладной математики — программирование. Как область науки и профессия она возникла в 1950-х гг. Первона­чально программы составлялись вручную на машинных языках (в машинных кодах). Программы были громоздки, их отладка — очень трудоемка. Для упрощения приемов и методов составления и отладки программ были созданы мнемокоды, по структуре близкие к машинному языку и использующие символьную адресацию. Ассемблеры переводили программу, записанную в мнемокоде, на машинный язык и, расширенные макрокомандами, используются и в настоящее время. Далее были созданы автокоды, которые можно применять на различных машинах, и позволившие обмениваться программами. Автокод — набор псевдокоманд для решения специализиро­ванных задач, например, научных или инженерных. Для таких задач имеется развитая библиотека стандартных программ.

До конца 1950-х гг. ЭВМ основным элементом конструкции были электронные лампы (1-е поколение). В этот период развитие идеологии и техники программирования шло за счет достижений американских ученых Дж. фон Неймана, сформулировавшего ос­новные принципы построения ЭВМ, и Дж. Бэкуса, под руково­дством которого в 1954 г. был создан Fortran (Formula Translation) — первый язык программирования высокого уровня, используемый до настоящего времени в разных модификациях. Так, в 1965 г. в Дартмутском колледже Д. Кэмэни и Т. Куртцем была разработана упро­щенная версия Фортрана — Basic. В 1966 г. комиссия при Амери­канской ассоциации стандартов (ASA) разработала два стандарта языка: Фортран и Базисный Фортран. Используются также даль­нейшие модификации языка (например 1970, 1990 гг.).

В США в 1954 г. стал применяться алгебраический подход, совпадающий, по существу, с операторным методом. В 1956 г. корпора­цией IBM разработана универсальная ПП Фортран для автоматического программирования на ЭВМ IBM/704.

Усложнение структуры ЭВМ привело (в 1953 г. для машин И-го поколения) к созданию операционных систем (ОС) — специальных управляющих программ для организации и решения задач на ЭВМ. Например, мониторная система МТИ, созданная в Массачусетском технологическом институте, обеспечивала пакетную обработку, т. е. непрерывное, последовательное прохождение через ЭВМ многих групп (пакетов) заданий и пользование библиотекой служебных программ, хранимой в машине. Это позволило совместить операции по запуску с выполнением программ.

Для ПЭВМ к настоящему времени разработаны ОС: MS DOS, Windows, ОС/2, МасОС, Unix, Linux и др. Широкое распростране­ние получили ОС MS DOS и Windows, имеющие развитый интер­фейс и широкий набор приложений, позволяющих последователь­ное выполнение заданий из пакета, обработку различной информа­ции во многих сферах человеческой деятельности.

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 методология, определяющая совокупность методов, используемых на разных этапах разработки.

1 этап - «стихийное» программирование. Первые программы имели простейшую структуру, состояли из программы на машинном языке и обрабатываемых ею данных. Сложность программ в машинных кодах ограничивалась способностью программиста одновременно мысленно отслеживать последовательность выполняемых операций и местонахождение данных при программировании.

Появление ассемблеров. Программы стали более «читаемыми».

Создание языков программирования высокого уровня существенно упростило программирование вычислений, снизив уровень детализации операций. Это, в свою очередь, позволило увеличить сложность программ.

2 этап - структурный подход к программированию - представляет собой совокупность рекомендуемых технологических приемов, охватывающих выполнение всех этапов разработки ПО. В основе – декомпозиция сложных систем с целью последующей реализации в виде отдельных небольших подпрограмм. С появлением других принципов декомпозиции данный способ получил название процедурной декомпозиции. Структурный подход требовал представления задачи в виде иерархии подзадач простейшей структуры. Проектирование осуществлялось «сверху – вниз» и подразумевало реализацию общей идеи, обеспечивая проработку интерфейсов подпрограмм.

Из-за роста сложности и размеров разрабатываемого ПО потребовалось развитие структурирования данных. Следовательно, в языках появляется возможность определения пользовательских типов данных. - появилась и начала развиваться технология модульного программирования. Модульное программирование -выделение групп подпрограмм, использующих одни и те же глобальные данные в отдельно компилируемые модули (библиотеки подпрограмм), Связи между модулями при использовании данной технологии осуществляются через специальный интерфейс, в то время как доступ к реализации модуля запрещен.

3 этап - объектный подход к программированию (с середины 80-х до конца 90-х годов XX в.). Объектно-ориентированное программирование - технология создания сложного ПО, основанная на представлении программы в виде совокупности объектов, каждый из которых является экземпляром определенного типа (класса), а классы образуют иерархию с наследованием свойств. Взаимодействие программных объектов в такой системе осуществляется путем передачи сообщений.

Были созданы среды, поддерживающие визуальное программирование. При использовании визуальной среды появляется возможность проектировать некоторую часть, например, интерфейсы будущего продукта, с применением визуальных средств добавления и настройки специальных библиотечных компонентов.

4 этап –компонентный подход и CASE-технологии. Компонентный подход - построение ПО из отдельных компонентов – физически отдельно существующих частей программного обеспечения, которые взаимодействуют между собой через стандартизованные двоичные интерфейсы. В отличие от обычных объектов объекты-компоненты можно собрать в динамически вызываемые библиотеки или исполняемые файлы, распространять в двоичном виде (без исходных текстов) и использовать в любом языке программирования, поддерживающем соответствующую технологию.

Отличительной особенностью современного этапа развития технологии программирования, кроме изменения подхода, является создание и внедрение автоматизированных технологий разработки и сопровождения программного обеспечения, которые были названы CASE-технологиями. На сегодня существуют CASE-технологии, поддерживающие как структурный, так и объектный подходы к программированию.

Появление нового подхода не означает, что отныне все программное обеспечение будет создаваться из программных компонентов, но анализ существующих проблем разработки сложного программного обеспечения показывает, что он будет применяться достаточно широко.

1. Классификация языков программирования и сред разработки

Сегодня можно насчитать более 2 тыс. различных языков программирования и их модификаций, однако лишь отдельные получили широкое признание. Все языки программирования можно условно классифицировать по некоторым основным признакам. Ниже приведена краткая классификация языков программирования:

По назначению:

* процедурные;
* непроцедурные.

Процедурная (алгоритмическая) программа — это система формальных предписаний, направленных на решение конкретных задач, которые выполняет ЭВМ.

Непроцедурное программирование представляет собой прямо противоположную методологию (парадигму) разработки, когда компьютеру ставится определённая задача в более или менее общем виде, без написания формализованного алгоритма, который отдаётся на усмотрение машины.

Процедурные языки отличаются тем, на кого в первую очередь направлены: на машину или человека. Они подразделяются на две категории:

* низкого уровня (или машинно-ориентированные);
* высокого уровня.

Низкоуровневые языки ориентированы на конкретные компьютерные архитектуры и учитывают их особенности. Они являются следующим уровнем после машинного кода и следуют конкретным указаниям, исходящим от процессора.

К языкам низкого уровня относится программирование в машинных кодах, ассемблер, макроассемблер.

Высокоуровневые языки более понятны для человека. Соответственно, программировать на них гораздо проще и удобнее. В них не учитываются особенности конкретных типов процессоров, поэтому такие программы легко переносятся с одной платформы на другую.

Языки высокого уровня делятся на универсальные и проблемно-ориентированные.

Наиболее распространенные универсальные языки C#, C++, Basic, Pascal (Delphi) используются для разработки Windows-приложений. Большой вклад в программирование на начальных этапах внесли языки Fortran, Cobol, Algol, C и др.

Непроцедурные языки включают две основные языковые группы:

* объектно-ориентированные;
* декларативные.

Объектно-ориентированные состоят из ряда независимых объектов, которые функционируют как отдельные компьютеры. С помощью этих блоков можно решать задачи, не вникая во «внутреннюю кухню» их работы.  
 Работа с декларативным языком подразумевает установление взаимосвязей между исходными информационными структурами и свойствами конечного результата. При этом в нём не существует понятия «команда», а программист не создаёт алгоритмы.

Декларативные языки подразделяются на два семейства:

* логические;
* функциональные.

Логическое программирование описывает проблемы в виде фактов и формул, а система решает их посредством механизмов логического вывода.

Функциональное, в свою очередь, формулирует задачу как совокупность определённых функций.

При разработке программных средств используется в той или иной мере компьютерная поддержка процессов разработки и сопровождения ПС [16.1]. Это достигается путем представления хотя бы некоторых программных документов ПС (прежде всего, программ) на компьютерных носителях данных (например, на дискетах) и предоставлению в распоряжение разработчика ПС специальных ПС или включенных в состав компьютера специальных устройств, созданных для какой-либо обработки таких документов. В качестве такого специального ПС можно указать компилятор с какого-либо языка программирования. Компилятор избавляет разработчика ПС от необходимости писать программы на языке компьютера, который для разработчика ПС был бы крайне неудобен, - вместо этого он составляет программы на удобном ему языке программирования, которые соответствующий компилятор автоматически переводит на язык компьютера. В качестве специального устройства, поддерживающего процесс разработки ПС, можно указать, например, эмулятор какого-либо языка. Эмулятор позволяет выполнять (интерпретировать) программы на языке, отличном от языка компьютера, поддерживающего разработку ПС, например, на языке компьютера, для которого эта программа предназначена.

ПС, предназначенное для поддержки разработки других ПС, будем называть программным инструментом разработки ПС, а устройство компьютера, специально предназначенное для поддержки разработки ПС, будем называть аппаратным инструментом разработки ПС.

Инструменты разработки ПС могут использоваться в течение всего жизненного цикла ПС для работы с разными программными документами. Так текстовый редактор может использоваться для разработки практически любого программного документа. С точки зрения функций, которые инструменты выполняют при разработке ПС, их можно разбить на следующие четыре группы:

* редакторы,
* анализаторы,
* преобразователи,
* инструменты, поддерживающие процесс выполнения программ.

Редакторы поддерживают конструирование (формирование) тех или иных программных документов на различных этапах жизненного цикла. Как уже упоминалось, для этого можно использовать один какой-нибудь универсальный текстовый редактор. Однако, более сильную поддержку могут обеспечить специализированные редакторы: для каждого вида документов - свой редактор. В частности, на ранних этапах разработки в документах могут широко использоваться графические средства описания (диаграммы, схемы и т.п.). В таких случаях весьма полезными могут быть графические редакторы. На этапе программирования (кодирования) вместо текстового редактора может оказаться более удобным синтаксически управляемый редактор, ориентированный на используемый язык программирования.

Анализаторы производят либо статическую обработку документов, осуществляя различные виды их контроля, выявление определенных их свойств и накопление статистических данных (например, проверку соответствия документов указанным стандартам), либо динамический анализ программ (например, с целью выявление распределения времени работы программы по программным модулям).

Преобразователи позволяют автоматически приводить документы к другой форме представления (например, форматеры) или переводить документ одного вида к документу другого вида (например, конверторы или компиляторы), синтезировать какой-либо документ из отдельных частей и т.п.

Инструменты, поддерживающие процесс выполнения программ, позволяют выполнять на компьютере описания процессов или отдельных их частей, представленных в виде, отличном от машинного кода, или машинный код с дополнительными возможностями его интерпретации. Примером такого инструмента является эмулятор кода другого компьютера. К этой группе инструментов следует отнести и различные отладчики. По существу, каждая система программирования содержит программную подсистему периода выполнения, которая выполняет программные фрагменты, наиболее типичные для языка программирования, и обеспечивает стандартную реакцию на возникающие при выполнении программ исключительные ситуации (такую подсистему мы будем называть исполнительной поддержкой). Такую подсистему также можно рассматривать как инструмент данной группы.

Совокупность инструментальных сред можно разбивать на разные классы, которые различаются значением следующих признаков:

* ориентированность на конкретный язык программирования,
* специализированность,
* комплексность,
* ориентированность на конкретную технологию программирования,
* ориентированность на коллективную разработку,

Ориентированность на конкретный язык программирования (языковая ориентированность) показывает: ориентирована ли среда на какой-либо конкретный язык программирования (и на какой именно) или может поддерживать программирование на разных языках программирования. В первом случае информационная среда и инструменты существенно используют знание о фиксированном языке (глобальная ориентированность), в силу чего они оказываются более удобным для использования или предоставляют дополнительные возможности при разработке ПС. Но в этом случае такая среда оказывается не пригодной для разработки программ на другом языке. Во втором случае инструментальная среда поддерживает лишь самые общие операции и, тем самым, обеспечивает не очень сильную поддержку разработки программ, но обладает свойством расширения (открытости). Последнее означает, что в эту среду могут быть добавлены отдельные инструменты, ориентированные на тот или иной конкретный язык программирования, но эта ориентированность будет лишь локальной (в рамках лишь отдельного инструмента).

Специализированность инструментальной среды показывает: ориентирована ли среда на какую-либо предметную область или нет. В первом случае информационная среда и инструменты существенно используют знание о фиксированной предметной области, в силу чего они оказываются более удобными для использования или предоставляют дополнительные возможности при разработке ПС для этой предметной области. Но в этом случае такая инструментальная среда оказывается не пригодной или мало пригодной для разработки ПС для других предметных областей. Во втором случае среда поддерживает лишь самые общие операции для разных предметных областей. Но в этом случае такая среда будет менее удобной для конкретной предметной области, чем специализированная на эту предметную область.

Комплексность инструментальной среды показывает: поддерживает ли она все процессы разработки и сопровождения ПС или нет. В первом случае продукция этих процессов должна быть согласована. Поддержка инструментальной средой фазы сопровождения ПС, означает, что она должна поддерживать работу сразу с несколькими вариантами ПС, ориентированными на разные условия применения ПС и на разную связанную с ним аппаратуру, т.е. должна обеспечивать управление конфигурацией ПС [16.1, 16.3].

Ориентированность на конкретную технологию программирования показывает: ориентирована ли инструментальная среда на фиксированную технологию программирования [16.2] либо нет. В первом случае структура и содержание информационной среды, а также набор инструментов существенно зависит от выбранной технологии (технологическая определенность). Во втором случае инструментальная среда поддерживает самые общие операции разработки ПС, не зависящие от выбранной технологии программирования.

Ориентированность на коллективную разработку показывает: поддерживает ли среда управление (management) работой коллектива или нет. В первом случае она обеспечивает для разных членов этого коллектива разные права доступа к различным фрагментам продукции технологических процессов и поддерживает работу менеджеров [16.1] по управлению коллективом разработчиков. Во втором случае она ориентирована на поддержку работы лишь отдельных пользователей.

1. Анализ предметной области «Магазин»

Предметная область информационной системы - это материальная система или система, характеризующая элементы материального мира, информация о которой хранится и обрабатывается.

С точки зрения проектирования программы в рамках системного анализа, необходимо осуществить первый этап, то есть провести подробное словесное описание объектов предметной области и реальных связей, которые присутствуют между описываемыми объектами. Желательно, чтобы данное описание позволяло корректно определить все взаимосвязи между объектами предметной области.

В общем случае существуют два подхода к выбору состава и структуры предметной области:

Функциональный подход — он реализует принцип движения "от задач" и применяется тогда, когда заранее известны функции некоторой группы лиц и комплексов задач, для обслуживания информационных потребностей.

Предметный подход — когда информационные потребности будущих пользователей программы жестко не фиксируются. Они могут быть многоаспектными и весьма динамичными. Мы не можем точно выделить минимальный набор объектов предметной области, которые необходимо описывать. В описание предметной области в этом случае включаются такие объекты и взаимосвязи, которые наиболее характерны и наиболее существенны для нее.

Чаще всего на практике рекомендуется использовать некоторый компромиссный вариант, который, с одной стороны, ориентирован на конкретные задачи или функциональные потребности пользователей, а с другой стороны, учитывает возможность наращивания новых приложений.

Анализ должен заканчиваться подробным описанием информации об объектах предметной области, которая требуется для решения конкретных задач с кратким описанием алгоритмов их решения, описанием выходных документов, которые должны генерироваться в системе.

Управление магазином осуществляется директором. В его подчинении находятся: заведующий магазина и бухлагтер.

Заведующий магазина – это сотрудник, участвующий в формировании товарного ассортимента, инвентаризациях, и проведении переоценок товаров; определяет ценовую категорию для всех товаров; осуществляет контроль качества реализуемой продукции, правильности ценников, наличия сопроводительной документации; обеспечивает своевременное выполнения планов реализации; введет документацию; осуществляет подбор и обучение персонала.

– бухгалтер осуществляет бухгалтерский учёт, работает по системе учёта в соответствии с действующим законодательством;

– грузчик выполняет физическую работу погрузку, выгрузку товара.

– охранник — человек, который отвечает за безопасность людей или сохранность материальных ценностей.

– Продавец — профессия, специальность, человек или организация, которая продает что-либо (товар, услугу и так далее). Продавец за соответствующее вознаграждение передает покупателю товар или услугу.

На рисунке 1.1 предоставлена организационная структура магазина

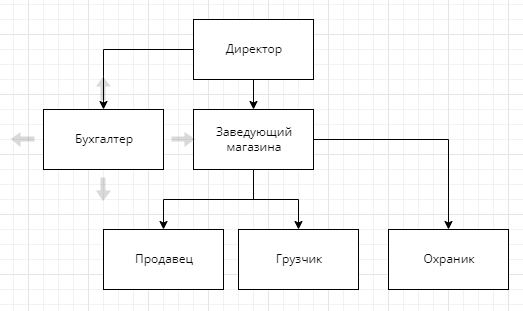


Рисунок 1.1 – Организационная структура магазина

Магазин, устар. лавка — предприятие розничной торговли, размещённое в стационарном здании, оборудованном для продаж товаров и оказания услуг. Здание или часть здания, где располагается современный магазин, обычно включает как собственно торговый зал, так и подсобные, административно-бытовые помещения и склады.

Виды деятельности магазина:

* розничная куплю продажи товаров;
* осуществлению всех видов внешнеэкономической деятельности;
* оказание помощи покупателю в совершении покупки и при её использовании;

Так как основным видом деятельности магазина является розничная куплю продажи товаров рассмотрим этот процесс подробнее.

Началом работы является пришедший клиент и выбранным им товар в магазине, на основе этих данных, продавец подсчитывает стоимость выбранных товаров, высказывает эту стоимость клиенту и просить личные данных покупателя, после чего клиент соглашается на стоимость товаров и предоставляет требуемые продавцу данные и оплачивает товары, далее продавец выписывает выбранные товары клиентом в тетрадь с указанным количеством ценой и стоимостью.

Проблема данного процесса в сложной отчётности, поиска информации о клиентах и наличием факто человеческого труда, который может сделать ошибку в подсчётах данный процесс можно оптимизировать с помощью программного обеспечения.

Рассмотрим данный процесс с использованием программного обеспечения.

Началом работы является пришедший клиент и выбранным им товар в магазине, на основе этих данных, продавец вводить в программу указанный товар после чего программа выводит ему итоговую стоимость, после чего клиент соглашается на стоимость товаров и предоставляет требуемые продавцу данные и оплачивает товары, далее продавец подтверждает в программе что была клиент оплатил за товар.

Видим, что в данном процессе при использовании программного обеспечения уменьшается ответственность работы продавца и автоматизируется процесс подсчета стоимости товара и формирование данных о проданных товарах.

После изучения процесса требуется определить сущности данных которые необходимо обрабатывать в программе:

Сущности товар состоит из:

* наименования;
* стоимости.

Сущности клиент состоит из:

* фамилии имя отчество;
* возраст;
* номер телефона.

Сущности заказ состоит из:

* дата;
* информации о клиенте;
* набор купленного товара.

Данная программа будет предоставлять интерфейс через консольную команду и разработана на языке программирование С++.

Выбран C++ за счет своих следующих качеств:

* Компилируемый язык со статической типизацией.
* Сочетание высокоуровневых и низкоуровневых средств.
* Реализация ООП.

Рассмотрим данные причины более подробно.

Компилятор. Тут C++ предстаёт во всей красе. Множество компиляторов, консольные команды, этапы сборки программы… Да, первую программу нужно написать в простом текстовом редакторе без подсветки синтаксиса и автокомплита, найти чем и как её можно запустить. Такой подход формирует у человека некоторое понимание того, как всё устроено.

Статическая типизация. На примере языка со статической типизацией проще понять, что такое тип данных, зачем он нужен и от чего зависит. Видно, что собой представляет объявление, определение и инициализация. Использование языка C++ даёт это явно увидеть, что способствует дальнейшему пониманию того, как работают эти механизмы в других языках. Помимо этого, можно на реальных примерах понять, чем беззнаковые целые отличаются от целых со знаком, чем отличаются числа двойной и одинарной точности, чем отличается символ от строки и т.д.

Высокоуровневые и низкоуровневые средства. Использование таких средств, как указатели и динамическое выделение памяти, позволяет понять (или в дальнейшем способствует пониманию), что такое стэк, куча, стэк вызовов, раскрутка стэка и т.д. Помимо этого, на практике закрепляется понимание концепции адресов и адресной арифметики. На примерах демонстрируется, что память надо выделять, освобождать, потому что она не бесконечная, что существуют утечки памяти. В будущем, при изучении языков с GC проще будет понять, что же это такое.

Реализация ООП. Это относительно чистая реализация ООП без всякого синтаксического сахара (относительно некоторых других языков). Чётко разграниченные уровни доступа к членам класса, возможность множественного наследования и динамический полиморфизм дают возможность быстро усвоить основные концепции ООП (абстракция, наследование, инкапсуляция и полиморфизм). Указатели и динамическое выделение памяти позволяют наглядно понять такие важные механизмы, как upcasting и downcasting. В дальнейшем, основываясь на этих знаниях, легко можно понять весь синтаксический сахар в других языках. Необходимость контроля ресурсов (в том числе и «правило трёх» или уже «правило пяти», с учётом C++), захват их в конструкторе и освобождение в деструкторе также способствуют более глубокому пониманию ООП.

1. Разработка программного обеспечения

Этапы разработки программного обеспечения

В зависимости от вида, масштабов и потребностей проекта определяется порядок разработки. Он будет несколько отличаться для разработки мобильных приложений, встроенного ПО, решений для автоматизации и БД, но общая последовательность действий для создания ПО универсальна:

Дизайн — вторая по важности составляющая продукта после технических характеристик, влияющая на эффективность и скорость взаимодействия пользователя с ним. Требования к дизайну определяются ТЗ — как правило, важны простота, интуитивность и минимальные затраты на совершения действия (достижение результата), а также красота и соответствие стилю компании и (или) продукта.

Код — та часть работы, которая обычно ассоциируется с разработкой ПО как таковой. Важно, чтобы код был в достаточной мере оптимизированным, лаконичным и понятным. Назначаем на подобранные под специфику задания в ТЗ языки специализирующихся на их использовании программистов.

Тестирование. Тестирование в EDISON проводится на каждом этапе разработки ПО, включает множество тестов по плану тестирования, кастомизируемому с учётом специфики проекта на этапе составления технического задания. Результаты тестирования документируются и доступны клиенту в режиме реального времени. Оплата за продукт производится только после прохождения всех видов тестов, в том числе клиентских.

Документирование — процедура, фиксирующая план, процесс и результат разработки программного обеспечения. Включает в себя всю исходную информацию (ТЗ, макеты), планы работ, затрат, тестирования, список задач исполнителей в каждый момент времени, отчеты о работе и так далее. Документация необходима для быстрого и точного выявления ошибок, прозрачности совместной работы, как обязательная юридическая часть договора.

Принципы разработки программного обеспечения:

Ориентация на качество. Мы прилагаем все усилия, чтобы это было не избитым маркетинговым клише, а объективной реальностью. Бесперебойность работы и удовлетворенность конечным результатом обеспечивают:

следование ГОСТам, лучшим практикам и методологиям качественной разработки (RUP, Agile),

лучшие спецы, четкое разделение труда и хорошая мотивация срок+качество,

отлаженная и мощная система тестирования продуктов,

качественное и прозрачное планирование и выполнение задач, система управления разработкой и обязательность грамотного технического задания,

документирование процесса и результата,

гарантии на разработанные продукты, техническая поддержка и обучение пользователей,

понятная и удобная система оплаты за разработку ПО.

Адаптивность и гибкость. В некоторых проектах нет возможности четкой формулировки требований на этапе составления ТЗ, а иногда у клиента уже на этапе разработки программного обеспечения появляется потребность в изменениях, — мы с пониманием относимся к таким ситуациям и заранее предусматриваем их вероятность и согласовываем с клиентом условия работы при прецеденте.

Основные этапы разработки ПО:

Этап 1 – Определение проблемы

Перед тем как приступать к кодированию, необходимо четко сформулировать проблему, которую Ваша будущая программа должна решать. Так как, не имея хорошего определения проблемы, Вы можете потратить много усилий и времени на решение не той проблемы, которую требуется решить.

На данном этапе проводится простая формулировка сути проблемы без каких-либо намеков на ее возможные решения, при этом формулировать ее следует на языке, понятном пользователю, т.е. она должна быть описана с пользовательской точки зрения.

Этап 2 – Выработка требований

Требования к программе – это подробное описание всех возможностей программы и действий, которые должна выполнять программа. Такие требования иногда также называют «Функциональной спецификацией» или просто «Спецификацией».

Требования вырабатывают для того, чтобы свести к минимуму изменения системы после начала непосредственной разработки. Такие требования должны быть обязательно официальными, т.е. документально оформлены. Так как это гарантирует то, что функциональность системы определяется заказчиком, а не программистом. Даже в случае с внутрикорпоративными разработками такие требования должны быть зафиксированы, например, в виде технического задания, подписанного всеми задействованными лицами, тем самым Вы избежите лишних разговоров и споров, например, о том, что реализованный функционал делает не все или не так.

Выработка требований очень важна, так как она позволяет определить функциональность программы до начала программирования.

Этап 3 – Создание плана разработки

На данном этапе нужно составить формальный план разработки программного обеспечения с учётом существующей проблемы и выработанных требований. Иными словами, Вы должны составить план того, как Вы будете действовать дальше.

Этап 4 – Разработка архитектуры системы или высокоуровневое проектирование

Данный этап также очень важный, так как, не имея хорошей архитектуры, Вы можете решать правильную проблему, но прийти к неправильному решению. Хорошая архитектура программы упрощает программирование, а плохая архитектура усложняет его.

Этап 5 – Детальное проектирование

На этом этапе проводится проектирование программы на низком уровне, иными словами, здесь проектируются классы и методы, рассматриваются, оцениваются и сравниваются различные варианты и причины выбора окончательных подходов и способов реализации.

Этап 6 – Кодирование и отладка

Это как раз тот этап, который все знают и, наверное, думают, что это единственный этап в процессе разработке программного обеспечения – это непосредственное написание кода и его отладка. Но, как видите, это далеко не первый и не единственный этап разработки ПО.

Этап 7 – Тестирование компонентов

После того, как код написан, и проведена отладка, необходимо провести тестирование реализованного функционала. Если программа состоит из нескольких компонентов, сначала тестируют каждый компонент в отдельности, так как очень крупные программы включают огромный функционал, который часто разделяют на отдельные компоненты, разработка которых осуществляется по отдельности. В менее крупных проектах этот этап может включать просто тестирование отдельных классов.

Этап 8 – Интеграция компонентов

Когда тестирование всех компонентов закончено, можно переходить к интеграции всех компонентов в единый программный комплекс, этот этап как раз и подразумевает процесс интеграции, т.е. слияния всех компонентов в единую систему.

В небольших проектах этот этап может заключаться в объединении нескольких классов, на что будет затрачено не больше одного дня, но в крупных проектах этот этап может длиться не один месяц.

Этап 9 – Тестирование всей системы

На данном этапе проводится тестирование всей системы, уже с учётом интеграции всех компонентов. На этом этапе можно выявить проблемы взаимодействия компонентов и устранить их. Также на этом этапе основным предметом тестирования является безопасность, производительность, утечка ресурсов и другие моменты, которые невозможно протестировать на более низких уровнях тестирования.

Этап 10 – Сопровождение, внесение изменений, оптимизация

После запуска программы в промышленную эксплуатацию осуществляется сопровождение этой программы, т.е. внесение изменений на основе выявленных недочетов в процессе эксплуатации системы, а также проводится оптимизация функционала или добавление нового.

Заключение

В ходе выполнения курсовой работы были выполнены следующие задачи:

* ­приобретены навыки самостоятельной работы с литературой по выбранной тематике;
* систематизация и закрепления теоретические знания по дисциплине;
* развиты навыки решения практических задач;
* произведены необходимые виды анализа для разработки программ;
* произведен анализ предметной области;
* рассмотрена история и этапы развития программирования;
* определены средства разработки программного обеспечения;

Была выполнена цель курсовой работы закреплены знания и навыки, в изучении дисциплины. Получены практические навыки по реализации программных средств на языке программирования Си.

Список использованных источников

1. Баранова, Е.В. Информационные технологии в образовании. [Электронный ресурс] / Е.В. Баранова, М.И. Бочаров, С.С. Куликова, Т.Н. Носкова. — Электрон. дан. — СПб. : Лань, 2016. — 296 с;
2. Грошев, А.С. Информатика. [Электронный ресурс] / А.С. Грошев, П.В. Закляков. — Электрон. дан. — М. : ДМК Пресс, 2015. — 588 с.;
3. Конова, Е.А. Алгоритмы и программы. Язык С++. [Электронный ресурс] / Е.А. Конова, Г.А. Поллак. — Электрон. дан. — СПб. : Лань, 2017.;
4. Кудинов, Ю.И. Основы современной информатики. [Электронный ресурс] / Ю.И. Кудинов, Ф.Ф. Пащенко. — Электрон. дан. — СПб. : Лань, 2017. — 256 с.;
5. Смоленцев, Н.К. MATLAB. Программирование на С++, С#, Java и VBA.