



А.П. ВОРОБЬЕВА

Л.А. ЛИТВИНОВ

ОСНОВЫ ИНФОРМАТИКИ

НОВОСИБИРСК 2019

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ
АРХИТЕКТУРНО-СТРОИТЕЛЬНЫЙ УНИВЕРСИТЕТ (СИБСТРИН)

А.П. Воробьева, Л.А. Литвинов

ОСНОВЫ ИНФОРМАТИКИ

Учебное пособие

Под редакцией Л.А. Литвинова

НОВОСИБИРСК 2019

УДК 004
ББК 32.81
В 751

Воробьева, А. П.

Основы информатики : учеб. пособие / А. П. Воробьева, Л. А. Литвинов ; под ред. Л. А. Литвинова ; Новосиб. гос. архитектур.-строит. ун-т (Сибстрин). – Новосибирск : НГАСУ (Сибстрин), 2019. – 136 с.

ISBN 978-5-7795-0879-7

В учебном пособии рассмотрены важные разделы курса "Информатика", на которых основано применение методов прикладной математики в производственно-технической сфере. Показана эволюция объектно-ориентированного подхода к проектированию. На примерах подробно проиллюстрировано использование современных средств разработки программного обеспечения.

Учебное пособие предназначено для студентов направления 08.03.01 "Строительство" всех форм обучения; может быть полезно магистрантам, аспирантам и инженерам, создающим программные продукты для численного моделирования.

Рецензенты:

- И.А. Бедарев, канд. физ.-мат. наук, доцент,
зав. лабораторией ИТПМ СО РАН;
- Н.П. Кисленко, канд. техн. наук, доцент
НГАСУ (Сибстрин)

ISBN 978-5-7795-0879-7

- © Воробьева А.П., Литвинов Л.А., 2019
- © Новосибирский государственный
архитектурно-строительный
университет (Сибстрин), 2019

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ	5
1. ЭЛЕМЕНТЫ ЛОГИКИ	7
1.1. Логика мышления и наука "Логика"	7
1.2. Основные определения	7
1.3. Основные законы логического мышления.....	14
1.4. Дедукция и индукция	16
2. ИЗ ИСТОРИИ РАЗВИТИЯ ЭВМ.....	20
2.1. Первые ЭВМ.....	20
2.2. Принципы и схема фон Неймана	21
2.3. Компьютеры в СССР.....	23
3. СТРУКТУРНОЕ ПРОГРАММИРОВАНИЕ	28
3.1. Алгоритм и его свойства.....	29
3.2. Алгоритмические структуры.....	31
3.3. Алгоритмы сортировки.....	33
3.4. Технология структурного программирования.....	36
4. СОВРЕМЕННЫЕ ТЕХНОЛОГИИ ПРОГРАММИРОВАНИЯ	41
4.1. Объектно-ориентированное программирование	41
4.2. Визуальное программирование и Visual Basic	44
4.3. Создание приложения в Visual Studio	45
5. ПРИЁМЫ РАБОТЫ В VISUAL BASIC FOR APPLICATIONS.....	52
5.1. Основные команды панели Разработчик в пакете Microsoft Office	52
5.2. Применение макросов для работы с книгой Excel	55
5.3. Основные структуры языка программирования Visual Basic	83

5.4. Базовые приёмы обработки структурированных данных.....	98
5.5. Программирование форм для работы с книгой Excel	105
6. РЕШЕНИЕ ЗАДАЧ НА КОМПЬЮТЕРЕ.....	116
6.1. Решение нелинейных алгебраических уравнений.....	116
6.2. Решение систем линейных алгебраических уравнений.....	119
6.3. Интерполяция	122
6.4. Аппроксимация функций.....	129
ЗАКЛЮЧЕНИЕ	135
БИБЛИОГРАФИЧЕСКИЙ СПИСОК.....	136

Легко в учении – тяжело в походе,

тяжело в учении – легко в походе.

Александр Васильевич Суворов

ВВЕДЕНИЕ

Информатика – это наука. Вот уже не один десяток лет эту науку преподают в средней школе и в техникумах, вузах и колледжах.

Традиционно в информатику входят следующие разделы:

- двоичная арифметика и логика;
- алгоритмы решения задач и их свойства;
- технологии программирования;
- устройства вычислительной техники;
- программное обеспечение;
- компьютерные сети и Интернет.

Естественно, подробное изучение всех разделов информатики – дело интересное, но подходящее скорее для студентов информационно-технологических направлений.

В этом небольшом по объёму учебном пособии мы постарались осветить те важные моменты, которые помогут студенту – будущему специалисту свободно ориентироваться в актуальной информационной среде.

В современном мире человек становится пользователем компьютера, даже не осознавая этого. Инженер – это разработчик нового. Таким образом, изучение информатики способствует превращению будущих инженеров в более квалифицированных пользователей, а затем и в разработчиков.

Человек основал работу вычислительных устройств на использовании двоичной логики и двоичной арифметики, которые берут своё начало соответственно в логике и математике. Для усиления фундаментальной составляющей части знаний, получаемых в процессе освоения курса информатики, первая глава настоящего пособия представляет собой краткий конспект знаменитого учебника С.Н. Виноградова и А.Ф. Кузьмина "Логика" [1].

Логическое мышление – неперенное условие успешной деятельности в науке и технике.

Во второй главе дан обзор основных вех истории развития цифровой вычислительной техники, который может мотивировать нынешнее поколение инженерно-технических сотрудников на разработку новых проектов.

В третьей главе рассмотрены основные составляющие структурного программирования – алгоритм и его свойства, алгоритмические структуры, подпрограмма, модуль и другие. Авторы разделяют точку зрения о необходимости в рамках освоения курса информатики обязательно изучать программирование. И абсолютно уверены, что одним из основных результатов изучения информатики для студентов является развитие у них алгоритмического мышления – неотъемлемого качества руководящего работника в инженерно-технической сфере деятельности.

Четвёртая глава посвящена актуальным технологиям программирования. Современное автоматизированное проектирование основано на объектно-ориентированной парадигме, когда описание объекта включает не только его характеристики, но и функционал. Визуальное программирование позволяет легко создавать полноценные Windows-приложения с помощью интегрированных сред разработки программных продуктов.

В пятой главе показывается, как элементы программирования на языке Visual Basic for Applications, который встроен в каждое приложение пакета Microsoft Office, придают большую гибкость широко распространенной технологии хранения и обработки данных с помощью электронных таблиц Excel. Рассмотрены основные конструкции языка программирования Visual Basic. Приведены тексты подпрограмм, иллюстрирующие некоторые приёмы программирования.

В шестой главе уделено внимание решению некоторых математических задач на компьютере.

1. ЭЛЕМЕНТЫ ЛОГИКИ

1.1. Логика мышления и наука "Логика"

Термин "логика" происходит от древнегреческого слова "логос". Он означает "мышление", "мысль" или "слово". Ведь человек выражает свои мысли словами. Логическое мышление, без которого невозможна научно-техническая деятельность, состоит в умении правильно построить мысль. Существует простой способ для выработки и совершенствования логического мышления – изучение науки "Логика", науки о законах и формах правильного построения мыслей.

Ниже будут приведены основные определения логики так, как они сформулированы в вышеупомянутом учебнике [1].

1.2. Основные определения

Логические законы:

- 1) определённости;
- 2) непротиворечивости;
- 3) последовательности;
- 4) обоснованности.

Соблюдение этих законов является обязательным качеством правильного мышления. Сознательное или несознательное нарушение логических законов ведёт к неправильному выводу. Человек, который нарушает логические законы, неизбежно оказывается побеждённым в споре, дискуссии.

Логическая форма – это структура, строение наших мыслей. Если в процессе рассуждения наши мысли облачаются в неправильные формы, то в таком случае прийти к истинным выводам невозможно. Таким образом, логика изучает формы мышления. Но это не значит, что логика не интересуется содержанием мышления. Изучение формы мысли вне связи с содержанием не имело бы для нас никакого смысла. Однако изучение её в связи с содержанием не означает, что мы не можем в необходимых случаях в целях изучения мысленно отвлекать эту форму.

Человеческое мышление развивается, изменяется, т.е. становится более совершенным. Но изменение форм мышления в течение длительного времени мало заметно. Логические формы и законы обладают устойчивостью, постоянством.

Ощущение – это отображение нашим сознанием того или иного свойства материального предмета, например твёрдости, цвета и т.д. Процесс познания начинается с ощущений, возникающих в результате непосредственного воздействия предметов и явлений материального мира на органы чувств.

Восприятие – это отображение в нашем сознании отдельных предметов и явлений как единого целого.

Мышление – деятельность мозга человека, направленная на то, чтобы познать законы, по которым совершается развитие и изменение вещей и явлений, и во время которой человек сопоставляет, сравнивает, перерабатывает в своём мозгу ощущения и восприятия этих вещей и явлений – отвлекает, выделяет важное и существенное, отображает связи и отношения вещей и явлений.

Мышление – *опосредованное и обобщённое* познание действительности.

Для того чтобы познать общие свойства, отношения, закономерности предметов и явлений объективной действительности, человек применяет различные **логические приёмы**. Такими основными приёмами мышления являются сравнение, анализ и синтез, абстрагирование и обобщение.

Сравнение – это такой логический приём, с помощью которого устанавливается сходство и различие предметов, явлений объективного мира. При этом:

- 1) нужно сравнивать такие предметы, которые в действительности имеют какие-то связи друг с другом;
- 2) правильность любого сравнения определяется тем, что мы возьмём за основу сравнения (показатель);
- 3) сравнение двух или нескольких предметов надо производить по одному и тому же признаку, взятому в одном и том же отношении;

- 4) всякое сравнение должно проводиться не по первым попавшимся признакам, а по таким признакам, которые имеют важное, существенное значение для сравниваемых предметов.

Анализ – это такой логический приём, с помощью которого мы мысленно расчлняем предметы, явления, выделяя отдельные их части, свойства. Никакой более или менее сложный предмет невозможно изучить, не подвергнув его анализу.

Синтез – это такой логический приём, с помощью которого мы мысленно соединяем в одно целое расчленённые в анализе отдельные части предмета, явления.

Анализ и синтез – это два неразрывно связанных друг с другом логических приёма. Синтез невозможен, если предмет не был проанализирован, а всякий анализ должен производиться на основе знания предмета как целого. Мышление состоит столько же в разложении предметов сознания на их элементы, сколько в объединении элементов в единство.

Абстрагирование – это такой логический приём, с помощью которого мы мысленно выделяем существенные свойства предметов, явлений и отвлекаем их от несущественных, второстепенных свойств предметов, явлений материального мира.

Результат абстрагирования называется **абстракцией**.

Абстракция может быть правильной, а может быть и неправильной. Правильная абстракция отображает содержание, заключённое в вещах. Так, абстрактное понятие "геометрическая фигура" отображает конкретное свойство предметов материального мира, их формы.

Неправильной абстракция бывает в тех случаях, когда мыслятся свойства, которые к изучаемому предмету никакого отношения не имеют. Грубая ошибка совершается и тогда, когда отвлечённый от предмета признак начинают рассматривать как что-то возникшее и существующее самостоятельно, забывая о связи абстрактного понятия с предметом.

В процессе абстракции мы выделяем свойства предметов, явлений. Но познать предмет вне связи с окружающей средой нельзя. Каждый единичный предмет входит в состав какого-то

класса предметов, органически связан с чем-то более общим. Наша мысль и отображает связи изучаемого предмета с тем общим, к которому принадлежит предмет.

Обобщение – это мысленное объединение общих свойств однородных предметов.

В процессе обобщения человек как бы отходит от конкретных предметов, отклоняется от массы деталей, присущих единичным вещам. Но это необходимо для того, чтобы, познав общее, глубже проникнуть в сущность единичных предметов. Абстрагируя свойства предметов, мы тем самым уже отображаем общие свойства предметов. Абстрагирование и обобщение представляют собой единый, неразрывный процесс.

Но обобщения могут быть как правильные, так и неправильные. Обобщение правильно только в том случае, если основано на познании общего, находящегося в самих вещах. Отступление от этого условия ведёт к логическим ошибкам. Так, если глубокое изучение свойств предмета или явления подменяется поверхностным ознакомлением с ним, то это может привести к неправильному, поспешному обобщению.

Предметами мысли в логике принято называть вещи и явления окружающей нас действительности.

Признаки – это свойства вещей и явлений в логике.

Понятие – это мысль, которая отображает общие и существенные признаки предметов.

Существенным признаком предмета называется тот признак, который выражает коренное, наиболее важное свойство предмета; если существенный признак отсутствует, то предмет перестаёт быть данным предметом.

Понятия только в том случае являются правильными, если они верно отражают действительность. Если же какое-либо понятие представляет собой неверное, искажённое отображение действительности, то такое понятие является ложным. Ложные понятия возникают, например, при отрыве теории от практики.

Правильные понятия вырабатываются в процессе трудовой деятельности многих людей; соответствие таких понятий пред-

метам и явлениям действительности проверяется практикой человека.

Представления – это наглядные образы предметов, явлений.

Представления всегда имеют индивидуальный характер. В них главное не отделяется от второстепенного, они могут складываться и из несущественных признаков. Понятия, в отличие от представлений, отражают сущность вещей. Они имеют характер всеобщности: одними и теми же понятиями пользуется множество разных людей.

Понятия, являясь отражением объективного мира, возникают в результате мыслительной деятельности многих людей. Они отличаются устойчивостью и, как всякий накопленный людьми опыт, передаются (с помощью языка) от человека к человеку.

Мы постоянно пользуемся понятиями как основным фондом всех наших знаний, в котором запечатлена многовековая практика человека.

Каждое понятие имеет *содержание* и *объём*.

Содержание понятия – это знание о совокупности существенных признаков класса предметов.

Объём понятия – это знание о круге предметов, существенные признаки которых отображены в понятии.

Закон обратного отношения между содержанием и объёмом понятия имеет следующую формулировку: чем шире содержание понятия, тем уже его объём. И соответственно наоборот: чем уже содержание понятия, тем шире его объём.

Закон обратного отношения распространяется только на такие понятия, из которых одно входит в объём другого.

Обобщение понятия – переход от менее общего к более общему понятию.

Ограничение понятия – переход от более общего понятия к менее общему понятию.

Ограничением понятий мы пользуемся в тех случаях, когда разъясняем содержание какого-либо понятия, причём строим своё разъяснение на основе уже известных, более общих понятий. Ограничение понятия применяется также в тех случаях, когда бывает необходимо уточнить содержание понятия, указать,

к какому именно кругу явлений относится данное понятие, следовательно, отграничить понятие от других понятий, в том числе и от более общих.

Обобщением понятий широко пользуется наука, которая всегда стремится вскрыть в предметах наиболее общие их свойства.

Обобщая понятия, переходя от менее общих к более общим, мы приходим, наконец, к предельно широким по объему понятиям, которые не подлежат дальнейшему обобщению. Такие понятия называются **категориями**. Примеры категорий: "материя", "время", "движение", "пространство", "количество", "форма" и др.

Более общие понятия называются **родовыми понятиями**, менее общие – **видовыми понятиями**.

Единичные понятия – это понятия об отдельных (единичных) предметах. В **общих понятиях** отображается множество однородных предметов.

Собирательные понятия – это такие понятия, в которых мыслится совокупность однородных предметов как единое целое.

Несравнимые понятия – это такие понятия, которые по своему содержанию находятся в далёком отношении друг к другу (рис. 1.1).

Сравнимые понятия – это понятия, имеющие общие родовые признаки. Например, для понятий "роза" и "незабудка" можно найти общий родовой признак: "быть цветком". Такие понятия делятся на две группы:

- 1) совместимые понятия, если их объёмы совпадают полностью или частично;
- 2) несовместимые понятия – с несовпадающими объёмами.

Понятия находятся в **отношении тождества**, если они могут различаться по своему содержанию, но в них мыслится один и тот же предмет.

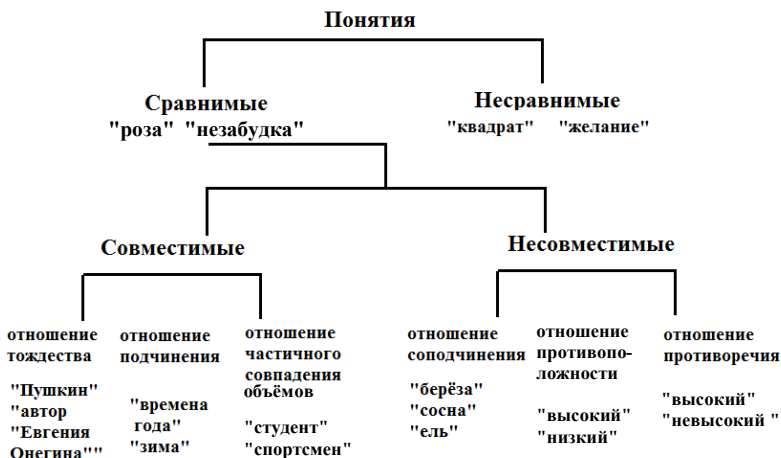


Рис. 1.1. Отношения между понятиями

При **отношении подчинения** одно понятие (менее общее) входит в объём другого понятия (более общего). Отношение подчинения есть отношение вида и рода. Отношение подчинения понятий не следует смешивать с отношением части и целого.

Такие понятия, объёмы которых частично совпадают, называются **перекрещивающимися понятиями**.

В **отношении противоположности** находятся такие два понятия, которые по своему содержанию противоположны друг другу, но оба входят в объём одного и того же родового понятия. Каждое из противоположных понятий не только отрицает своим содержанием другое, противоположное понятие, но и утверждает взамен другого, противоположного, нечто новое, несовместимое с ним.

В **отношении противоречия** находятся такие два понятия, из которых одно полностью отрицает другое, но содержание отрицающего понятия остаётся неопределённым.

Определение понятия есть такое логическое действие, в процессе которого раскрывается содержание понятия. Раскрыть содержание понятия – это значит указать его существенные признаки.

Определением понятия называется также результат указанного действия.

По своему строению определение состоит из двух основных частей: определяемого понятия и определяющего понятия. Определяющее понятие указывает на ближайший род определяемого и на его видовое отличие. Состав определения схематически можно изобразить таким образом: "вид" есть "род и видовое отличие".

Существует четыре правила определения:

1. Определение должно быть соразмерным.
2. Определение не должно делать круга.
3. Определение не должно быть отрицательным.
4. Определение должно быть ясным, чётким, не допускающим двусмысленных или метафорических выражений.

Существуют *логические приёмы*, которые могут дополнить определение. Среди таких приёмов отметим следующие пять: **указание, описание, характеристика, сравнение, различение.**

Деление понятия есть такое логическое действие, в процессе которого раскрывается объём понятия. Раскрыть объём понятия – это значит указать видовые понятия, соподчинённые делимому понятию.

Всего правил деления четыре:

1. Деление должно быть соразмерным.
2. Деление должно производиться по одному основанию, и притом существенному.
3. Члены деления должны исключать друг друга.
4. Деление не должно делать скачка.

Классификацией называется система расположения предметов по классам на основании сходства этих предметов внутри класса и их отличия от предметов других классов.

Суждением называется мысль, которая утверждает или отрицает что-либо относительно предметов и их признаков.

1.3. Основные законы логического мышления

Логика рассматривает четыре закона логического мышления: закон тождества, закон противоречия, закон исключённого

третьего и закон достаточного основания. Эти законы выражают коренные черты мышления: определённая, непротиворечивость, последовательность и обоснованность.

Чтобы мышление было правильным, оно должно быть вполне точным и определённым, строго последовательным, не должно отклоняться от предмета рассуждения, не должно содержать в себе логических противоречий и двусмысленностей. Наши суждения и выводы должны быть обоснованными, без чего они не могут стать убедительными. Логические законы являются обязательным условием правильного мышления.

Закон тождества формулируется так: **в данном рассуждении, споре, дискуссии каждое понятие должно употребляться в одном и том же смысле.**

Нарушением закона тождества является логическая ошибка, которая называется подменой понятия.

Закон противоречия формулируется следующим образом: **два противоположных высказывания не могут быть оба истинными в одно и то же время, в одном и том же отношении.**

Закон противоречия указывает лишь на несовместимость двух противоположных суждений, но не говорит о том, является ли ложным одно из них или оба они ложны.

Закон исключённого третьего имеет следующую формулировку: **из двух противоречащих суждений всегда одно истинное, другое ложное, а третьего быть не может.**

Закон достаточного основания формулируется таким образом: **всякая истинная мысль должна быть обоснованной.**

Как в природе всё имеет своё реальное основание, так и наши мысли, отражающие действительность, должны быть обоснованными.

Закон достаточного основания направлен против нелогичного мышления, принимающего на веру ничем не обоснованные суждения. Суждения, которые приводятся для обоснования правильности других суждений, называются логическим основанием.

1.4. Дедукция и индукция

Умозаключение – это такое логическое действие, посредством которого из двух или нескольких суждений мы получаем новое суждение.

Умозаключения:

- бывают **дедуктивные**, когда мысль идёт **от общего к частному**;
- **индуктивные**, когда мысль идёт **от частного к общему**;
- могут быть такими, когда мысль идёт **от частного к частному**.

Силлогизм, или **дедуктивное умозаключение**, – это такое умозаключение, в котором из двух данных суждений **выводится третье суждение, причём одно из двух данных суждений – непременно общее**. Например: 1) каждый студент – человек, 2) Иван – студент, следовательно, Иван – человек.

В состав силлогизма входят две посылки (или предпосылки) и заключение (или вывод). Посылки и заключение содержат в себе термины. Терминами называются понятия, которые входят в состав посылок и заключения. Терминов всего три: меньший термин (S), больший термин (P) и средний термин (M).

Структура силлогизма:

Большая посылка: (M – P).

Меньшая посылка: (S – M).

Заключение: (S – P).

Умозаключение в форме силлогизма является для нас привычной, естественной формой мышления. Эта естественность силлогизма объясняется тем, что он отражает обычные отношения вещей (например, кошелёк в сумке, деньги в кошельке, следовательно, деньги в сумке (рис. 1.2)).

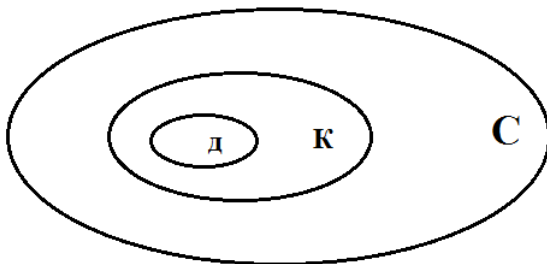


Рис. 1.2. Структура силлогизма

Силлогистическое рассуждение основывается на аксиоме силлогизма, которая имеет следующую формулировку: всё, что утверждается (или отрицается) относительно всего класса предметов, то утверждается (или отрицается) относительно части этого класса.

Индукция – это такое умозаключение, посредством которого из единичных или частных посылок мы получаем общий вывод.

Полная индукция – это такой вид индуктивного умозаключения, посредством которого мы получаем общий вывод из посылок, **исчерпывающих все случаи данного явления**.

Неполная индукция – это такой вид индуктивного умозаключения, посредством которого общий вывод получается из посылок, **не охватывающих всех случаев изучаемого явления**.

С неправильным применением неполной индукции связана логическая ошибка, известная под названием *поспешное обобщение*.

Научная индукция – это такой вид индуктивного умозаключения, посредством которого делается общий вывод относительно всех предметов какого-либо класса на основе исследования существенных свойств и причинных связей части предметов данного класса.

Все предметы, явления в мире связаны между собой, зависят друг от друга, обуславливают друг друга. Нет в мире изолированных предметов, явлений.

Причиной называется явление, которое необходимо вызывает другое явление, а это другое явление, вызванное причиной, называется **следствием**.

Аналогия – это такая форма умозаключения, в которой по сходству двух предметов в одних каких-либо признаках мы делаем заключение о сходстве этих предметов в других признаках (умозаключение от частного к частному).

Гипотеза – это предположение, которым пользуются в науке для объяснения каких-либо явлений, но достоверность которого ещё не доказана опытным путём.

Проверить гипотезу – значит: 1) установить, что следствия, которые из неё должны вытекать, действительно совпадают с наблюдаемыми явлениями; 2) показать, что применяемая нами гипотеза не противоречит другим законам, которые считаются нами истинными, другим гипотезам, которые мы приняли раньше как более или менее вероятные.

Теория – это гипотеза, которая не только не противоречит наблюдаемым фактам, но и подтверждается на практике.

Доказательство – это логическое действие, в процессе которого истинность одного какого-либо суждения обосновывается с помощью других суждений, истинность которых проверена на практике.

Состав доказательства:

1. **Тезис** – суждение или положение, истинность которого требуется доказать.
2. **Основания** (доводы или аргументы) – те суждения, истинность которых уже установлена и которые поэтому могут быть приведены в подтверждение тезиса в качестве достаточного основания.
3. **Способ доказательства** – формы связи и сочетания оснований и выводов из оснований, которые дают возможность доказать истинность тезиса.

Доказательство, в котором доводы непосредственно обосновывают истинность тезиса, называется **прямым доказательством**.

Доказательство, в котором истинность тезиса обосновывается посредством опровержения истинности других положений, называется **косвенным доказательством**.

Косвенное доказательство может быть или **апагогическим**, или **разделительным**.

Способ доказательства в апагогическом косвенном доказательстве заключается в следующем: вначале опровергается положение, противоречащее доказываемому тезису, а затем, на основании закона исключённого третьего, согласно которому из двух противоречащих высказываний одно истинно, а другое обязательно ложно, устанавливается, что доказываемый тезис необходимо истинен (в математике метод "от противного").

Разделительное косвенное доказательство применяется в тех случаях, когда известно, что доказываемый тезис входит в число фактов, которые в своей сумме полностью исчерпывают все возможные факты по данному вопросу. Способ такого доказательства заключается в следующем: отвергаются все факты, кроме одного, который и является доказываемым тезисом.

2. ИЗ ИСТОРИИ РАЗВИТИЯ ЭВМ

Современного человека окружают компьютеры (раньше их называли ЭВМ, и на них работали специально подготовленные работники – учёные и инженеры). Сейчас же компьютер многими воспринимается как бытовой прибор или средство развлечения, постепенно заменяющее телевизор. За короткий по историческим меркам промежуток времени произошла громадная эволюция вычислительной техники.

2.1. Первые ЭВМ

Электронные вычислительные машины ведут свою историю с конца 1940-х годов, когда их начали разрабатывать в научных центрах некоторых стран. Первую ламповую ЭВМ ENIAC разработали в 1945 году сотрудники Пенсильванского университета США Дж. Эккерт и Дж. Моучли. В сентябре 1950 года Конрад Цузе построил первый в Европе универсальный программируемый компьютер Z4. В СССР в ноябре 1950 года создали МЭСМ (малую электронную счетную машину) в Киевском институте электротехники под руководством С.А. Лебедева. МЭСМ содержала около 6000 электровакуумных ламп и потребляла 15 кВт. Машина могла выполнять около 3000 операций в секунду [10].

Первой советской серийной ЭВМ стала "Стрела", производившаяся с 1953 года на Московском заводе счетно-аналитических машин. "Стрела" относится к классу больших универсальных ЭВМ с трёхадресной системой команд. ЭВМ имела быстроедействие 2000–3000 операций в секунду. В качестве внешней памяти использовались два накопителя на магнитной ленте ёмкостью 200 000 слов, объём оперативной памяти – 2048 ячеек по 43 разряда. Компьютер состоял из 6200 ламп, 60 000 полупроводниковых диодов и потреблял 150 кВт энергии [10].

В 1954 году американская фирма IBM выпустила машину IBM 650, ставшую довольно популярной (всего было выпущено более 2000 машин). Она весила около 900 кг, и ещё 1350 кг весил блок питания; оба модуля имели размер примерно

1,5×0,9×1,8 м. Память на магнитном барабане хранила 2000 10-знаковых слов, позже память была увеличена до 4000 слов. По мере исполнения программы инструкции считывались прямо с магнитного барабана. В каждой инструкции был задан адрес следующей исполняемой инструкции. Использовался компилятор Symbolic Optimal Assembly Program (SOAP), который размещал инструкции по оптимальным адресам так, чтобы следующая инструкция читалась сразу и не требовалось ждать, пока барабан повернется до нужного ряда [10]. Видно, что в те годы развитие вычислительной техники в СССР и США шло одинаковыми темпами.

2.2. Принципы и схема фон Неймана

В 1946 году трое учёных – Артур Бёркс, Герман Голдстейн и Джон фон Нейман – опубликовали статью "Предварительное рассмотрение логического конструирования электронного вычислительного устройства". В статье обосновывалось применение двоичной системы счисления для представления данных в ЭВМ, выдвигалась идея использования общей памяти для программы и данных [11].

Принцип двоичного кодирования. Согласно этому принципу вся информация, поступающая и обрабатываемая в ЭВМ, представляется в двоичном виде.

Принцип однородности памяти. Программы и данные сохраняются в одной и той же памяти, поэтому ЭВМ не различает, что хранится в данной ячейке памяти – число, текст или команда. Над командами можно выполнять такие же действия, как и над данными.

Принцип адресуемости памяти. Основная память состоит из пронумерованных ячеек с произвольным доступом. Воплощена идея хранить данные в памяти ЭВМ как значения переменных – поименованных ячеек памяти.

Принцип последовательного программного управления предполагает, что программа состоит из набора операторов, которые выполняются процессором автоматически друг за другом в определённой последовательности.

Принцип жёсткости архитектуры – неизменяемость в процессе работы топологии, архитектуры, списка команд.

Компьютер должен состоять:

- из **арифметико-логического устройства**, выполняющего арифметические и логические операции;
- **устройства управления**, организующего процесс выполнения программ;
- **оперативной памяти**, где находятся данные и команды программы при работе программы;
- **постоянного запоминающего устройства**, для длительного хранения различных данных.

В современных компьютерах арифметико-логическое устройство и устройство управления объединены в **центральный процессор** – микропроцессор, представляющий собой супер-большую микросхему.

Принцип работы компьютера описывается так называемой схемой фон Неймана (рис. 2.1).

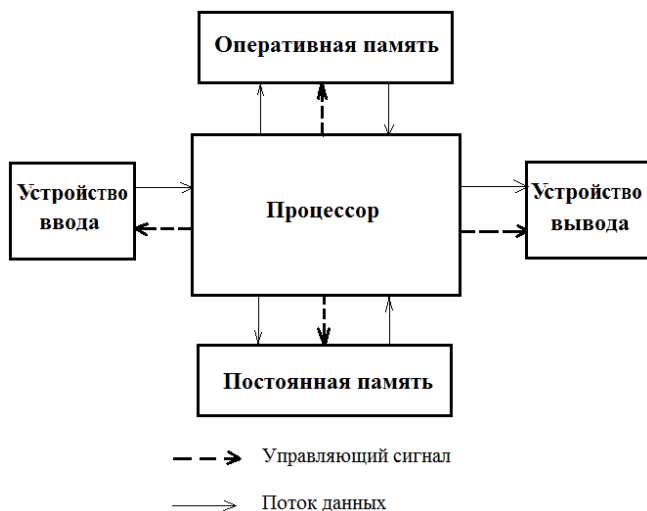


Рис. 2.1. Схема фон Неймана [11]

2.3. Компьютеры в СССР

С изобретением транзистора появилась возможность заменить электронно-вакуумные лампы на их полупроводниковые аналоги. В ЭВМ второго поколения значительно уменьшились энергопотребление и габариты, возросли надёжность, быстродействие и ресурсы памяти.

В 1966 году была создана знаменитая БЭСМ-6, лучшая отечественная ЭВМ второго поколения. На тот момент она была самой быстрой не только в СССР, но и в Европе, немного уступая американским аналогам. В архитектуре БЭСМ-6 впервые был широко использован принцип совмещения выполнения команд (до 14 одноадресных машинных команд могли находиться на разных стадиях выполнения). Механизмы прерывания, защиты памяти и другие новаторские решения позволили использовать БЭСМ-6 в мультипрограммном режиме и режиме разделения времени. ЭВМ имела 128 Кб оперативной памяти на ферритовых сердечниках и внешнюю память на магнитных барабанах и ленте. БЭСМ-6 работала с тактовой частотой 10 МГц и рекордной для того времени производительностью – около 1 млн операций в секунду. Всего было выпущено 355 этих ЭВМ [12].

Одной из первых в мире серийной персональной ЭВМ была электронная вычислительная машина "МИР-1" (расшифровывалась как машина для инженерных расчётов), разработанная в 1965 году в Институте кибернетики Академии наук УССР под руководством академика В.М. Глушкова (табл. 2.1). Это была единственная советская ЭВМ, которую купила фирма IBM [10].

Первыми советскими серийными полупроводниковыми ЭВМ стали "Весна" и "Снег", выпускавшиеся с 1964 по 1972 год. Первыми в мире серийными ЭВМ на интегральных схемах (компьютеры третьего поколения) стали советские ЭВМ "Гном", выпускавшиеся с 1965 года. В начале 1970-х разработаны системы серии "Эльбрус". "Эльбрус-2" использовался в ядерных центрах, системах противоракетной обороны и других отраслях оборонной промышленности [10].

Таблица 2.1

Технические характеристики ЭВМ МИР-1 [10]

Оперативная память	4096 12-разрядных энергонезависимых ячеек
Время выборки	Не более 2,5 мкс
Физический носитель	Ферритовые сердечники
Внешняя память	8-дорожечная перфолента
Устройство ввода	FS-1501 (до 1500 символов в секунду)
Устройство вывода	ПЛ-80 (до 80 символов в секунду)
Быстродействие	200–300 операций в секунду
Мощность	1,5 кВт
Масса	400 кг

В 1972 году были введены в строй железнодорожная система комплексной автоматизации билетно-кассовых операций АСУ "Экспресс" и система резервирования авиабилетов "Сирена", обеспечивавшие передачу и обработку больших массивов информации.

В июле – августе 1972 года минский завод им. Г.К. Орджоникидзе приступил к серийному выпуску ЭВМ третьего поколения ЕС-1020 (табл. 2.2). Производственный план предусматривал выпуск от 12 до 15 тыс. машин указанной модели в 1972–1975 гг. для удовлетворения потребностей советской науки и промышленности в вычислительной технике [13].

В табл. 2.3 и табл. 2.4 представлены некоторые характеристики ЭВМ ЕС-1020, на рис. 2.2 – схема подключения устройств в ЭВМ ЕС-1020.

Таблица 2.2

Комплектация ЭВМ ЕС-1020 [13]

Устройство	Количество
Процессор ЕС-2020	1
Накопитель на магнитной ленте ЕС-5010	4
Накопитель на сменных магнитных дисках ЕС-5056	2
Устройство управления накопителями на магнитной ленте ЕС-5511	1
Устройство управления накопителями на магнитных дисках и барабанах ЕС-5551	1
Устройство ввода с перфокарт ЕС-6012	1
Устройство ввода с перфоленты ЕС-6022	1
Устройство вывода на перфокарты ЕС-7010	1
Устройство вывода на перфоленту ЕС-7022	1
Печатающее устройство ЕС-7030	1
Пишущая машинка с блоком сопряжения со стандартным каналом ЕС-7070	1
Устройство подготовки данных на перфокартах ЕС-9010	1
Устройство подготовки данных на перфоленте ЕС-9020	1

Таблица 2.3

Емкость памяти ЭВМ ЕС-1020 [13]

Память	Размер
Основная оперативная память	64–256 Кб
Накопитель на магнитной ленте	25 Мб

Таблица 2.4

Среднее время выполнения операций процессором ЕС-1020 [13]

Операции	Среднее время выполнения, 10^{-6} с
сложения, вычитания с фиксированной запятой	20–30
сложения, вычитания с плавающей запятой	50–70
умножения с фиксированной запятой	220–350
умножения с плавающей запятой	480
деления с фиксированной запятой	400
деления с плавающей запятой	400
короткие	20–30

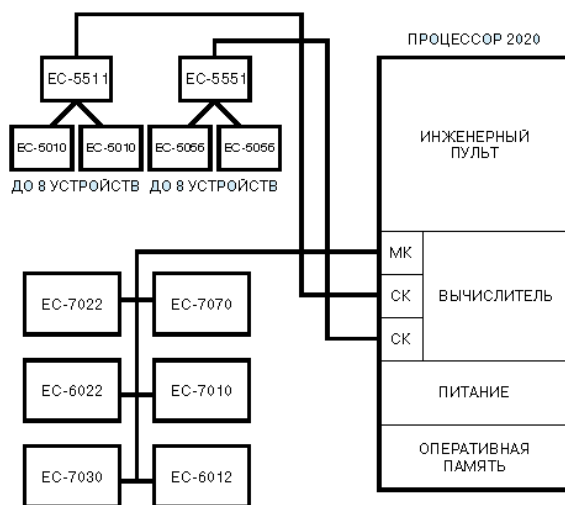


Рис. 2.2. Схема подключения устройств ЭВМ ЕС-1020 [13]:
МК – мультимплексный канал; СК – селекторный канал

Столь подробное описание приведено для того, чтобы можно было сравнить, как изменились компьютеры за почти 50 лет.

В 1980-х годах в СССР массово выпускались и персональные компьютеры, например IBM-совместимые ЕС-1840, ЕС-1841,

"Нейрон", "Ассистент". Линейки компьютеров для сферы образования – ДВК, "Агат", "Искра"; компьютеры для домашнего использования – БК-0010 в виде приставки для телевизора, "Микроша" и другие. Причем по своим характеристикам эти персональные компьютеры были не хуже американских аналогов.

Об отечественных достижениях в области вычислительной техники красноречиво говорит тот факт, что именно в СССР впервые в мире провели посадку возвращаемого с околоземной орбиты космического корабля "Буран" в автоматическом режиме без экипажа (под управлением бортового компьютера "Бисер-4") 15 ноября 1988 года [14]. В США только в 2006 году провели аналогичное испытание беспилотного орбитального самолета Boeing X-37 [15].

3. СТРУКТУРНОЕ ПРОГРАММИРОВАНИЕ

Прогресс в развитии вычислительной техники – это улучшение характеристик ЭВМ со временем. Согласно известному закону Мура, количество транзисторов в микросхемах удваивается каждые два года, и это продолжалось почти полвека. Кстати, нынешний средний смартфон в 1990 году был бы самым мощным в мире суперкомпьютером. К основным параметрам компьютеров относится быстродействие (или производительность процессора), которое измеряется в флоспах (от англ. Floating-point Operations Per Second – внесистемная единица, используемая для измерения производительности компьютеров, показывающая, сколько операций с плавающей запятой в секунду выполняет данная вычислительная система).

От объёма оперативной памяти компьютера зависит размер программы, которая может быть выполнена.

Размер ПЗУ – постоянной памяти – ограничивает количество данных, с которыми может работать исполняемая на компьютере программа.

Программирование – это процесс создания программ для компьютера. Естественно, с ростом ресурсов компьютера будут расти сложность и размер программ, соответственно усложнятся и задачи, решаемые с помощью компьютера.

Средства программирования – комплексы специальных программ, установленных на компьютере и позволяющих создавать новые программы. Это так называемые языки программирования и интегрированные системы разработки программ, которые вместе с принципами и правилами, в них заложенными, образуют технологии программирования.

С развитием вычислительной техники технологии программирования совершенствуются. На заре компьютерной эры преобладало "стихийное программирование" – одна задача программировалась в одной программе. Текст программы должен был иметь сравнительно небольшой размер, чтобы программист мог его "окинуть взглядом" полностью при редактировании.

С ростом памяти и быстродействия компьютеров появилась возможность решать более сложные задачи. Количество строк в текстах программ возросло.

Таким образом, в 70-х годах XX века не могла не возникнуть новая методология программирования для решения сложных задач на ЭВМ – структурное программирование, наибольший вклад в развитие которого внесли учёные Э.В. Дейкстра (разработал язык программирования Algol) и Н. Вирт (Pascal).

Характерной особенностью структурного программирования является возможность запрограммировать любой алгоритм, не используя оператор безусловного перехода GOTO.

Структурный подход к программированию опирается на всестороннюю алгоритмизацию программы. Отсюда следует:

- деление программы на подпрограммы;
- использование алгоритмических структур;
- строгая типизация данных.

3.1. Алгоритм и его свойства

В фундаментальной монографии Д. Кнута "Искусство программирования на ЭВМ", первый том которой называется "Основные алгоритмы" [2], всесторонне освещены вопросы разработки и анализа различных алгоритмов.

Алгоритм – совокупность определенных действий, выполнение которых в определённом порядке приведёт от исходных данных задачи к её решению.

Исходные данные задачи являются входными данными алгоритма. Для этого они должны быть представлены в определённом виде, иметь некую "понятную" для алгоритма структуру. Аналогично результат решения задачи – это выходные данные алгоритма.

Если алгоритм реализуется на компьютере, то данные должны сохраняться в памяти. Для этого используются переменные. В программировании переменная – это поименованная ячейка оперативной памяти. Чтобы компьютер при выполнении программы мог создать переменную, необходимо указать её тип, который определяется тем, как устроены её значения. Чем

сложнее структура переменной, тем больший объём памяти у соответствующей ячейки.

Алгоритм должен удовлетворять свойствам дискретности, конечности, эффективности, массовости и определённости [2]. Алгоритм состоит из отдельных формально определённых элементарных действий и достигает результата за конечное время. Алгоритм должен выполняться для любых наборов исходных данных из определённого диапазона. Таким образом, алгоритм определяет, как решать любую задачу рассматриваемого вида.

Подобное алгоритмическое свойство присуще методу математической индукции, с помощью которого доказывается не одно конкретное математическое утверждение, а сразу все утверждения определённого вида. Допустим, $P(n)$ – некоторое утверждение, касающееся натурального числа n , например, "значение выражения $n(n + 1)$ является чётным числом" или " $2^{(n+9)} > (n + 9)^3$, если n принадлежит множеству натуральных чисел". Для доказательства подобных утверждений применяется метод математической индукции, который состоит из двух этапов:

- 1) базис индукции, устанавливается справедливость утверждения $P(n)$ при $n = 1$;
- 2) предположение индукции, исходя из предположения, что при $n = k$ утверждение $P(n)$ справедливо, вытекает выполнение утверждения $P(n)$ при $n = k + 1$.

В качестве примера рассмотрим известные с древних времён равенства [2]:

$$\begin{aligned}1 &= 1^2 \\1 + 3 &= 2^2 \\1 + 3 + 5 &= 3^2 \\1 + 3 + 5 + 7 &= 4^2 \\1 + 3 + 5 + 7 + 9 &= 5^2\end{aligned}$$

В общем виде, зависящем от натурального параметра n , эти равенства можно записать таким образом:

$$1 + 3 + (2n - 1) = n^2. \quad (*)$$

В справедливости равенства (*) при определённых значениях n можно убедиться, подставив их в (*). Справедливость ра-

венства (*) для любого натурального значения n докажем методом математической индукции:

- 1) базис индукции, устанавливаем справедливость равенства (*) при $n = 1$, действительно, $1 = 1$;
- 2) предположение индукции, исходя из предположения, что при $n = k$ равенство (*) справедливо, а значит,

$$1 + 3 + (2k - 1) = k^2,$$

прибавим к обеим частям равенства $(2k + 1)$, получим

$$1 + 3 + (2k - 1) + (2k + 1) = k^2 + (2k + 1).$$

Правую часть преобразуем по формуле квадрата двучлена и получим

$$1 + 3 + (2k - 1) + (2k + 1) = (k + 1)^2,$$

что и требовалось доказать, т.е. справедливость равенства (*) при $n = k + 1$.

3.2. Алгоритмические структуры

Алгоритм состоит из простых команд (например, "сделай три шага на север"), но эти команды можно выполнять в разной последовательности.

Существуют три разные алгоритмические структуры. *Линейная структура* (рис. 3.1) предусматривает выполнение команд в строго определённом порядке.

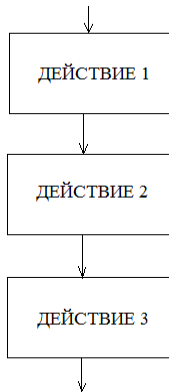


Рис. 3.1. Линейная алгоритмическая структура

Разветвляющаяся структура (рис. 3.2) позволяет заранее предусмотреть потенциальную возможность продолжения алгоритма по двум различным ветвям на альтернативной основе. При выполнении алгоритма в точке разветвления принимается решение, по которой из двух ветвей будет продолжен алгоритм (свойство однозначности не нарушается!).

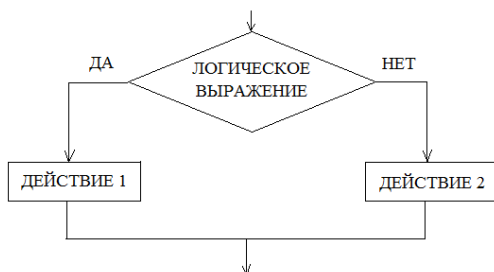


Рис. 3.2. Разветвляющаяся алгоритмическая структура

Циклическая структура (рис. 3.3) предназначена для повторения выполнения некоторой совокупности команд (одной или нескольких) необходимое количество раз.

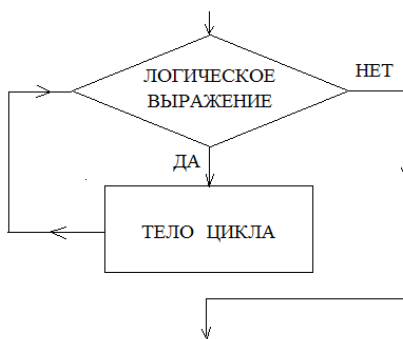


Рис. 3.3. Циклическая алгоритмическая структура

Алгоритм любого уровня сложности и объёма можно представить как комбинацию сколь угодно большого количества алгоритмических структур этих трёх видов, в том числе и вложенных друг в друга.

3.3. Алгоритмы сортировки

Сортировка – это процесс, в результате которого некие объекты располагаются по возрастанию (убыванию). Это подразумевает, что все объекты обладают признаком, для которого существует объективный критерий сравнения. Одно значение этого признака может быть меньше или больше другого, или равняться ему. Таким образом, расположение объектов по возрастанию означает, что если два объекта находятся рядом, значение этого признака у предыдущего объекта меньше или равно, чем у последующего из них.

На важность алгоритмов сортировки указывает Д. Кнут в своей знаменитой книге "Искусство программирования для ЭВМ", третий том которой так и называется "Сортировка и поиск" [3].

Основными характеристиками алгоритма сортировки являются следующие:

1. **Время**, затраченное на сортировку.
2. Необходимая для реализации алгоритма **память**.
3. Меняется или нет первоначальное взаимное расположение равных объектов, что говорит об **устойчивости** алгоритма.
4. **Естественность** алгоритма, которая показывает, как первоначальная упорядоченность объектов влияет на уменьшение времени сортировки.

Рассмотрим один из самых простых алгоритмов сортировки – *сортировку пузырьком*. Представим, что у нас имеется последовательность из N числовых значений $A_1, A_2, \dots, A_N$. Требуется произвести сортировку этой числовой последовательности. Это означает, что элементы исходной последовательности необходимо переставить в новом порядке

$$A_{K_1}, A_{K_2}, \dots, A_{K_i}, A_{K_{i+1}}, \dots, A_{K_N},$$

чтобы выполнялось условие

$$A_{K_i} \leq A_{K_{i+1}} \text{ при } i = 1, \dots, N - 1.$$

Идея алгоритма состоит в последовательной перестановке соседних элементов по возрастанию, начиная с конца последо-

вательности. За один такой проход от конца до начала последовательности на первое место переместится (всплывёт, как пузырёк) наименьший элемент последовательности. Для сортировки понадобится не более N проходов. На рис. 3.4 изображена блок-схема данного алгоритма.

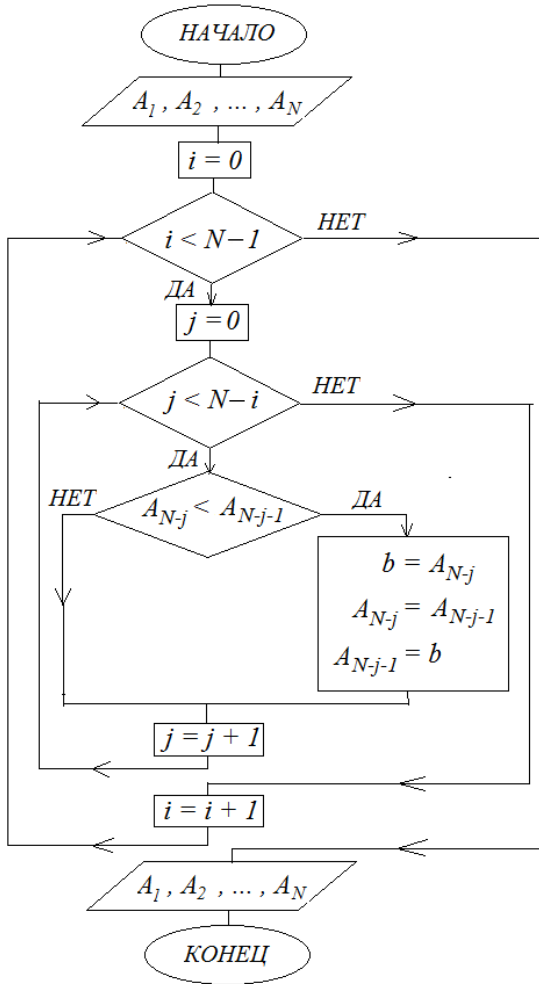


Рис. 3.4. Блок-схема алгоритма сортировки пузырьком

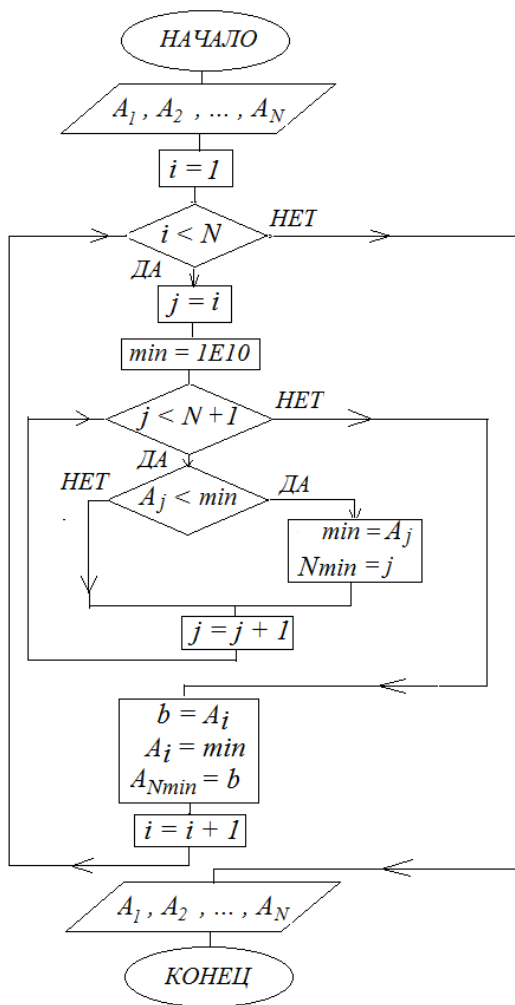


Рис. 3.5. Блок-схема алгоритма сортировки выбором

При реализации алгоритма сортировки выполняются операции сравнения значений пары элементов и взаимной перестановки двух элементов. Количество сравнений в алгоритме сортировки пузырьком равно $(N^2 - N)/2$, количество перестановок

может достигать $(N^2 - N)/2$. Таким образом, время сортировки последовательности элементов по алгоритму сортировки пузырьком пропорционально квадрату количества элементов в последовательности. На практике применяются алгоритмы, время выполнения которых пропорционально величине $N \cdot \log N$ [3]. Это максимально возможная скорость сортировки.

На рис. 3.5 представлена блок-схема алгоритма *сортировки выбором*. Данный алгоритм отличается от алгоритма сортировки пузырьком тем, что в неотсортированной части последовательности выбирается минимальный (или максимальный) элемент, который переставляется с ближайшим к отсортированной части последовательности элементом.

3.4. Технология структурного программирования

Структурное программирование – это программирование с помощью подпрограмм. **Подпрограмма – это поименованная часть программы.** Смысл использования подпрограммы заключается в следующем: в программе неоднократно возникает необходимость выполнить один и тот же набор действий (операторов). Вместо того, чтобы каждый раз помещать в программу описание этого набора действий (операторов), поступают по-другому. Заранее в отдельной подпрограмме размещают описание этого набора действий (операторов), а в программу вставляют вызов этой подпрограммы – её имя. Таким образом, определённый фрагмент кода (описание этого набора действий (операторов)) в рамках программы не повторяется, размещается в подпрограмме только один раз. Ещё одно преимущество состоит в том, что описание действий локализовано в подпрограмме и может редактироваться отдельно от программы, не загромождает программу, а применение специальным образом подобранных имён улучшает читаемость и понятность самой программы.

Различают два вида подпрограмм: **процедуры** и **подпрограммы-функции**. Различие между ними не принципиально, но механизм подпрограммы-функции позволяет помещать её вызов

в выражения в качестве операнда (удобно при организации вычислений по формулам).

Напомним очень важное понятие в программировании: **переменная в программе – это поименованная ячейка оперативной памяти**. Различают два вида переменных:

глобальные – это переменные, описанные в основной программе (область действия – вся программа, время жизни – время исполнения основной программы);

локальные – это переменные, описанные в подпрограмме (область действия – только данная подпрограмма, время жизни – время выполнения данной подпрограммы).

Глобальные переменные представляют опасность для программы, потому что в любой из подпрограмм может быть по ошибке изменено значение этой переменной. Такую ошибку очень трудно обнаружить. За локальными переменными следить очень легко по определению.

Вышеуказанный механизм применения подпрограмм, работающий в рамках одной основной программы, можно распространить и на несколько разных основных программ. Если в разных программах подразумевается использование одной и той же подпрограммы, то вместо того, чтобы помещать описание данной подпрограммы в каждую программу, поступают по-другому. Данную подпрограмму размещают в специальной образом оформленную библиотеку подпрограмм, так называемый **модуль**, а в основные программы помещают указания использовать этот модуль (подключают модуль к программе). Тогда данная подпрограмма становится доступной для этих основных программ и в них можно делать вызов данной подпрограммы. Поэтому структурное программирование называют также модульным.

Из каких именно подпрограмм должна состоять программа? Ответом на этот вопрос является ещё одна составляющая структурного программирования – **принцип нисходящего проектирования программы**. Для этого необходимо произвести декомпозицию общей задачи, которую решает программа, на несколько независимых друг от друга подзадач, которые в свою очередь

тоже разбивают на подзадачи, и так до тех пор, пока подзадачи не надо будет делить. В итоге декомпозиции общая задача преобразуется в виде иерархически устроенной системы подзадач (рис. 3.6). На каждом уровне иерархии подзадачи не зависят друг от друга и являются достаточно простыми по сравнению с общей задачей. Решение каждой подзадачи программируется в отдельной подпрограмме, начиная с самого низа иерархии. В подпрограммах верхних подзадач будут находиться вызовы подпрограмм нижних подзадач.

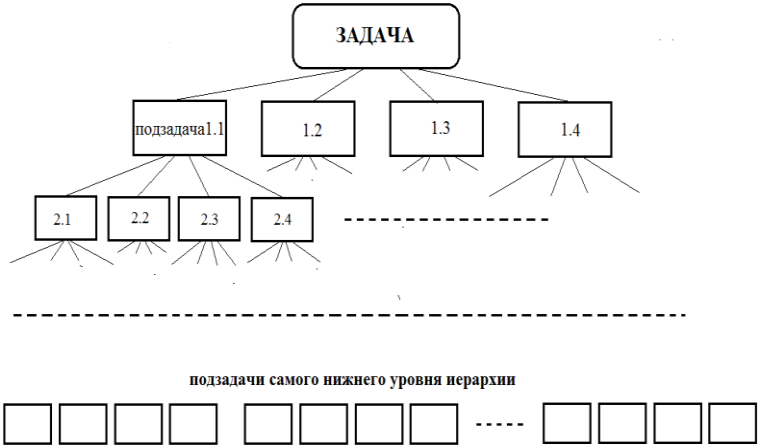


Рис. 3.6. Декомпозиция задачи

Итак, решение подзадачи программируется в подпрограмме по определённому алгоритму. Согласно свойству массовости алгоритм предусматривает решение не конкретной задачи, а возможность решить задачу определённого сорта при любых исходных данных из широкого круга возможных. Значит, и подпрограмма должна исполняться во всех соответствующих случаях. Эту возможность обеспечивает механизм параметров подпрограмм.

Формальные параметры подпрограмм – это идентификаторы (имена), которые помещают в список формальных параметров в заголовке подпрограммы. В теле подпрограммы нахо-

дятся описания действий (операторов) подпрограммы, в которых участвуют формальные параметры, т.е. тело подпрограммы – это формальное описание того, что необходимо сделать с исходными данными (формальные параметры), чтобы получить решение подзадачи (результат выполнения подпрограммы).

Фактические параметры подпрограммы – это значения, выражения или переменные, которые подставляются в список фактических параметров после имени подпрограммы при её вызове, в них содержатся конкретные экземпляры значений исходных данных, при которых необходимо решить подзадачу в данный момент.

Проведём аналогию, поясняющую, чем формальные параметры отличаются от фактических. Допустим, мы собираемся приготовить некое блюдо, используя рецепт из кулинарной книги. Рецепт – как подпрограмма, в нём указано, какие необходимы исходные продукты – это список формальных параметров, затем описывается процесс превращения ингредиентов в готовое блюдо – это тело подпрограммы.

Ещё раз подчеркнём, что рецепт – это просто текст на бумаге, формальное описание. А когда мы решили приготовить блюдо, мы должны заготовить необходимые продукты. Начало приготовления блюда по рецепту – вызов процедуры, продукты в требуемых количествах – фактические параметры, все необходимые действия и манипуляции с продуктами при готовке блюда – выполнение процедуры. Блюдо готовится не из слов, а из муки, мяса, масла и т.д.

Различают **параметры-значения**, их передают в подпрограмму по значению, и **параметры-переменные**, их передают в подпрограмму по ссылке (в таком случае у фактического параметра-переменной в результате выполнения подпрограммы можно изменить значение).

Программа, удовлетворяющая структурному подходу, приобретает высокую степень наглядности и упорядоченности. Все подробности "спрятаны" в подпрограммы, в программе – только вызовы подпрограмм, глобальных переменных нет. Вероятность

ошибки снижается, каждая подпрограмма редактируется по отдельности.

Третья составляющая структурного программирования – использование **сложных типов данных**. К простым типам данных относятся числовые целые и вещественные, символьный, логический типы, к сложным – массив, запись, множество и строковый тип.

Массив – это сложный тип данных, переменные такого типа представляют собой пронумерованную совокупность однотипных элементов. Например, переменная **ИК** типа целый массив длины 100 будет состоять из 100 ячеек, в каждую можно записать целое значение, доступ к отдельному элементу – по его номеру, скажем, 10 элемент будет **ИК(10)**.

Строка представляется как символьный массив. Переменная типа **запись** может объединять в себе элементы разных типов, называемых **полями** (аналогично понятию "запись в базы данных" – строка таблицы). Переменная типа **множество** состоит из нескольких перечислимых значений.

Индексированные переменные широко используются в математике для сокращения математических формул, аналогично в программировании тысячи числовых значений можно обработать одним оператором с параметрически проиндексированным элементом массива.

Применение принципов структурного программирования позволяет создавать программные комплексы решения задач вычислительного характера, особенно в тех случаях, когда не предусматривается последующая глубокая модернизация. При структурном подходе легче написать новую программу, чем усовершенствовать старую.

4. СОВРЕМЕННЫЕ ТЕХНОЛОГИИ ПРОГРАММИРОВАНИЯ

4.1. Объектно-ориентированное программирование

Во второй половине 1980-х годов появилась новая методология программирования, способная справиться со всё более усложнявшимися задачами – объектно-ориентированное программирование (ООП).

При структурном подходе процесс написания программы идёт от алгоритма. Объектно-ориентированный подход к проектированию программы основан на представлении предметной области задачи в виде совокупности моделей, отражающей реальность [4–6]. Например, в программном обеспечении для расчёта фундаментов зданий будут задействованы модели фундаментов, конструктивных элементов, грунтов, погодных условий и других реальных сущностей, от которых зависит расчёт.

Подобные модели являются абстракциями, появившимися в результате абстрагирования сущностей предметной области задачи. Абстракции содержат важные для правильного решения задачи признаки и характеристики реальных сущностей [6]. Например, для расчёта фундамента важно, из какого материала будут стены, а в какой цвет будет выкрашен фасад – не важно.

В результате абстрагирования вместо предметов или понятий реального мира, входящих в задачу, получаем абстракции – модели, которые содержат не все признаки и свойства представляемых ими предметов или понятия, а только существенные для разрабатываемой программной системы. В программе эти модели представляются **объектами** с заданными **атрибутами** их состояния (переменные других типов) и **методами**, определяющими их поведение (процедуры и функции).

Класс – это описание того, как устроены объекты. В ООП классы проекта всегда образуют иерархию [5]. Суперкласс по-

рождает всю иерархическую систему классов проекта, остальные классы всегда имеют предка.

Объектно-ориентированное программирование – это методология программирования, основанная на представлении программы в виде совокупности взаимодействующих объектов, каждый из которых является экземпляром определённого класса, а классы образуют иерархию наследования [6] (рис. 4.1).

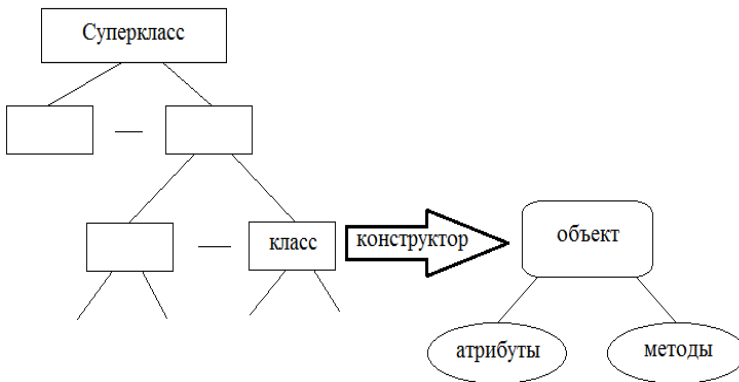


Рис. 4.1. Иерархия наследования и объект

Механизм взаимодействия объектов между собой обеспечивают **события** – свойства процедурного типа [4].

В основе ООП лежат три принципа:

Инкапсуляция – это механизм объединения данных (атрибутов состояния) и методов, работающих с ними в классе, с сокрытием деталей реализации класса и отделение его внутреннего представления от внешнего (**свойств** – атрибутов класса, доступных для использования другими классами, и **интерфейса** – набора методов класса, доступных для использования другими классами) [5]. Например, во время движения автомобиля шофёру доступны только органы управления, даже случайно он не сможет разобрать двигатель или открутить колесо.

Наследование – это механизм, дающий возможность наследовать структуру и поведение класса от другого класса [5].

Полиморфизм (означает "имеющий много форм") позволяет использовать методы с одинаковыми именами, но имеющими различную реализацию [5]. Например, у разных автомобилей могут быть двигатели, работающие на разных физических принципах. Функциональное название по разному устроенных вариатора и механической коробки передач одно – трансмиссия.

Объектно-ориентированное проектирование – это метод проектирования, сочетающий процесс объектно-ориентированной декомпозиции и систему обозначений для представления логической и физической, а также статической и динамической моделей проектируемой системы [6].

Объектно-ориентированное программирование не отрицает структурного подхода, наоборот, структурность сохраняется, подпрограммы, сложные типы данных, алгоритмичность сохраняются. При решении не сложных задач возможен вариант так называемой обёртки: структурный код помещается в объектную обёртку – получаем программный продукт, который выглядит по-современному [5].

Главная трудность в освоении объектно-ориентированной методологии программирования заключается в том, что преимущества этой методологии, её эффективность проявляются при решении очень сложных задач. Простые задачи эффективнее программировать структурно. Очень простые примеры вообще получаются короче безо всякой структурности. Тем не менее изучать объектно-ориентированный подход необходимо, особенно тем, кто собирается участвовать в реализации супер-проектов будущего.

Здесь можно привести следующую аналогию. Когда человек сам строит себе небольшой домик – это стихийный подход, структуры нет. Когда строительная организация строит типовые многоэтажные дома – это структурный подход. В организации

каждый работник – узкий специалист, он выполняет своё задание. А вот в крупной корпорации, которая создаёт суперпроект, без объектно-ориентированного подхода не обойтись. В такой корпорации работник (структурность присутствует: узкий специалист выполняет своё задание) по своему пропуску сможет занять только своё рабочее место и только определёнными (санкционированными) инструментами выполнять работу (согласно должностной инструкции). Это пример инкапсуляции, без наследования и полиморфизма крупный современный суперпроект также не обойдется.

4.2. Визуальное программирование и Visual Basic

Файл – это поименованная область памяти на постоянном носителе, например на жёстком диске или USB-накопителе.

Операционная система компьютера – это комплекс программ, обеспечивающих процесс управления аппаратными средствами, организующий работу с файлами и выполнение прикладных программ (приложений), осуществляющий ввод и вывод данных. Операционная система Windows 7 имеет графический интерфейс (интуитивно понятный, дружелюбный для пользователя). Пользователь управляет работой компьютера с помощью мыши и клавиатуры. Клавиатура требуется для набора символов, мышь – для манипуляций с элементами управления.

Элемент управления операционной системы Windows – это специальный объект для ввода данных, имеющий графическое изображение, которое взаимодействует с графическим изображением курсора мыши (в основном это разнообразные кнопки, флажки, переключатели, текстовые окна, окна и т.д.).

В операционной системе Windows все объекты отображаются на экране одинаковым образом, стандартно. Любая программа (приложение) открывается в стандартном окне, имеет стандартные элементы управления, расположенные и функционирующие согласно стандартной идеологии Windows. Поэтому разработчик программы для Windows освобождён от рутинной работы по созданию того, что устроено одинаково. И помогает

в этом **визуальное программирование** – способ создания приложений без написания программных кодов, основанный на объектно-ориентированной парадигме. Такая технология программирования предоставляет программисту наглядные средства конструирования интерфейса своей программы, установки свойств объектов этого интерфейса и автоматическую генерацию программы и заготовок текстов подпрограмм стандартных методов.

Основным элементом пользовательского интерфейса любого приложения является форма. На неё добавляются элементы управления. Форма и её элементы управления являются объектами, экземплярами соответствующих классов. Все возможные настройки внешнего вида этих объектов устанавливаются путем присвоения соответствующим свойствам определённых значений. Разработкой формы можно заниматься в режиме конструктора, меняя геометрические размеры, взаимное расположение объектов, шрифт и содержание надписей, цветовую гамму и т.д., с помощью мыши. Можно действовать и по-другому: программно присвоить необходимые значения свойствам объектов.

В настоящее время существует большое многообразие средств разработки программного обеспечения и языков программирования. Выбор в качестве предмета изучения языка программирования **Visual Basic** объясняется по крайней мере двумя причинами: 1) на данный момент операционная система Windows является самой распространённой; 2) Visual Basic как программный продукт также доступен и распространён.

4.3. Создание приложения в Visual Studio

Интегрированная среда разработки программных продуктов Visual Studio совместно с языком Visual Basic 2013 или Visual C# 2013 предназначена для создания приложений, поддерживающих различные информационные технологии на различных компьютерных платформах.

Продemonстрируем процесс создания самого простого приложения с графическим интерфейсом пользователя Windows

(Windows Forms) в интегрированной среде разработки (IDE) Visual Studio 2013 совместно с языком Visual Basic 2013.

Для начала запустим Visual Studio. В открывшемся окне отобразится начальная страница (рис. 4.2). Выберем ссылку **Создать проект**, откроется окно создания проекта (рис. 4.3). Выберем шаблон **Visual Basic**, затем **Приложение Windows Forms**.

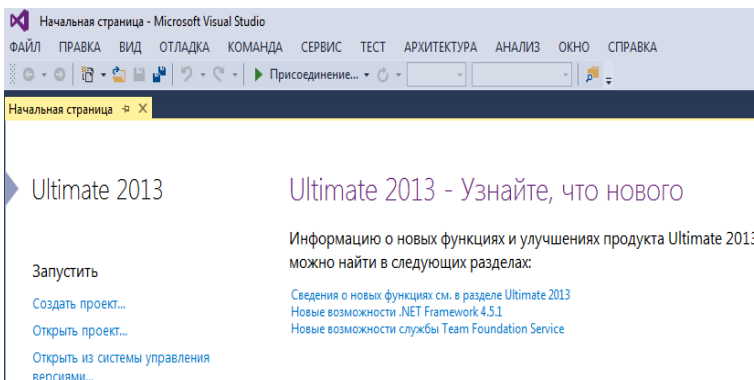


Рис. 4.2. Начальная страница Visual Studio

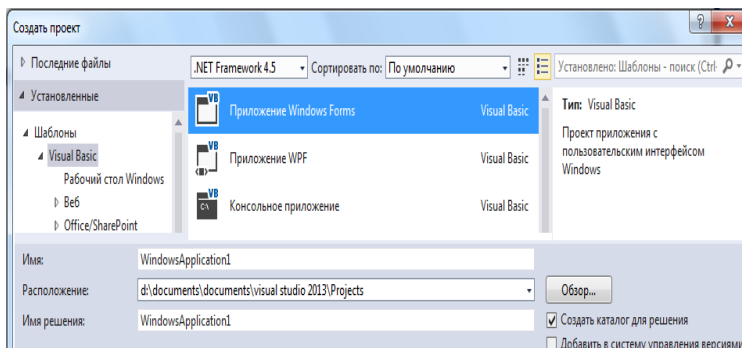


Рис. 4.3. Окно создания проекта

По умолчанию имя проекта **WindowsApplication1** (можно сразу поставить другое и поменять папку, где будут сохранены

файлы проекта). Нажимаем "ОК", откроется окно интегрированной среды разработки с конструктором формы (рис. 4.4).

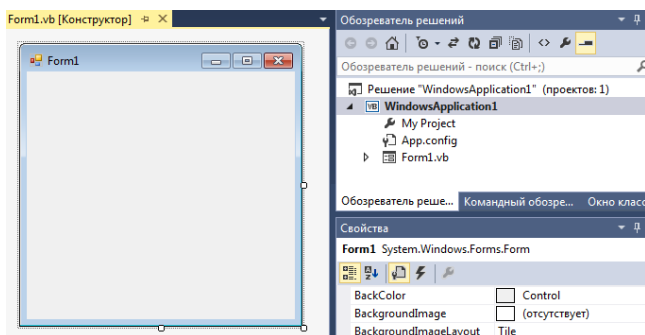


Рис. 4.4. Конструктор формы

На **Панели элементов** выбираем кнопку **Button** (рис. 4.5). Размещаем объект **Button1** на форме в нужном месте, "протягивая" мышью, задаём нужный размер.

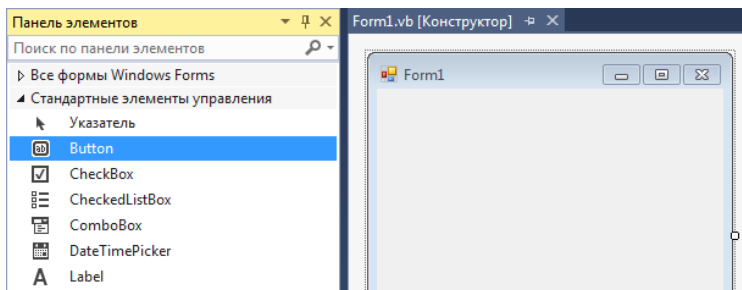


Рис. 4.5. Выбор элемента управления

Значения свойств объекта можно задать программно с помощью операторов или в режиме конструктора, где мы сейчас и находимся. Для этого открываем панель **Свойства**. Например, мы хотим изменить надпись на кнопке, в строке свойства **Text** набираем слово **ЗАПУСК** (рис. 4.6).

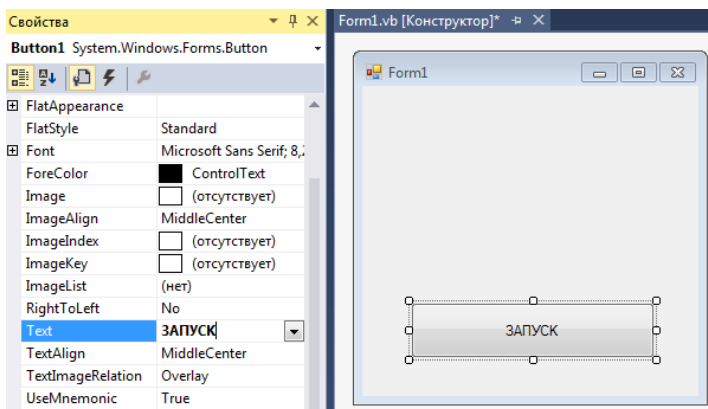


Рис. 4.6. Формирование свойств объекта в конструкторе

Таким же образом разместим на форме надпись – объект **Label1** и изменим его свойство **Font**, выберем нужный шрифт (рис. 4.7) и сделаем его невидимым на форме, установив для свойства **Visible** значение **False** (рис. 4.8).

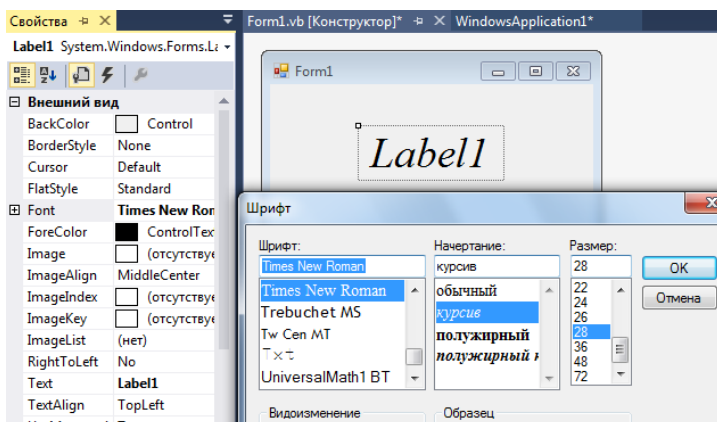


Рис. 4.7. Выбор шрифта

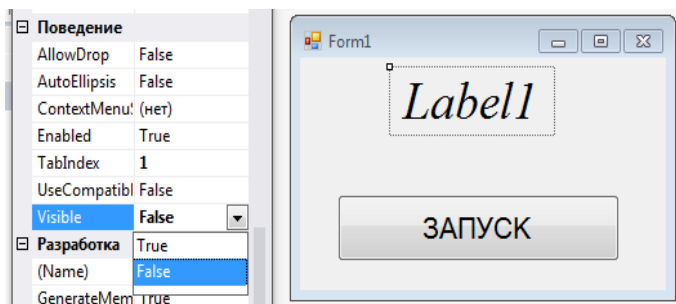


Рис. 4.8. Установка свойства **Visible**

Итак, форма простейшего демонстрационного приложения готова. В принципе этого хватит, чтобы создать приложение, при открытии которого на экране будет появляться окно с нашей формой. Что с ним можно будет делать? Только закрыть.

Чтобы приложение реагировало на действия пользователя, необходимо запрограммировать некоторые события объектов формы приложения.

Пусть в нашем приложении при нажатии на кнопку "ЗАПУСК" появляется надпись "Работает!". Для этого на панели **Свойства** открываем вкладку **События** и выбираем событие **Click** для объекта **Button1** (рис. 4.9).

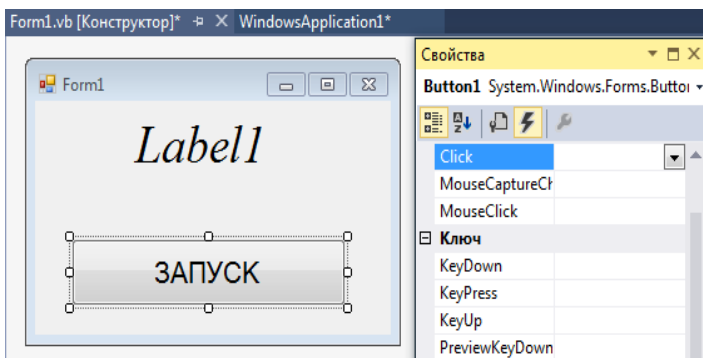


Рис. 4.9. Выбор события **Click**

Двойной щелчок мыши, и в окне программного кода появляется шаблон – заготовка подпрограммы события **Click** объекта **Button1** (рис. 4.10).

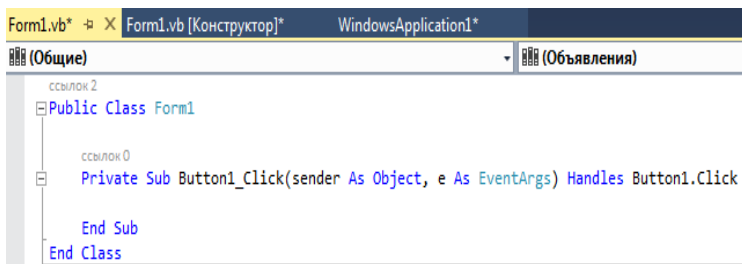


Рис. 4.10. Заготовка подпрограммы

Согласно принципу инкапсуляции ООП, мы не видим весь код, программирующий приложение, а только код, который описывает результат взаимодействия пользователя с нашим приложением (интерфейс). Механизмы, обеспечивающие весь функционал приложения, скрыты от разработчика. Помещаем в процедуру **Button1_Click** операторы, которые программно изменяют значения свойств объекта **Label1** (рис. 4.11).

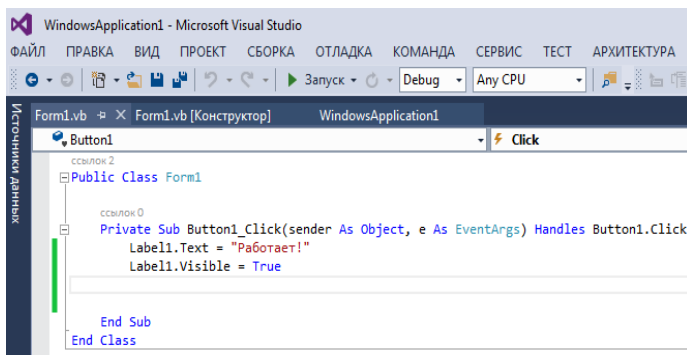


Рис. 4.11. Формирование тела подпрограммы

Всё! Готовое приложение можно запустить из среды разработки, нажав на кнопку с зелёным треугольником (рис. 4.11). На

экране компьютера появится форма нашего приложения (рис. 4.12). Протестируем его работу. Нажимаем мышью на кнопку "ЗАПУСК", на форме становится видимой надпись "Работает!", значит, наше приложение работает корректно (рис. 4.13).

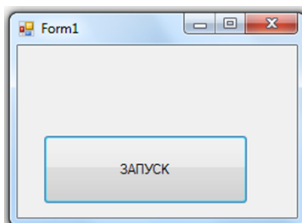


Рис. 4.12. Приложение в действии

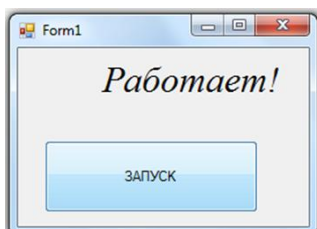


Рис. 4.13. Приложение выполняется

Выполнив команду **Построить приложение**, сгенерируем исполняемый файл приложения. Этот файл можно будет сохранить и открыть на компьютере сам по себе, без среды разработки.

Команда **Опубликовать приложение** служит для создания дистрибутива установки приложения на компьютер.

5. ПРИЁМЫ РАБОТЫ В VISUAL BASIC FOR APPLICATIONS

Visual Basic for Applications (VBA) – средство разработки проектов, интегрированное в приложения пакета Microsoft Office, состоящее из языка программирования высокого уровня и интегрированной среды разработки форм. VBA предназначен для расширения возможностей Microsoft Word, Microsoft Excel и других приложений пакета Microsoft Office, установленного почти на каждом компьютере с операционной системой Windows.

5.1. Основные команды панели Разработчик в пакете Microsoft Office

Пользователь при работе с книгой Microsoft Excel имеет возможность, не покидая приложения, перейти в интегрированную среду разработки для проектирования и программирования форм, которые помогут повысить эффективность работы с книгой. Эти формы не будут окнами (пользовательскими интерфейсами) отдельных приложений, а будут привязаны к книге Excel. Чтобы открыть окно интегрированной среды разработки VBA и приступить к конструированию формы, необходимо выполнить следующие действия. На вкладке **Разработчик** нажать на кнопку **Visual Basic**, затем нажать кнопку **Insert UserForm**.

Другой путь – сразу располагать элементы управления на листе Excel, который сам будет в качестве формы. Для этого на вкладке **Разработчик** необходимо нажать кнопку **Режим конструктора**, а затем рядом кнопку **Вставить**, после чего откроется панель с элементами управления (рис. 5.1). Для размещения элемента управления следует нажать необходимый, например **Кнопка**, поместить курсор на лист и "протянуть" до нужного размера элемента.

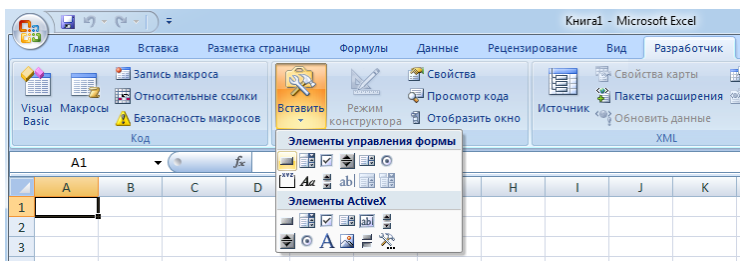


Рис. 5.1. Выбор элемента управления

Рядом с элементом управления появится окно **Назначить макрос объекту** (рис. 5.2).

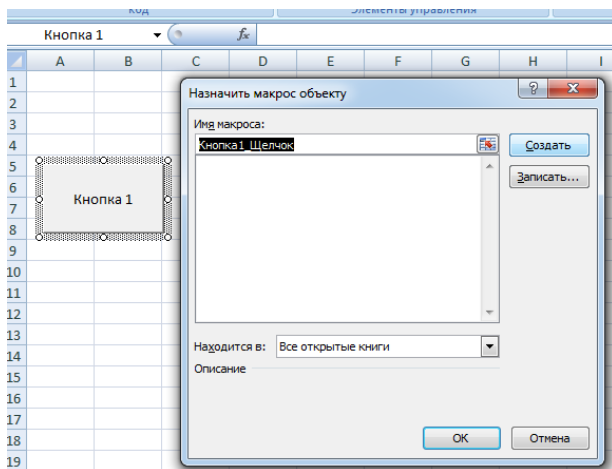


Рис. 5.2. Создание макроса

Макрос – это подпрограмма, тело которой содержит описание определённых действий, которые могут быть выполнены с книгой Excel, состоящее из команд Excel и операторов VBA. Как любая подпрограмма, макрос имеет имя.

Необходимо учитывать, что подпрограмма – это понятие структурного программирования, а любое Windows-приложение поддерживает объектно-ориентированность. Значит, макрос на

самом деле является некоторым методом некоторого объекта. Таким образом, макрос – это подпрограмма, которая выполняется, когда с определённым объектом происходит определённое событие (Например, щелчок мыши или двойной щелчок, когда указатель мыши находится на объекте. В данном случае события – это такие взаимодействия элементов управления с графическим объектом – указателем мыши, которыми пользователь управляет работой компьютера через графический интерфейс операционной системы Windows.).

Чтобы описать макрос, необходимо нажать на кнопку **Создать** (рис. 5.2). Откроется окно среды разработки (рис. 5.3) с заготовкой подпрограммы обработчика события (щелчок мышью на Кнопке1).

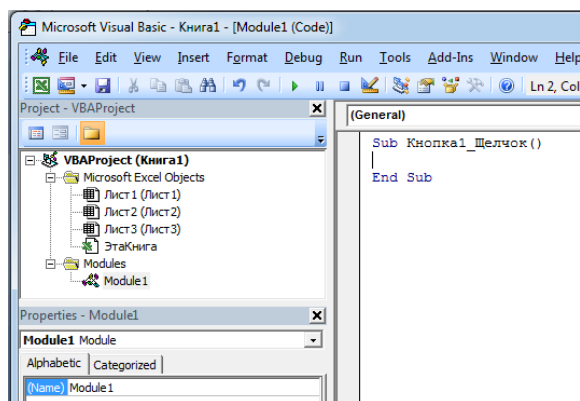


Рис. 5.3. Шаблон макроса

Напомним, что в Excel можно использовать макросы, не зная про VBA. Произвольную последовательность рутинных операций можно запомнить с помощью команды **Записать макрос** и в дальнейшем запускать в автоматическом режиме командой **Выполнить макрос**. Этот механизм позволяет быстро узнать, как выглядит описание любой команды Excel.

5.2. Применение макросов для работы с книгой Excel

Любое действие по редактированию в Excel может быть задано программно. Когда пользователь совершает какую-либо манипуляцию с данными, ячейками и объектами таблицы с помощью мыши или клавиатуры, инициируются определённые события с объектами книги Excel и выполняются соответствующие методы. Таким образом, код макроса состоит из вызовов определённых методов. Применение VBA позволяет максимально автоматизировать работу в Excel.

	A	B	C	D	E	F	G	H	I
1									
2		Музыкальные интервалы							
3		Название тона	Обозначение	Название интервала по отношению к "до"	Чистая гамма			Темперированная гамма	
4					частота по отношению к частоте "до"	интервал в септах	интервал в центах	интервал в септах	интервал в центах
5		1	2	3	4	5	6	7	8
6		до	C	тоника	1	0	0	0	0
7		ре	D	секунда	9/8	51,1	204	50,0	200
8		ми	E	терция	5/4	96,9	386	100,4	400
9		фа	F	кварта	4/3	125,0	498	125,5	500
10		соль	G	квинта	3/2	176/1	702	175,6	700
11		ля	A	секста	5/3	221,9	884	225,8	900
12		си	H	септима	15/8	273,0	1088	276,0	1100
13		до	C	октава	2	301,0	1200	301,0	1200
14									

Рис. 5.4. Таблица музыкальных интервалов

Пример 1. Следующий макрос создаёт таблицу музыкальных интервалов на чистом листе книги Excel (рис. 5.4). Из текста подпрограммы видно, как работает объектно-ориентированная модель книги Excel. Диапазон ячеек – это объект Selection, он формируется, когда выделяют одну или несколько ячеек и мышь отпускают, выполняется метод Range("Диапазон").Select. Затем у объекта Selection задаются необходимые значения его свойств, определяющих его формат и содержимое.

```

Sub Таблица_музыкальные_интервалы()
' Таблица_музыкальные_интервалы Макрос
' Сочетание клавиш: Ctrl+м
    Range("B2:I6").Select
    Selection.Borders(xlDiagonalDown).LineStyle = xlNone
    Selection.Borders(xlDiagonalUp).LineStyle = xlNone
    With Selection.Borders(xlEdgeLeft)
        .LineStyle = xlContinuous
        .ColorIndex = xlAutomatic
        .TintAndShade = 0
        .Weight = xlMedium
    End With
    With Selection.Borders(xlEdgeTop)
        .LineStyle = xlContinuous
        .ColorIndex = xlAutomatic
        .TintAndShade = 0
        .Weight = xlMedium
    End With
    With Selection.Borders(xlEdgeBottom)
        .LineStyle = xlContinuous
        .ColorIndex = xlAutomatic
        .TintAndShade = 0
        .Weight = xlMedium
    End With
    With Selection.Borders(xlEdgeRight)
        .LineStyle = xlContinuous
        .ColorIndex = xlAutomatic
        .TintAndShade = 0
        .Weight = xlMedium
    End With
    With Selection.Borders(xlInsideVertical)
        .LineStyle = xlContinuous
        .ColorIndex = xlAutomatic
        .TintAndShade = 0
        .Weight = xlThin
    End With
    With Selection.Borders(xlInsideHorizontal)

```

```

.LineStyle = xlContinuous
.ColorIndex = xlAutomatic
.TintAndShade = 0
.Weight = xlThin
End With
Range("B2:I2").Select
With Selection
    .HorizontalAlignment = xlGeneral
    .VerticalAlignment = xlBottom
    .WrapText = False
    .Orientation = 0
    .AddIndent = False
    .IndentLevel = 0
    .ShrinkToFit = False
    .ReadingOrder = xlContext
    .MergeCells = True
End With
Range("B3:B4").Select
With Selection
    .HorizontalAlignment = xlGeneral
    .VerticalAlignment = xlBottom
    .WrapText = False
    .Orientation = 0
    .AddIndent = False
    .IndentLevel = 0
    .ShrinkToFit = False
    .ReadingOrder = xlContext
    .MergeCells = True
End With
Range("C3:C4").Select
With Selection
    .HorizontalAlignment = xlGeneral
    .VerticalAlignment = xlBottom
    .WrapText = False
    .Orientation = 0
    .AddIndent = False
    .IndentLevel = 0

```

```

        .ShrinkToFit = False
        .ReadingOrder = xlContext
        .MergeCells = True
    End With
    Range("D3:D4").Select
    With Selection
        .HorizontalAlignment = xlGeneral
        .VerticalAlignment = xlBottom
        .WrapText = False
        .Orientation = 0
        .AddIndent = False
        .IndentLevel = 0
        .ShrinkToFit = False
        .ReadingOrder = xlContext
        .MergeCells = True
    End With
    Range("E3:G3").Select
    With Selection
        .HorizontalAlignment = xlGeneral
        .VerticalAlignment = xlBottom
        .WrapText = False
        .Orientation = 0
        .AddIndent = False
        .IndentLevel = 0
        .ShrinkToFit = False
        .ReadingOrder = xlContext
        .MergeCells = True
    End With
    Range("H3:I3").Select
    With Selection
        .HorizontalAlignment = xlGeneral
        .VerticalAlignment = xlBottom
        .WrapText = False
        .Orientation = 0
        .AddIndent = False
        .IndentLevel = 0
        .ShrinkToFit = False

```

```

        .ReadingOrder = xlContext
        .MergeCells = True
    End With
    Rows("4:4").RowHeight = 97.5
    Range("B3:D4").Select
    With Selection
        .HorizontalAlignment = xlGeneral
        .VerticalAlignment = xlBottom
        .WrapText = False
        .Orientation = 90
        .AddIndent = False
        .IndentLevel = 0
        .ShrinkToFit = False
        .ReadingOrder = xlContext
    End With
    Range("E4:I4").Select
    With Selection
        .HorizontalAlignment = xlGeneral
        .VerticalAlignment = xlBottom
        .WrapText = False
        .Orientation = 90
        .AddIndent = False
        .IndentLevel = 0
        .ShrinkToFit = False
        .ReadingOrder = xlContext
        .MergeCells = False
    End With
    Range("B2:I2").Select
    ActiveCell.FormulaR1C1 = "Музыкальные интервалы"
    Range("E3:G3").Select
    ActiveCell.FormulaR1C1 = "Чистая гамма"
    Range("H3:I3").Select
    ActiveCell.FormulaR1C1 = "Темперированная гамма"
    Range("B3:B4").Select
    Rows("3:3").RowHeight = 32.25
    Range("H3:I3").Select
    With Selection

```

```

.HorizontalAlignment = xlGeneral
.VerticalAlignment = xlBottom
.WrapText = True
.Orientation = 0
.AddIndent = False
.IndentLevel = 0
.ShrinkToFit = False
.ReadingOrder = xlContext
.MergeCells = True
End With
Range("B3:B4").Select
ActiveCell.FormulaR1C1 = "Название тона"
With ActiveCell.Characters(Start:=1, Length:=13).Font
.Name = "Calibri"
.FontStyle = "обычный"
.Size = 11
.Strikethrough = False
.Superscript = False
.Subscript = False
.OutlineFont = False
.Shadow = False
.Underline = xlUnderlineStyleNone
.ThemeColor = xlThemeColorLight1
.TintAndShade = 0
.ThemeFont = xlThemeFontMinor
End With
Range("C3:C4").Select
ActiveCell.FormulaR1C1 = "Обозначение"
With ActiveCell.Characters(Start:=1, Length:=11).Font
.Name = "Calibri"
.FontStyle = "обычный"
.Size = 11
.Strikethrough = False
.Superscript = False
.Subscript = False
.OutlineFont = False
.Shadow = False

```

```

.Underline = xlUnderlineStyleNone
.ThemeColor = xlThemeColorLight1
.TintAndShade = 0
.ThemeFont = xlThemeFontMinor
End With
Range("D3:D4").Select
ActiveCell.FormulaR1C1 = "Название интервала
                        по отношению к "до""
With ActiveCell.Characters(Start:=1, Length:=38).Font
.Name = "Calibri"
.FontStyle = "обычный"
.Size = 11
.Strikethrough = False
.Superscript = False
.Subscript = False
.OutlineFont = False
.Shadow = False
.Underline = xlUnderlineStyleNone
.ThemeColor = xlThemeColorLight1
.TintAndShade = 0
.ThemeFont = xlThemeFontMinor
End With
Range("D3:D4").Select
With Selection
.HorizontalAlignment = xlGeneral
.VerticalAlignment = xlBottom
.WrapText = True
.Orientation = 90
.AddIndent = False
.IndentLevel = 0
.ShrinkToFit = False
.ReadingOrder = xlContext
.MergeCells = True
End With
Range("E4").Select
ActiveCell.FormulaR1C1 = "частота по отношению
                        к частоте"до""

```

```

With ActiveCell.Characters(Start:=1, Length:=34).Font
    .Name = "Calibri"
    .FontStyle = "обычный"
    .Size = 11
    .Strikethrough = False
    .Superscript = False
    .Subscript = False
    .OutlineFont = False
    .Shadow = False
    .Underline = xlUnderlineStyleNone
    .ThemeColor = xlThemeColorLight1
    .TintAndShade = 0
    .ThemeFont = xlThemeFontMinor
End With
Range("F4").Select
ActiveCell.FormulaR1C1 = "интервал в соварах"
With ActiveCell.Characters(Start:=1, Length:=18).Font
    .Name = "Calibri"
    .FontStyle = "обычный"
    .Size = 11
    .Strikethrough = False
    .Superscript = False
    .Subscript = False
    .OutlineFont = False
    .Shadow = False
    .Underline = xlUnderlineStyleNone
    .ThemeColor = xlThemeColorLight1
    .TintAndShade = 0
    .ThemeFont = xlThemeFontMinor
End With
Range("G4").Select
ActiveCell.FormulaR1C1 = "интервал в центрах"
With ActiveCell.Characters(Start:=1, Length:=17).Font
    .Name = "Calibri"
    .FontStyle = "обычный"
    .Size = 11

```

```

.Strikethrough = False
.Superscript = False
.Subscript = False
.OutlineFont = False
.Shadow = False
.Underline = xlUnderlineStyleNone
.ThemeColor = xlThemeColorLight1
.TintAndShade = 0
.ThemeFont = xlThemeFontMinor
End With
Range("F4:G4").Select
Selection.Copy
Range("H4:I4").Select
ActiveSheet.Paste
Range("E4:I4").Select
Application.CutCopyMode = False
With Selection
    .HorizontalAlignment = xlGeneral
    .VerticalAlignment = xlBottom
    .WrapText = True
    .Orientation = 90
    .AddIndent = False
    .IndentLevel = 0
    .ShrinkToFit = False
    .ReadingOrder = xlContext
    .MergeCells = False
End With
Range("B2:I4").Select
With Selection
    .HorizontalAlignment = xlCenter
    .VerticalAlignment = xlCenter
    .AddIndent = False
    .IndentLevel = 0
    .ShrinkToFit = False
    .ReadingOrder = xlContext
End With

```

```

Range("B5").Select
ActiveCell.FormulaR1C1 = "1"
Range("C5").Select
ActiveCell.FormulaR1C1 = "2"
Range("B5:C5").Select
Selection.AutoFill
    Destination:=Range("B5:I5"), Type:=xlFillDefault
Range("B5:I5").Select
Range("B6").Select
ActiveCell.FormulaR1C1 = "до"
Range("B7").Select
ActiveCell.FormulaR1C1 = "ре"
Range("B8").Select
ActiveCell.FormulaR1C1 = "ми"
Range("B9").Select
ActiveCell.FormulaR1C1 = "фа"
Range("B10").Select
ActiveCell.FormulaR1C1 = "соль"
Range("B11").Select
ActiveCell.FormulaR1C1 = "ля"
Range("B12").Select
ActiveCell.FormulaR1C1 = "си"
Range("B13").Select
ActiveCell.FormulaR1C1 = "до"
Range("C6").Select
ActiveCell.FormulaR1C1 = "C"
Range("C7").Select
ActiveCell.FormulaR1C1 = "D"
Range("C8").Select
ActiveCell.FormulaR1C1 = "E"
Range("C9").Select
ActiveCell.FormulaR1C1 = "F"
Range("C10").Select
ActiveCell.FormulaR1C1 = "G"
Range("C11").Select
ActiveCell.FormulaR1C1 = "A"

```

```
Range("C12").Select
ActiveCell.FormulaR1C1 = "H"
Range("C13").Select
ActiveCell.FormulaR1C1 = "C"
Range("D6").Select
ActiveCell.FormulaR1C1 = "тоника"
Range("D7").Select
ActiveCell.FormulaR1C1 = "секунда"
Range("D8").Select
ActiveCell.FormulaR1C1 = "терция"
Range("D9").Select
ActiveCell.FormulaR1C1 = "кварта"
Range("D10").Select
ActiveCell.FormulaR1C1 = "квинта"
Range("D11").Select
ActiveCell.FormulaR1C1 = "секста"
Range("D12").Select
ActiveCell.FormulaR1C1 = "септима"
Range("D13").Select
ActiveCell.FormulaR1C1 = "октава"
Range("E6").Select
ActiveCell.FormulaR1C1 = "1"
Range("E7").Select
ActiveCell.FormulaR1C1 = "8/9/2019"
Range("E6:E13").Select
Selection.NumberFormat = "@"
Range("E7").Select
ActiveCell.FormulaR1C1 = "9/8"
Range("E8").Select
ActiveCell.FormulaR1C1 = "5/4"
Range("E9").Select
ActiveCell.FormulaR1C1 = "4/3"
Range("E10").Select
ActiveCell.FormulaR1C1 = "3/2"
Range("E11").Select
ActiveCell.FormulaR1C1 = "5/3"
```

Range("E12").Select
ActiveCell.FormulaR1C1 = "15/8"
Range("E13").Select
ActiveCell.FormulaR1C1 = "2"
Range("F6").Select
ActiveCell.FormulaR1C1 = "0"
Range("F7").Select
ActiveCell.FormulaR1C1 = "51,1"
Range("F8").Select
ActiveCell.FormulaR1C1 = "96,9"
Range("F9").Select
ActiveCell.FormulaR1C1 = "125,0"
Range("F10").Select
ActiveCell.FormulaR1C1 = "176/1"
Range("F11").Select
ActiveCell.FormulaR1C1 = "221,9"
Range("F12").Select
ActiveCell.FormulaR1C1 = "273,0"
Range("F13").Select
ActiveCell.FormulaR1C1 = "301,0"
Range("G6").Select
ActiveCell.FormulaR1C1 = "0"
Range("G7").Select
ActiveCell.FormulaR1C1 = "204"
Range("G8").Select
ActiveCell.FormulaR1C1 = "386"
Range("G9").Select
ActiveCell.FormulaR1C1 = "498"
Range("G10").Select
ActiveCell.FormulaR1C1 = "702"
Range("G11").Select
ActiveCell.FormulaR1C1 = "884"
Range("G12").Select
ActiveCell.FormulaR1C1 = "1088"
Range("G13").Select
ActiveCell.FormulaR1C1 = "1200"

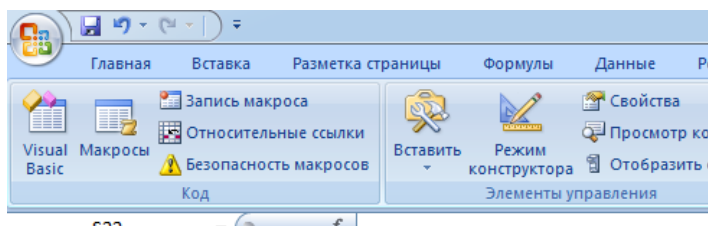
```
Range("H6").Select
ActiveCell.FormulaR1C1 = "0"
Range("H7").Select
ActiveCell.FormulaR1C1 = "50,0"
Range("H8").Select
ActiveCell.FormulaR1C1 = "100,4"
Range("H9").Select
ActiveCell.FormulaR1C1 = "125,5"
Range("H10").Select
ActiveCell.FormulaR1C1 = "175,6"
Range("H11").Select
ActiveCell.FormulaR1C1 = "225,8"
Range("H12").Select
ActiveCell.FormulaR1C1 = "276,0"
Range("H13").Select
ActiveCell.FormulaR1C1 = "1"
Range("H13").Select
ActiveCell.FormulaR1C1 = "301,0"
Range("I6").Select
ActiveCell.FormulaR1C1 = "0"
Range("I7").Select
ActiveCell.FormulaR1C1 = "200"
Range("I8").Select
ActiveCell.FormulaR1C1 = "400"
Range("I9").Select
ActiveCell.FormulaR1C1 = "500"
Range("I10").Select
ActiveCell.FormulaR1C1 = "700"
Range("I11").Select
ActiveCell.FormulaR1C1 = "900"
Range("I12").Select
ActiveCell.FormulaR1C1 = "1100"
Range("I13").Select
ActiveCell.FormulaR1C1 = "1200"
Range("B5:I13").Select
With Selection
```

```

.HorizontalAlignment = xlCenter
.VerticalAlignment = xlCenter
.WrapText = False
.Orientation = 0
.AddIndent = False
.IndentLevel = 0
.ShrinkToFit = False
.ReadingOrder = xlContext
.MergeCells = False
End With
End Sub

```

Пример 2. Макрос, который создаёт таблицу значений функции на чистом листе книги Excel (рис. 5.5).



	A	B	C	D	E	F	G
1		График функции $y=x^3-9x^2+23x-15$					
2							
3		Таблица сеточных значений функции					
4		на сетке $x_i=x_0+ih$, $i=0,1,2,...N$					
5		$x_0=$	-2				
6		$x_N=$	8				
7		$N=$	100				
8		$h=$	0.1				
9		X	Y				
10	0	-2	-105				
11	1	-1.9	-98.049				
12	2	-1.8	-91.392				
13	3	-1.7	-85.023				
14	4	-1.6	-78.936				

Рис. 5.5. Таблица значений функции

```

Sub таблица_значений_функции ()
' построение_графика Макрос
' Сочетание клавиш: Ctrl+г
    Range("B1").Select
    ActiveCell.FormulaR1C1 = "График функции
                                 $y=x^3-9x^2+23x-15$ "
    With ActiveCell.Characters(Start:=1, Length:=18).Font
        .Name = "Calibri"
        .FontStyle = "обычный"
        .Size = 11
        .Strikethrough = False
        .Superscript = False
        .Subscript = False
        .OutlineFont = False
        .Shadow = False
        .Underline = xlUnderlineStyleNone
        .ThemeColor = xlThemeColorLight1
        .TintAndShade = 0
        .ThemeFont = xlThemeFontMinor
    End With
    With ActiveCell.Characters(Start:=19, Length:=1).Font
        .Name = "Calibri"
        .FontStyle = "обычный"
        .Size = 11
        .Strikethrough = False
        .Superscript = True
        .Subscript = False
        .OutlineFont = False
        .Shadow = False
        .Underline = xlUnderlineStyleNone
        .ThemeColor = xlThemeColorLight1
        .TintAndShade = 0
        .ThemeFont = xlThemeFontMinor
    End With
    With ActiveCell.Characters(Start:=20, Length:=3).Font
        .Name = "Calibri"

```

```

.FontStyle = "обычный"
.Size = 11
.Strikethrough = False
.Superscript = False
.Subscript = False
.OutlineFont = False
.Shadow = False
.Underline = xlUnderlineStyleNone
.ThemeColor = xlThemeColorLight1
.TintAndShade = 0
.ThemeFont = xlThemeFontMinor
End With
With ActiveCell.Characters(Start:=23, Length:=1).Font
.Name = "Calibri"
.FontStyle = "обычный"
.Size = 11
.Strikethrough = False
.Superscript = True
.Subscript = False
.OutlineFont = False
.Shadow = False
.Underline = xlUnderlineStyleNone
.ThemeColor = xlThemeColorLight1
.TintAndShade = 0
.ThemeFont = xlThemeFontMinor
End With
With ActiveCell.Characters(Start:=24, Length:=7).Font
.Name = "Calibri"
.FontStyle = "обычный"
.Size = 11
.Strikethrough = False
.Superscript = False
.Subscript = False
.OutlineFont = False
.Shadow = False
.Underline = xlUnderlineStyleNone

```

```

.ThemeColor = xlThemeColorLight1
.TintAndShade = 0
.ThemeFont = xlThemeFontMinor
End With
Range("B3").Select
ActiveCell.FormulaR1C1 = "Таблица сеточных значений
                          функции"
Range("B4").Select
ActiveCell.FormulaR1C1 = "на сетке  $x_i=x_0+ih$ ,  $i=0,1,$ 
                           $2,\dots,N$ "
With ActiveCell.Characters(Start:=1, Length:=10).Font
.Name = "Calibri"
.FontStyle = "обычный"
.Size = 11
.Strikethrough = False
.Superscript = False
.Subscript = False
.OutlineFont = False
.Shadow = False
.Underline = xlUnderlineStyleNone
.ThemeColor = xlThemeColorLight1
.TintAndShade = 0
.ThemeFont = xlThemeFontMinor
End With
With ActiveCell.Characters(Start:=11, Length:=1).Font
.Name = "Calibri"
.FontStyle = "обычный"
.Size = 11
.Strikethrough = False
.Superscript = False
.Subscript = True
.OutlineFont = False
.Shadow = False
.Underline = xlUnderlineStyleNone
.ThemeColor = xlThemeColorLight1
.TintAndShade = 0

```

```

.ThemeFont = xlThemeFontMinor
End With
With ActiveCell.Characters(Start:=12, Length:=2).Font
.Name = "Calibri"
.FontStyle = "обычный"
.Size = 11
.Strikethrough = False
.Superscript = False
.Subscript = False
.OutlineFont = False
.Shadow = False
.Underline = xlUnderlineStyleNone
.ThemeColor = xlThemeColorLight1
.TintAndShade = 0
.ThemeFont = xlThemeFontMinor
End With
With ActiveCell.Characters(Start:=14, Length:=1).Font
.Name = "Calibri"
.FontStyle = "обычный"
.Size = 11
.Strikethrough = False
.Superscript = False
.Subscript = True
.OutlineFont = False
.Shadow = False
.Underline = xlUnderlineStyleNone
.ThemeColor = xlThemeColorLight1
.TintAndShade = 0
.ThemeFont = xlThemeFontMinor
End With
With ActiveCell.Characters(Start:=15, Length:=18).Font
.Name = "Calibri"
.FontStyle = "обычный"
.Size = 11
.Strikethrough = False
.Superscript = False

```

```

.Subscript = False
.OutlineFont = False
.Shadow = False
.Underline = xlUnderlineStyleNone
.ThemeColor = xlThemeColorLight1
.TintAndShade = 0
.ThemeFont = xlThemeFontMinor
End With
Range("B5").Select
ActiveCell.FormulaR1C1 = "x0="
With ActiveCell.Characters(Start:=1, Length:=1).Font
.Name = "Calibri"
.FontStyle = "обычный"
.Size = 11
.Strikethrough = False
.Superscript = False
.Subscript = False
.OutlineFont = False
.Shadow = False
.Underline = xlUnderlineStyleNone
.ThemeColor = xlThemeColorLight1
.TintAndShade = 0
.ThemeFont = xlThemeFontMinor
End With
With ActiveCell.Characters(Start:=2, Length:=1).Font
.Name = "Calibri"
.FontStyle = "обычный"
.Size = 11
.Strikethrough = False
.Superscript = False
.Subscript = True
.OutlineFont = False
.Shadow = False
.Underline = xlUnderlineStyleNone
.ThemeColor = xlThemeColorLight1
.TintAndShade = 0

```

```

.ThemeFont = xlThemeFontMinor
End With
With ActiveCell.Characters(Start:=3, Length:=1).Font
.Name = "Calibri"
.FontStyle = "обычный"
.Size = 11
.Strikethrough = False
.Superscript = False
.Subscript = False
.OutlineFont = False
.Shadow = False
.Underline = xlUnderlineStyleNone
.ThemeColor = xlThemeColorLight1
.TintAndShade = 0
.ThemeFont = xlThemeFontMinor
End With
Range("B6").Select
ActiveCell.FormulaR1C1 = "xN="
With ActiveCell.Characters(Start:=1, Length:=1).Font
.Name = "Calibri"
.FontStyle = "обычный"
.Size = 11
.Strikethrough = False
.Superscript = False
.Subscript = False
.OutlineFont = False
.Shadow = False
.Underline = xlUnderlineStyleNone
.ThemeColor = xlThemeColorLight1
.TintAndShade = 0
.ThemeFont = xlThemeFontMinor
End With
With ActiveCell.Characters(Start:=2, Length:=1).Font
.Name = "Calibri"
.FontStyle = "обычный"
.Size = 11

```

```

.Strikethrough = False
.Superscript = False
.Subscript = True
.OutlineFont = False
.Shadow = False
.Underline = xlUnderlineStyleNone
.ThemeColor = xlThemeColorLight1
.TintAndShade = 0
.ThemeFont = xlThemeFontMinor
End With
With ActiveCell.Characters(Start:=3, Length:=1).Font
.Name = "Calibri"
.FontStyle = "обычный"
.Size = 11
.Strikethrough = False
.Superscript = False
.Subscript = False
.OutlineFont = False
.Shadow = False
.Underline = xlUnderlineStyleNone
.ThemeColor = xlThemeColorLight1
.TintAndShade = 0
.ThemeFont = xlThemeFontMinor
End With
Range("B7").Select
ActiveCell.FormulaR1C1 = "N="
Range("B5:B7").Select
With Selection
.HorizontalAlignment = xlRight
.VerticalAlignment = xlBottom
.WrapText = False
.Orientation = 0
.AddIndent = False
.IndentLevel = 0
.ShrinkToFit = False
.ReadingOrder = xlContext

```

```

.MergeCells = False
End With
Range("B9").Select
ActiveCell.FormulaR1C1 = "X"
Range("C9").Select
ActiveCell.FormulaR1C1 = "Y"
Range("B9:C68").Select
With Selection
    .HorizontalAlignment = xlCenter
    .VerticalAlignment = xlBottom
    .WrapText = False
    .Orientation = 0
    .AddIndent = False
    .IndentLevel = 0
    .ShrinkToFit = False
    .ReadingOrder = xlContext
    .MergeCells = False
End With
Range("C5").Select
ActiveCell.FormulaR1C1 = "-2"
Range("C6").Select
ActiveCell.FormulaR1C1 = "8"
Range("C7").Select
ActiveCell.FormulaR1C1 = "100"
Range("B10").Select
ActiveCell.FormulaR1C1 = "=R[-5]C[1]"
Range("B8").Select
ActiveCell.FormulaR1C1 = "h="
Range("C8").Select
ActiveCell.FormulaR1C1 = "=(R[-2]C-R[-3]C)/R[-1]C"
Range("A10").Select
ActiveCell.FormulaR1C1 = "0"
Range("A11").Select
ActiveCell.FormulaR1C1 = "1"
Range("A12").Select
ActiveCell.FormulaR1C1 = "2"

```

```

Range("A11:A12").Select
Selection.AutoFill
Destination:=Range("A11:A110"), Type:=xlFillDefault
Range("A11:A110").Select
ActiveWindow.SmallScroll Down:=-81
Range("B11").Select
ActiveCell.FormulaR1C1 = "=R[-6]C[1]+RC[-1]*R8C3"
Range("B11").Select
ActiveCell.FormulaR1C1 = "=R[-6]C3+RC[-1]*R8C3"
Range("B11").Select
ActiveCell.FormulaR1C1 = "=R[-6]C[1]+R11C1*R[-3]C[1]"
Range("B11").Select
Selection.AutoFill
Destination:=Range("B11:B12"), Type:=xlFillDefault
Range("B11:B12").Select
Range("B11").Select
ActiveCell.FormulaR1C1 = "=R[-6]C3+RC[-1]*R8C3"
Range("B12").Select
Selection.ClearContents
Range("B11").Select
Selection.AutoFill
Destination:=Range("B11:B12"), Type:=xlFillDefault
Range("B11:B12").Select
Range("B11").Select
ActiveCell.FormulaR1C1 = "=R5C3+RC[-1]*R8C3"
Range("B12").Select
Selection.ClearContents
Range("B11").Select
Selection.AutoFill
Destination:=Range("B11:B12"), Type:=xlFillDefault
Range("B11:B12").Select
Selection.AutoFill
Destination:=Range("B11:B110"), Type:=xlFillDefault
Range("B11:B110").Select
ActiveWindow.SmallScroll Down:=-78
Range("C10").Select

```

```

ActiveCell.FormulaR1C1 = "=RC[-1]^3-9*RC[-1]^2+23*RC[-1]-15"

Range("C10").Select
Selection.AutoFill
Destination:=Range("C10:C110"), Type:=xlFillDefault
Range("C10:C110").Select
ActiveWindow.SmallScroll Down:=-90
End Sub

```

Пример 3. Макрос, который строит график функции по данным из таблицы значений этой функции (рис. 5.6).

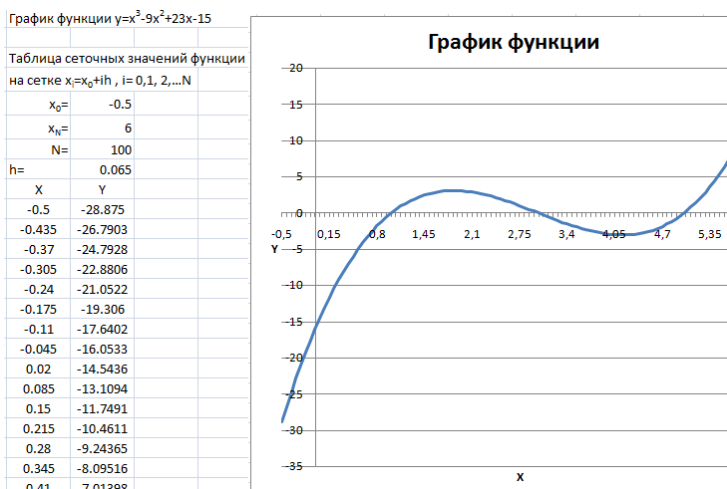


Рис. 5.6. График функции

```

Sub построение_графика()
' Сочетание клавиш: Ctrl+f
ActiveSheet.Shapes.AddChart.Select
ActiveChart.ChartType = xlLine
ActiveChart.SetSourceData Source:=Range("C10:C110")
ActiveChart.SeriesCollection(1).XValues =
    "=Лист1!$B$10:$B$110"

```

```

ActiveWindow.SmallScroll Down:=-93
ActiveChart.PlotArea.Select
ActiveSheet.ChartObjects("Диаграмма 2").Activate
ActiveChart.ChartArea.Select
ActiveSheet.ChartObjects("Диаграмма 2").Activate
ActiveSheet.ChartObjects("Диаграмма 2").Activate
ActiveChart.PlotArea.Select
ActiveChart.HasLegend = True
ActiveSheet.ChartObjects("Диаграмма 2").Activate
ActiveChart.Legend.Select
Selection.Delete
ActiveChart.SetElement (msoElementPrimaryCategoryAxis
                        TitleAdjacentToAxis)
ActiveSheet.ChartObjects("Диаграмма 2").Activate
ActiveChart.SetElement (msoElementPrimaryValueAxisTit
                        leHorizontal)
ActiveSheet.ChartObjects("Диаграмма 2").Activate
ActiveChart.Axes(xlValue, xlPrimary).AxisTitle.Text = "Y"
Range("N14").Select
ActiveSheet.ChartObjects("Диаграмма 2").Activate
ActiveChart.PlotArea.Select
ActiveChart.SetElement (msoElementChartTitleAboveChart)
ActiveSheet.ChartObjects("Диаграмма 2").Activate
ActiveChart.ChartTitle.Text = "График функции"
ActiveSheet.ChartObjects("Диаграмма 2").Activate
ActiveChart.ChartArea.Select
ActiveChart.HasAxis(xlCategory) = True
ActiveSheet.ChartObjects("Диаграмма 2").Activate
ActiveChart.Axes(xlCategory).Select
ActiveChart.Axes(xlCategory).TickLabelSpacing = 1
ActiveChart.Axes(xlCategory).TickLabelSpacing = 10
ActiveChart.Axes(xlCategory).CrossesAt = 1
ActiveChart.Axes(xlCategory).CrossesAt = 8
ActiveSheet.ChartObjects("Диаграмма 2").Activate
ActiveChart.PlotArea.Select
End Sub

```

Пример 4. Макрос, который строит гистограмму частот сотни числовых значений случайной величины, равномерно распределённой на отрезке от 0 до 1. Количество интервалов равно 10 (рис. 5.7).

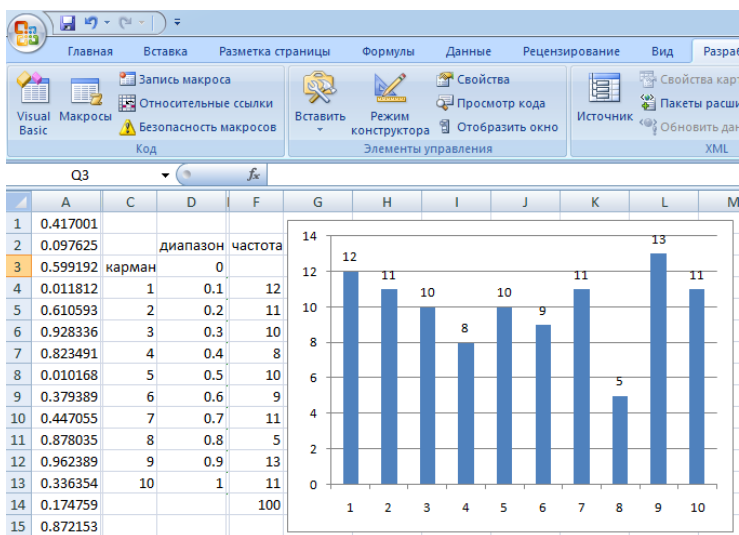


Рис. 5.7. Гистограмма частот случайной величины

Sub построение_гистограммы()

' Сочетание клавиш: Ctrl+д

ActiveCell.FormulaR1C1 = "=RAND()"

Selection.AutoFill

Destination:=Range("A1:A100"), Type:=xlFillDefault

Range("A1:A100").Select

ActiveWindow.SmallScroll Down:=-111

Range("C3").Select

ActiveCell.FormulaR1C1 = "карман"

Range("D3").Select

ActiveCell.FormulaR1C1 = "диапазон"

Range("F3").Select

ActiveCell.FormulaR1C1 = "частота"

Range("C4").Select

```

ActiveCell.FormulaR1C1 = "1"
Range("C5").Select
ActiveCell.FormulaR1C1 = "2"
Range("C4:C5").Select
Selection.AutoFill
Destination:=Range("C4:C13"), Type:=xlFillDefault
Range("C4:C13").Select
Range("D3").Select
Selection.AutoFill
Destination:=Range("D2:D3"), Type:=xlFillDefault
Range("D2:D3").Select
Range("D3").Select
Selection.ClearContents
Range("F3").Select
Selection.AutoFill
Destination:=Range("F2:F3"), Type:=xlFillDefault
Range("F2:F3").Select
Range("F3").Select
Selection.ClearContents
Range("D3").Select
ActiveCell.FormulaR1C1 = "0"
Range("D4").Select
ActiveCell.FormulaR1C1 = "0.1"
Range("D3:D4").Select
Selection.AutoFill
Destination:=Range("D3:D13"), Type:=xlFillDefault
Range("D3:D13").Select
Range("E3").Select
ActiveCell.FormulaR1C1 =
"=FREQUENCY(R[-2]C[-4]:R[97]C[-4],RC[-1]:R[10]C[-1])"
Selection.AutoFill
Destination:=Range("E3:E14"), Type:=xlFillDefault
Range("E3:E14").Select
Range("E3").Select
ActiveCell.FormulaR1C1=
"=FREQUENCY(R1C1:R100C1,RC[-1]:R[10]C[-1])"

```

```

Range("E4:E14").Select
Selection.ClearContents
Range("E3").Select
Selection.AutoFill
Destination:=Range("E3:E14"), Type:=xlFillDefault
Range("E3:E14").Select
Range("F4").Select
ActiveCell.FormulaR1C1 = "=RC[-1]-R[-1]C[-1]"
Range("F4").Select
Selection.AutoFill
Destination:=Range("F4:F13"), Type:=xlFillDefault
Range("F4:F13").Select
Range("F14").Select
ActiveCell.FormulaR1C1 = "=SUM(R[-10]C:R[-1]C)"
Range("G2").Select
ActiveSheet.Shapes.AddChart.Select
Range("J4").Select
ActiveSheet.Shapes.AddChart.Select
ActiveChart.ChartType = xlColumnClustered
ActiveChart.SetSourceData Source:=Range("F4:F13")
ActiveChart.SetElement (msoElementLegendNone)
ActiveChart.ApplyDataLabels
ActiveChart.SeriesCollection(1).DataLabels.Select
Range("S24").Select
ActiveSheet.ChartObjects("Диаграмма 2").Activate
ActiveChart.PlotArea.Select
ActiveSheet.ChartObjects("Диаграмма 2").Activate
ActiveChart.ChartArea.Select
Columns("B:B").ColumnWidth = 0.17
Columns("C:C").ColumnWidth = 6.43
Columns("E:E").ColumnWidth = 0.25
Columns("F:F").ColumnWidth = 6.43
End Sub

```

5.3. Основные структуры языка программирования Visual Basic

В языке программирования Visual Basic каждая переменная должна быть описана в разделе описания переменных, который начинается с зарезервированного слова **Dim**. **Описание переменной** имеет следующую структуру:

<имя переменной> As <тип переменной>

В табл. 5.1 представлены основные простые типы переменных.

Таблица 5.1

Стандартные типы переменных VB

Тип переменной	Объём памяти, байт	Описание
Boolean	2	Логическое, имеет два значения – True, False
Byte	1	Целые числа от 0 до 255
Integer	2	Целое, одинарной точности, (–32768, +32768)
Long Integer	4	Целое, двойной точности, от –2 147 483 648 до +2 147 483 647
Single	4	Вещественное, одинарной точности, $\pm(10^{-45} \dots 3 \cdot 10^{38})$
Double	8	Вещественное, двойной точности, $\pm(5 \cdot 10^{-324} \dots 1,8 \cdot 10^{308})$
String	1 на символ	Символьная переменная переменной длины, 2 млрд символов для динамически изменяемых строк
Currency	8	Денежный тип, используется для финансовых расчётов, $\pm 9 \cdot 10^{14}$
Date	8	Дата и время, от 1/1/100 до 12/31/9999; от 0:0:0 до 23:59:59

Окончание табл. 5.1

Тип переменной	Объём памяти, байт	Описание
Object	4	Экземпляр класса; объект типа OLE
Variant	16 + 1 на символ	Вариантный

Рассмотрим самый простой оператор VB. Это **оператор присваивания**, он имеет следующую структуру:

<имя переменной> = <выражения>

Типы переменной и выражения должны соответствовать, например, вещественной переменной можно будет присвоить значение целого выражения, а наоборот нельзя, компилятор выдаст ошибку несоответствия типов.

Для удобства выполнения различных вычислений в языке программирования Visual Basic имеются встроенные математические функции (табл. 5.2).

Таблица 5.2

Стандартные математические функции VB

Функция	Возвращаемое значение
Abs(<число>)	Модуль (абсолютная величина) числа
Atn(<число>)	Арктангенс
Sin (<число>)	Синус
Cos(<число>)	Косинус
Exp(<число>)	Экспонента
Log(<число>)	Натуральный логарифм
Rnd(<число>)	Случайное число из интервала [0,1)
Randomize	Переустановка генератора случайных чисел
Sgn(<число>)	Знак числа
Sqr (<число>)	Квадратный корень
Fix (<число>)	Целая часть числа, для отрицательного аргумента возвращает ближайшее отрицательное целое число, большее либо равное указанному
Int (<число>)	Целая часть числа, для отрицательного аргумента возвращает ближайшее отрицательное целое число, меньшее либо равное указанному

Ниже мы рассмотрим, как устроены основные операторы языка программирования Visual Basic. Также покажем реализацию некоторых базовых алгоритмов, часто используемых в программировании.

Любое несложное приложение функционирует следующим образом:

- 1) ввод исходных данных;
- 2) программная обработка данных;
- 3) вывод результатов.

У нас это будет организовано по следующей схеме. На листе Excel располагаем элемент управления Кнопка1, преобразование исходных данных в результаты (с помощью соответствующих операторов VB) будет происходить в макросе Кнопка1_Щелчок.

Ввод исходных данных. Исходные данные будут попадать в подпрограмму с помощью стандартного окна ввода, инициируемого вызовом функции **InputBox**, имеющей следующую структуру:

InputBox (<заголовок>, <текст перед полем ввода>)

Вывод результата. Результаты будут выводиться в стандартном окне сообщений при вызове функции **MsgBox**, имеющей следующую структуру:

MsgBox <заголовок>, <пробел>, <текст вывода>

При вызове функция **InputBox** возвращает содержимое своего текстового поля, которое имеет строковый тип (набор символов). Необходимо использовать функцию преобразования строкового типа данных в числовой (табл. 5.3). Аналогично при выводе необходимо преобразовывать числовые данные, наоборот, в строковые, для включения их в текст вывода функции **MsgBox**.

Таблица 5.3

Функции преобразования типов VB

Функция	Возвращаемое значение
Val(<строка>)	Возвращает последовательность символов цифр, как значение числового типа

Функция	Возвращаемое значение
Str(<число>)	Возвращает последовательность символов, которыми записано число
Chr(<число>)	Возвращает символ, ASCII-код которой равен аргументу. Например, Chr (13) – переход на следующую строку
Ord(<символ>)	Возвращает ASCII-код символа

Пример 1. Требуется организовать вычисление по формуле

$$z = \frac{\sin^2(x + y)}{\sqrt{1 + |xy|}}$$

Ясно, что значения переменных x и y надо вводить, а значение переменной z вычислять и выводить. Эти действия выполнит следующий макрос:

```
Sub Кнопка1_Щелчок ()
    Dim x As Single, y As Single, z As Single
    x = Val(InputBox("Введите x", "Ввод x"))
    y = Val(InputBox("Введите y", "Ввод y"))
    z = sin(x+y)^2/sqr(1+abs(x*y))
    MsgBox "Результат z = " & Str(z), , "Результат вычисления"
End Sub
```

Пример 2. Составить макрос, в результате выполнения которого будет выведено значение True, если точка с заданными координатами (x, y) лежит в заштрихованной области на координатной плоскости (рис. 5.8), и False в противном случае.

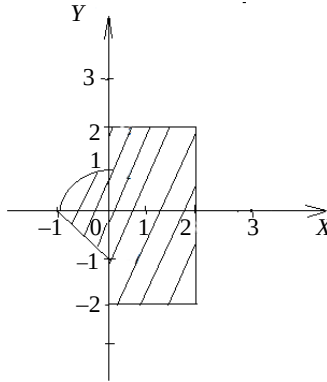


Рис. 5.8. Область на координатной плоскости

Рассматриваемая область состоит из двух частей, каждая из которых описывается системой неравенств.

1-я часть: $x \leq 0$; $x^2 + y^2 \leq 1$; $y \geq -x - 1$.

2-я часть: $x \geq 0$; $x \leq 2$; $y \leq 2$; $y \geq -2$.

Точка с координатами (x, y) лежит в заштрихованной области, если она принадлежит 1-й или 2-й части.

Следующая подпрограмма осуществляет ввод координат точки, вычисление значения логического выражения, которое определяет принадлежность точки области, и выводит полученную логическую величину на экран.

```
Sub Кнопка2_Щелчок ()
```

```
    Dim x As Single, y As Single, L As Boolean
```

```
    x = Val(InputBox("Введите абсциссу", "Ввод  
координат"))
```

```
    y = Val(InputBox("Введите ординату", "Ввод  
координат"))
```

```
    L = (x <= 0) And (x*x + y*y <= 1) And (y >= -x - 1)
```

```
    Or (x >= 0) And (x <= 2) And (y <= 2) And (y >= -2)
```

```
    MsgBox "Лежит ли точка в заданной области?" & Str(L), ,  
        "Результат"
```

```
End Sub
```

В этом случае переменные x и y описаны как вещественные, переменная L – логического типа, она может принимать

только два значения – True и False, в операторе присваивания в правой части вычисляется значение логического выражения с помощью логических операций **And** и **Or**.

Таким образом реализуется линейная алгоритмическая структура для последовательного выполнения действий одного за другим. Операторы в макросе выполняются в том порядке, в котором они записаны.

Более сложная алгоритмическая структура – **разветвление**. Реализуется с помощью оператора **If**, который имеет следующую структуру:

```
If <выражение логического типа> Then  
    <набор операторов 1>  
Else  
    <набор операторов 2>  
End If
```

Условный оператор работает по следующей схеме. Вначале вычисляется значение <выражения логического типа>. Если результат есть True (истина), то выполняется <набор операторов 1>, а <набор операторов 2> не выполняется. Если результат есть False (ложь), наоборот, <набор операторов 1> пропускается, а выполняется <набор операторов 2>.

Условный оператор можно использовать в сокращённой форме: ветвь **Else** может быть опущена, при этом если оператор записан в одну строку, то слово **End If** необходимо убрать.

```
If <выражение логического типа> Then  
    <набор операторов>  
End If
```

или

```
If <выражение логического типа> Then <оператор>
```

При этом, если условное выражение имеет значение True, выполняется <оператор>, стоящий в строке после **Then**; в противном случае весь <оператор> пропускается.

Пример 3. Вычислить значение величины по следующей формуле:

$$Z = \begin{cases} \sqrt{x-1} , & x > 1 \\ x^2 , & x \leq 1 \end{cases}$$

```
Sub Кнопка3_Щелчок()
Dim Z As Single, x As Single
x = Val(InputBox("Введите число", "Ввод числа"))
If x > 1 Then
    Z = Sqr(x-1)
Else
    Z = x^2
End If
MsgBox "Z = " & Str(Z), , " Вывод результата"
End Sub
```

Из условных операторов можно формировать вложенные структуры для организации разветвлений алгоритмов на три и более ветви.

Пример 4. Вычислить значение величины по следующей формуле:

$$y = \begin{cases} 1 - x , & x < 0 \\ 1 + 3x , & 0 \leq x \leq 1 \\ 4 , & x > 1 \end{cases}$$

```
Sub Кнопка4_Щелчок()
Dim y As Single, x As Single
x = Val(InputBox("Введите число", "Ввод числа"))
If x > 1 Then
    y = 4
Else
    If x < 0 Then
        y = 1 - x
    Else
        y = 1 + 3*x
    End If
End If
MsgBox "y = " & Str(y), , " Вывод результата"
```

В VB существует ещё одна возможность разветвления алгоритма на множество ветвей – **оператор выбора Select Case**, который имеет следующую структуру:

```

Select Case <выражение>
    Case <значение 1>
        <набор операторов 1>
    Case <значение 2>
        <набор операторов 2>
    ...
    Case <значение N>
        <набор операторов N>
    Case Else <значение N+1>
        <набор операторов N+1>
End Select

```

Типы <выражения> и <значений> должны соответствовать. Блок **Case Else** может отсутствовать. В блоке **Case** может быть указан набор значений, диапазон значений или условие, как в следующем примере.

Пример 5. Вычислить значение величины по следующей формуле:

$$y = \begin{cases} -x, & x \in [4; 5] \\ x, & x > 7 \\ 0, & x = 2 \text{ или } x = -2 \end{cases}$$

```

Sub Кнопка5_Щелчок()
    Dim x As Single, y As Single
    x = InputBox("Введите число", "Ввод")
    Select Case x
        Case 4 To 5
            y = -x
        Case Is > 7
            y = x
        Case 2, -2
            y = 0
    End Select
    MsgBox "y = " & Str(y), , " Вывод результата"
End Sub

```

Пример 6. В традиционном китайском календаре принят двенадцатилетний цикл [9]. Согласно этому календарю годы в двенадцатилетнем цикле имеют названия животных: крысы, быка, тигра, кролика, дракона, змеи, лошади, козы, обезьяны, петуха, собаки и свиньи. Следующий макрос определяет по введённой дате (кроме январских и февральских дат) название года по китайскому календарю.

```
Sub Кнопка6_Щелчок()  
Dim D As Date,  
    x As Integer,  
    st As String  
' вводим дату в формате "дд"/"мм"/"гггг"  
D = InputBox("Ввод ", "Введите дату")  
'функция DatePart("уууу", D) выделяет из даты год  
x = DatePart("уууу", D)  
Select Case x Mod 12  
Case 0  
st= "Год Обезьяны"  
Case 1  
st="Год Петуха"  
Case 2  
st="Год Собаки"  
Case 3  
st="Год Свиньи"  
Case 4  
st="Год Крысы"  
Case 5  
st= "Год Быка"  
Case 6  
st= "Год Тигра"  
Case 7  
st="Год Кролика"  
Case 8  
st= "Год Дракона"  
Case 9  
st="Год Змеи"
```

```
Case 10
st="Год Лошади"
Case 11
st= "Год Козы"
End Select
MsgBox st, , "Результат"
End Sub
```

Цикл – алгоритмическая структура, реализующая многократное повторение определённых действий (тела цикла). В VB существуют операторы, реализующие цикл с предусловием, цикл с постусловием и цикл с параметром.

Цикл с предусловием проверяет условие до тела цикла. Условия могут быть двух видов: **While** <логическое выражение> или **Until** <логическое выражение>. Различие значений <логического выражения>, при котором происходит повторение тела цикла: **While** проверяет <логическое выражение> на истину, а **Until** – на ложь. Цикл с предусловием может иметь одну из следующих структур:

```
Do While <логическое выражение>
<тело цикла>
Loop
```

или

```
Do Until <логическое выражение>
<тело цикла>
Loop
```

Алгоритм работы оператора цикла с предусловием **While** следующий. Вначале вычисляется значение <логического выражения>. Если <логическое выражение> имеет значение True, выполняется <тело цикла>; после чего вычисление значения <логического выражения> повторяется. Если <логическое выражение> имеет значение False, оператор прекращает свою работу. Таким образом, выход из цикла осуществляется, если <логическое выражение> принимает значение False. Истинность <логического выражения> проверяется вначале каждого

прохождения цикла, поэтому **<тело цикла>** может не выполняться ни разу.

Цикл с постусловием (до тех пор) проверяет условие после выполнения тела цикла и позволяет организовать многократное выполнение операторов, если число повторений заранее неизвестно. Цикл с постусловием может иметь одну из следующих структур:

Do

<тело цикла>

Loop Until <логическое выражение>

или

Do

<тело цикла>

Loop While <логическое выражение>

Оператор цикла с постусловием **Until** работает по следующему алгоритму. Вначале выполняется **<тело цикла>**, после чего вычисляется значение **<логического выражения>**. Если его значение есть **False**, операторы, образующие **<тело цикла>**, повторяются. В противном случае оператор завершает свою работу. То есть выход из цикла осуществляется, если **<логическое выражение>** принимает значение **True** (истина).

Поскольку значение **<логического выражения>** вычисляется в конце каждого прохождения цикла, **<тело цикла>** выполнится хотя бы один раз.

Параметрический цикл предназначен для организации циклов с заранее известным числом повторений тела цикла. Структура оператора параметрического цикла:

For <параметр> = <начальное значение> To <конечное значение> [Step <шаг>]

<тело цикла>

Next <параметр>

<Тело цикла> выполняется столько раз, сколько будет значений у переменной цикла, когда она пробегает от начального значения до конечного с указанным шагом (по умолчанию **<шаг>** равен 1).

Пример 7. Требуется вычислить значения функции $y = x^3$ при значениях аргумента, лежащих в промежутке от 0 до 3 с шагом 0,1.

```
Sub Кнопка7_Щелчок()
Dim x As Single, y As Single, s As String
x = 0
s = " "
Do While x <= 3
y = x ^ 3
s = s & Str(x) & " " & Str(y) & Chr(13)
x = x + 0.1
Loop
End Sub
```

Пример 8. Вычислить величину

$$y = \sum_{i=1}^n \frac{1}{i^2}$$

по заданному значению n .

```
Sub Кнопка8_Щелчок()
Dim n As Integer, i As Integer, x As Single,
y As Single, s As String
Do
x = Val(InputBox("Введите n"))
Loop Until (x = Int(x)) And (x > 0)
n = x
y = 0
For i = 1 To n
y = y + 1 / i ^ 2
Next
MsgBox "y = " & Str(y), , "Результат"
End Sub
```

В первом цикле **Do Loop Until** происходит проверка того, что введённое значение является натуральным. Окно ввода будет появляться до тех пор, пока не будет введено натуральное число.

Пример 9. Требуется сгенерировать сто целых случайных чисел из диапазона от 0 до 50 и вычислить сумму всех полученных значений, которые являются чётными.

```
Sub Кнопка9_Щелчок()  
Dim j As Integer, y As Single, n As Integer  
y = 0  
For j = 1 To 100  
    n = Int(Rnd * 50)  
    If n Mod 2 = 0 Then y = y + n  
Next  
MsgBox "Сумма" & Str(y), , "Результат"  
End Sub
```

Напомним определение из третьей главы данного пособия.

Массив – это сложный (структурированный) тип данных, переменная типа массив (для краткости тоже называется "массив") представляет собой совокупность пронумерованных индексом элементов одного типа.

Описание массива определяет имя, размер массива, базовый тип и производится в разделе описания переменных. В VB структура описания массивов имеет следующий вид:

1. **<имя массива> (<номер последнего элемента>) [As <тип>]**

2. **<имя массива> (<начальный индекс> To <конечный индекс>) [As <тип>]**

Значение **<начального индекса>** массива по умолчанию (базового индекса) можно изменить оператором **Option Base**, иначе его значение равно 0. Например, для изменения базового индекса с 0 на 1 перед процедурой необходимо поместить описание **Option Base 1**.

Заполнение массива в программе производится поэлементно. Чаще всего для этого используется цикл с параметром, где в качестве параметра применяется индексная переменная. Возможно заполнение массива путём простого присвоения значения элементам:

```
Dim M (1 To 3) As Byte  
M (1) = 7
```

$M(2) = 23$

$M(3) = 41$

Двумерный массив – это сложный (структурированный) тип данных, переменная типа массив (для краткости тоже называется "массив") представляет собой совокупность пронумерованных двумя индексами элементов одного типа.

Описание двумерного массива имеет следующую структуру:

1. **<имя массива> (<k1>, <k2>) [As <тип>]**

2. **<имя массива> (<n1> To <k1>, <n2> To <k2>) [As <тип>]**

Например, строки

Dim A (25, 25) As Single

Dim A (0 To 25, 0 To 25) As Single

описывают один и тот же массив.

Пример 10. Следующий макрос осуществляет ввод 20 элементов целочисленного массива M1, формирование строки значений элементов массива и вывод его на экран.

```
Sub Кнопка10_Щелчок()
```

```
Dim M1(1 To 20) As Integer,
```

```
    i As Integer,
```

```
    Stroka As String
```

```
Stroka = " "
```

```
For i = 1 To 20
```

```
    M1(i) = Val(InputBox("Введите " & i & "-й элемент массива", "Заполнение массива"))
```

```
Next i
```

```
For i = 1 To 20
```

```
    Stroka = Stroka & Str(M1(i)) & " "
```

```
Next i
```

```
MsgBox Stroka, , "Результат"
```

```
End Sub
```

Пример 11. Для наперёд заданного натурального числа n требуется сохранить в двумерном массиве и организовать вывод в окне сообщений квадратной матрицы чисел, определённой по формуле

$$\begin{pmatrix} 1 & 2 & 3 & \dots & n-2 & n-1 & n \\ 2 & 1 & 2 & \dots & n-3 & n-2 & n-1 \\ 3 & 2 & 1 & \dots & n-4 & n-3 & n-2 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ n-1 & n-2 & n-3 & \dots & 2 & 1 & 2 \\ n & n-1 & n-2 & \dots & 3 & 2 & 1 \end{pmatrix}$$

```

Sub Кнопка11_Щелчок()
Dim A() As Integer, n As Integer,
    i As Integer, j As Integer,
    Stroka As String
n = InputBox("Введите порядок матрицы ", "размерность")
'переопределение размерности массива
ReDim A(n, n) As Integer
For i = 1 To n
    For j = i To n
        A(i, j) = i - j + 1
        A(j, i) = A(i, j)
    Next j
Next i
For i = 1 To n
    For j = 1 To n
        Stroka = Stroka & Str(A(i, j)) & " "
    Next j
    Stroka = Stroka & Chr(13)
Next i
MsgBox "Matrix: " & Chr(13) & Stroka, ,
    "Результат"
End Sub

```

5.4. Базовые приёмы обработки структурированных данных

Допустим, что в некотором массиве сохраняется набор числовых значений. Можно вычислить сумму или произведение всех числовых значений из массива, определить наименьшее или наибольшее, переставить в порядке возрастания или убывания. Можно определить количество значений элементов массива, удовлетворяющих некоторому условию. Могут возникать задачи, в которых потребуется выполнить комбинацию из этих базовых приёмов.

Пример 1. Вычислить сумму значений элементов целого массива, номера которых кратны 3. Пусть длина массива равна 100, значения элементов массива – случайные целые числа.

```
Sub Кнопка1_Щелчок()  
    Dim A(1 To 100) As Integer, j As Integer, y As Integer, n As Integer  
    'Заполняем массив целыми случайными числами  
    For j = 1 To 100  
        A(j) = Int(Rnd * 50)  
    Next  
    'Вычисляем сумму  
    y = 0  
    For j = 1 To 100  
        If j Mod 3 = 0 Then y = y + A(j)  
    Next  
    MsgBox "Сумма =" & Str(y), , "Результат"  
End Sub
```

Пример 2. Определить наибольшее значение среди значений элементов массива случайных целых чисел из предыдущего примера.

```
Sub Кнопка2_Щелчок()  
    Dim A(1 To 100) As Integer, j As Integer, y As Integer, n As Integer  
    'Заполняем массив целыми случайными числами
```

```

For j = 1 To 100
A(j) = Int(Rnd * 50)
Next j
'Определяем наибольшее значение
y = -30000
For j = 1 To 100
If A(j) > y Then y = A(j)
Next j
MsgBox "Наибольшее значение =" & Str(y), , "Результат"
End Sub

```

Пример 3. Определить наименьшее значение и его номер среди значений элементов массива случайных целых чисел из предыдущего примера.

```

Sub Кнопка3_Щелчок()
Dim A(1 To 100) As Integer, j As Integer, y As Integer,
Ny As Integer
'Заполняем массив целыми случайными числами
For j = 1 To 100
A(j) = Int(Rnd * 50)
Next
'Определяем наименьшее значение и его номер
y = 30000
For j = 1 To 100
If A(j) < y Then
y = A(j)
Ny = j
End If
Next j
MsgBox "Наименьшее значение =" & Str(y) & Chr(13) &
"его номер =" & Str(Ny), , "Результат"
End Sub

```

Пример 4. Вычислить произведение значений элементов целого массива из предыдущего примера, значения которых не превышают 5.

```

Sub Кнопка4_Щелчок()
Dim A(1 To 100) As Integer, j As Integer,
    y As Integer, n As Integer
'Заполняем массив целыми случайными числами
For j = 1 To 100
    A(j) = Int(Rnd * 50 + 1)
Next j
'Вычисляем произведение
y = 1
For j = 1 To 100
    If A(j) <= 5 Then y = y * A(j)
Next j
MsgBox "Произведение =" & Str(y), , "Результат"
End Sub

```

Пример 5. Определить количество значений элементов двумерного целого массива 10×10 , заполненного случайными числами, сумма цифр которых нечётна.

```

Sub Кнопка5_Щелчок()
Dim A(1 To 10, 1 To 10) As Integer, sum As Integer,
    kol As Integer, i As Integer, j As Integer, Stroka As String
'заполнение массива случайными значениями
For i = 1 To 10
    For j = 1 To 10
        A(i, j) = Int(99 * Rnd)
    Next j
Next i
Stroka = " "
For i = 1 To 10
    For j = 1 To 10
        Stroka = Stroka & Str(A(i, j)) & " "
    Next j
    Stroka = Stroka & Chr(13)
Next i
'определение количества
kol = 0

```

```

For i = 1 To 10
    For j = 1 To 10
        d = Int(A(i, j)/10)
        ed = A(i, j) - 10 * d
        sum = d + ed
        If sum mod 2 = 1 Then kol = kol + 1
    Next j
Next i
MsgBox "Массив: " & Chr(13) & Stroka & Chr(13)
      "Количество = "& Str(kol), , "Результат"
End Sub

```

Пример 6. В матрице 5×5 необходимо определить столбцы, имеющие наибольшую и наименьшую суммы элементов и поменять их местами. Для удобства матрицу будем сохранять в двумерном массиве, первоначально заполним его целыми случайными числами.

```

Sub Кнопка6_Щелчок()
Dim A(1 To 5, 1 To 5) As Integer, sum(1 To 5) As Integer,
    summin As Integer, summax As Integer, Nmin As Integer,
    Nmax As Integer, b As Integer, i As Integer, j As Integer,
    StrokaA As String, StrokaB As String
'заполнение массива случайными значениями
For i = 1 To 5
    For j = 1 To 5
        A(i, j) = Int(99 * Rnd)
    Next j
Next i
'вывод исходной матрицы
StrokaA = " "
For i = 1 To 5
    For j = 1 To 5
        StrokaA = StrokaA & Str(A(i, j)) & " "
    Next j
    StrokaA = StrokaA & Chr(13)
Next i
'формирование массива сумм элементов столбцов

```

```

For j = 1 To 5
    sum(j) = 0
    For i = 1 To 5
        sum(j) = sum(j) + A(i, j)
    Next i
Next j
'номер столбца с наибольшей суммой
summax = -10000
For j = 1 To 5
    If summax < sum(j) Then
        summax = sum(j)
        Nmax = j
    End If
Next j
'номер столбца с наименьшей суммой
summin = 10000
For j = 1 To 5
    If summin > sum(j) Then
        summin = sum(j)
        Nmin = j
    End If
Next j
'перестановка столбцов
For i = 1 To 5
    b = A(i, Nmax)
    A(i, Nmax) = A(i, Nmin)
    A(i, Nmin) = b
Next i
'вывод матрицы с переставленными столбцами
StrokaB = " "
For i = 1 To 5
    For j = 1 To 5
        StrokaB = StrokaB & Str(A(i, j)) & " "
    Next j
    StrokaB = StrokaB & Chr(13)
Next i

```

```

MsgBox "До: " & Chr(13) & StrokaA & Chr(13) &
      "После: " & Chr(13) & StrokaB, , "Результат"
End Sub

```

Пример 7. Отсортировать значения элементов целочисленного массива по убыванию методом пузырька. Алгоритм метода изложен в главе 3 данного пособия.

```

Sub Кнопка7_Щелчок()
    Dim M() As Integer, n As Integer, i As Integer, j As Integer, P
    As Integer, Stroka1 As String, Stroka2 As String
    n = Val(InputBox("Введите число элементов массива ", "Определение размера"))
    ReDim M(1 To n) As Integer
    Str1 = " "
    Str2 = " "
    'Заполнение массива с клавиатуры и формирование строки
    For i = 1 To n
        M(i) = Val(InputBox("Введите " & i & "-й элемент массива", "Заполнение массива"))
        Stroka1 = Stroka1 & Str(M(i)) & " "
    Next i
    'Вложенные циклы, организующие сортировку массива
    For i = 1 To n - 1
        For j = 1 To n - i
            If M(j) < M(j + 1) Then
                P = M(j)
                M(j) = M(j + 1)
                M(j + 1) = P
            End If
        Next j
    Next i
    'Вывод отсортированного массива
    For i = 1 To n
        Stroka2 = Stroka2 & Str(M(i)) & " "
    Next i
    MsgBox "Исходный массив: " & Stroka1 & Chr(13) &
    "Отсортированный массив:" & Stroka2, , "Результат"
End Sub

```

Пример 8. Отсортировать значения элементов целочисленного массива по возрастанию методом выбора. Алгоритм метода изложен в главе 3 данного пособия.

```
Sub Кнопка8_Щелчок()  
Dim M() As Integer, n As Integer, i As Integer, j As Integer,  
    Nmin As Integer, Stroka1 As String,  
    Stroka2 As String,  
    min As Integer  
n = Val(TextBox("Введите число элементов массива ",  
    "Определение размера"))  
ReDim M(1 To n) As Integer  
Str1 = " "  
Str2 = " "  
'Заполнение массива случайными числами  
For i = 1 To n  
    M(i) = Int(Rnd*1000)  
Next i  
'Вывод исходного массива  
For i = 1 To n  
    Stroka1 = Stroka1 & Str(M(i)) & " "  
Next i  
'Вложенные циклы, организующие сортировку массива  
For i = 1 To n - 1  
    min = 10000  
    For j = i To n  
        If min > M(j) Then  
            min = M(j)  
            Nmin=j  
        End If  
    Next j  
    M(Nmin) =M(i)  
    M(i) = min  
Next i  
'Вывод отсортированного массива  
For i = 1 To n
```

```

    Stroka2 = Stroka2 & Str(M(i)) & " "
Next i
MsgBox "Исходный массив: " & Stroka1 & Chr(13) &
"Отсортированный массив:" & Stroka2, , "Результат"
End Sub

```

5.5. Программирование форм для работы с книгой Excel

Табличный процессор Excel – мощнейшее средство хранения и обработки данных. В ячейках листа книги Excel можно размещать данные различных типов и форматов и всевозможные функции для обработки этих данных. Но механизм функций в ячейках недостаточно гибок для проведения не совсем стандартных манипуляций с данными. Этого недостатка лишено программирование в VBA, позволяющее организовать преобразование данных по любому алгоритму. Итак, за хранение и отображение данных на экране отвечает лист Excel, а за обработку – VBA.

Данные из ячеек листа попадают в программу и обратно с помощью функции **Cells**, которая устроена следующим образом: Cells(i, j) – это содержимое ячейки, расположенной в i-й строке и j-м столбце листа.

Пример 1. В верхнем левом углу листа Excel в ячейках записана матрица целых чисел размером 3×3. Необходимо считать значения элементов матрицы в двумерный массив, затем в другой массив поместить те же значения, но в "транспонированном виде", после из этого массива поместить значения в виде матрицы под исходными ячейками. С этой целью в макрос необходимо поместить следующие операторы:

```

Sub Кнопка1_Щелчок()
Dim A(3, 3) As Integer, B(3, 3) As Integer,
As Integer, j As Integer
'считывание значений из ячеек листа таблицы в массив
For i = 1 To 3
    For j = 1 To 3

```

```

        A(i, j) = Cells(i, j)
    Next j
Next i
'формирование нового массива
'запись значений из нового массива в ячейки листа
' электронной таблицы
For i = 1 To 3
    For j = 1 To 3
        B(i, j) = A(j, i)
        Cells(i + 5, j) = B(i, j)
    Next j
Next i
End Sub

```

Элементы управления можно располагать на листе, данные вводить с помощью стандартного окна **InputBox**, выводить на экран стандартным окном **MsgBox**. Этот механизм позволил изучить операторы и некоторые приёмы программирования. Ещё раз отметим, что это можно делать и без VBA, но инструменты полноценного программирования реализуют наиболее гибкий подход, позволяя эффективно обрабатывать данные по алгоритмам любой сложности.

С электронной таблицей удобно работать через форму, этот инструмент есть и в стандартном арсенале Excel. В следующем примере покажем, как запрограммировать процесс работы с таблицей через несколько связанных между собой пользовательских форм.

Пример 2. Необходимо организовать ввод, хранение и просмотр оценок, полученных студентами за экзамены, средствами электронной таблицы Excel.

Сначала обсудим логику работы. Манипуляции с данными будем производить через три формы. Первая – главная, на ней две кнопки, каждая из которых вызывает свою форму. Работа с таблицей предполагает выполнение двух действий: добавление записи в таблицу и просмотр определённой записи. Для каждого действия – своя форма.

Сначала заготовим пустую таблицу (рис. 5.9). Это не обязательно, ведь для работы с таблицей нам не понадобится на неё смотреть, но поможет для отладки (убедимся, что она заполняется корректно). В ячейках, в левом верхнем углу, поставим два нуля, в них будет храниться текущая информация об общем количестве записей и количестве отличников по результатам сессии.

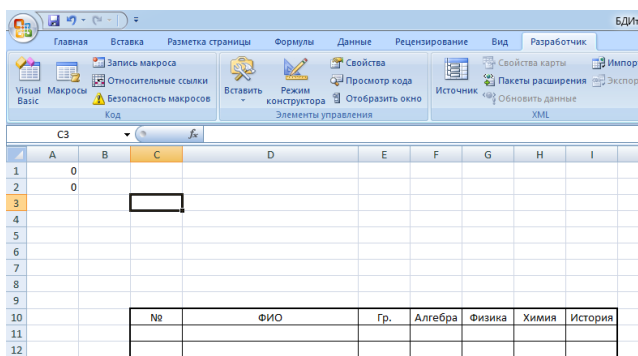


Рис. 5.9. Заготовка таблицы

Открываем вкладку **Разработчик** и жмём кнопку **Visual Basic**. Открылось окно интегрированной среды разработки. Нажимаем кнопку **Insert UserForm** (вставить форму), потянув за нижний правый угол заготовки формы, настраиваем её размер. Напомним, что при первом сохранении проекта необходимо задать тип файла **Книга Excel с поддержкой макросов**, не следует забывать и периодически сохранять файл.

Помещаем на форму необходимые стандартные элементы управления (рис. 5.10), для этого щёлкаем на кнопке соответствующего элемента на панели **Toolbox** (вкладка **Controls**) и протягиваем мышью в нужном месте формы до нужного размера (размер и местоположение можно будет поменять в любой момент). Получаем две командные кнопки и фрейм (панель с рамкой) с четырьмя текстовыми метками. Чтобы метки были на фрейме, необходимо сначала поместить фрейм на форму, а затем, выделив его (нажав на нём мышью), помещать на него мет-

ки. Тогда метки будут наследовать некоторые свойства фрейма, например **Visible** (видимость на экране). Фрейм управляет своим содержимым, а красивая рамка логично объединяет элементы управления. Надписи, которые располагаются на элементах управления, например заголовков формы, задаются свойством **Caption**. Выделяем элемент и в окне **Properties** (свойства) в строке **Caption** меняем исходную стандартную надпись. Заметим, что сразу после вставки элемента надпись на нём совпадает с его именем, например **Label1**, свойство **Caption** мы поменяли на "Всего сдавших", а имя метки так и осталось **Label1**. Конечно, в серьёзных больших приложениях стандартные имена элементов сразу меняют на имена, отражающие смысл использования элементов, но в нашем примере проще не менять имена элементов, их не много.

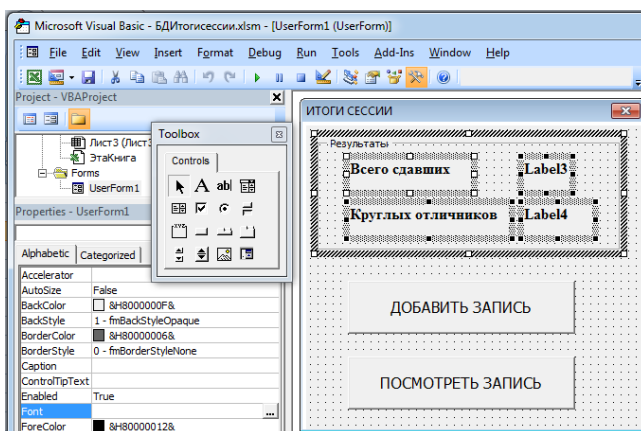


Рис. 5.10. Проектирование главной формы проекта

Вставляем вторую форму, помещаем в неё фрейм, во фрейм помещаем шесть меток, шесть окошек **TextBox** и командную кнопку, внизу формы располагаем ещё одну кнопку. Меняем надписи на элементах в соответствии с логикой приложения, также настраиваем размеры элементов и выравниваем их расположение, как на рис. 5.11.

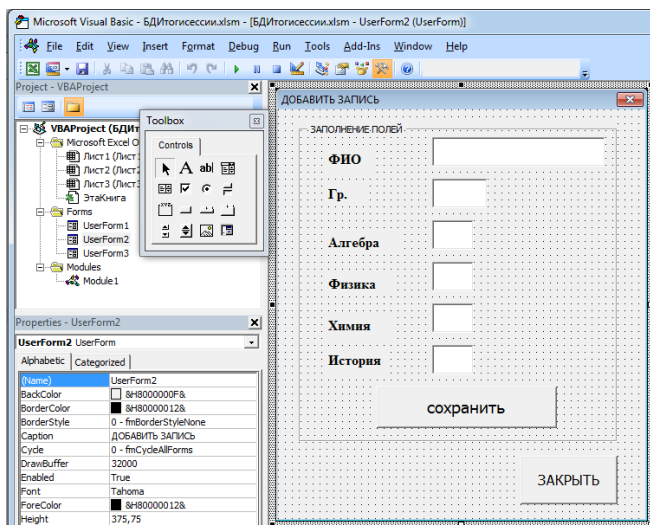


Рис. 5.11. Вторая форма в режиме конструктора

Шрифт надписи на элементе управления определяется его свойством **Font**. Чтобы его изменить с предустановленного на необходимый для нашего проекта, в соответствующей строке окна свойств элемента нажимаем на кнопку и выбираем шрифт в стандартном окне (рис. 5.12).

Вставляем третью форму (рис. 5.13). В ней фрейм с меткой, окном **ComboBox**, ещё одним фреймом и командной кнопкой. Внутри второго фрейма восемь меток. Отдельно внизу формы ещё одна кнопка. Выравниваем элементы и задаём им новые надписи (рис. 5.14). У второго фрейма свойству **Visible** установим в окне свойств значение **True**.

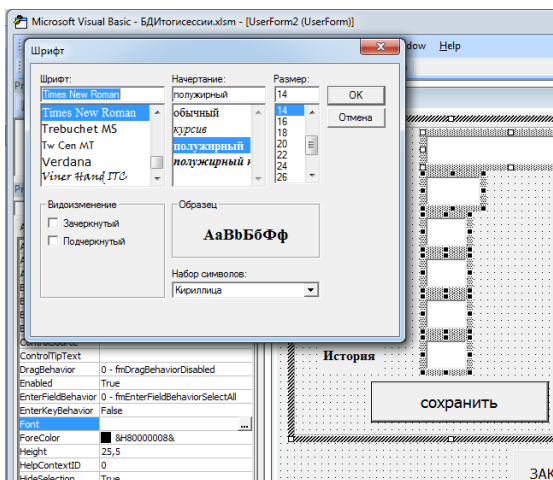


Рис. 5.12. Выбор шрифта для надписи

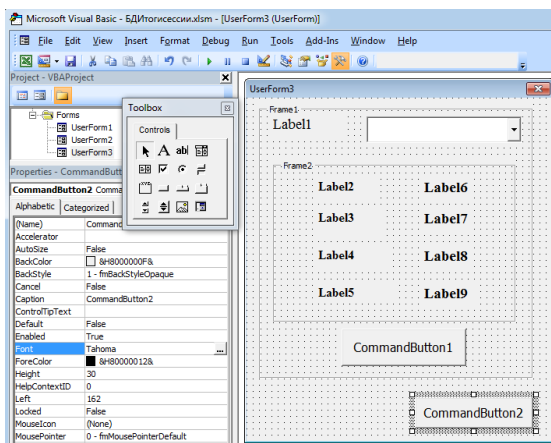


Рис. 5.13. Элементы управления третьей формы проекта

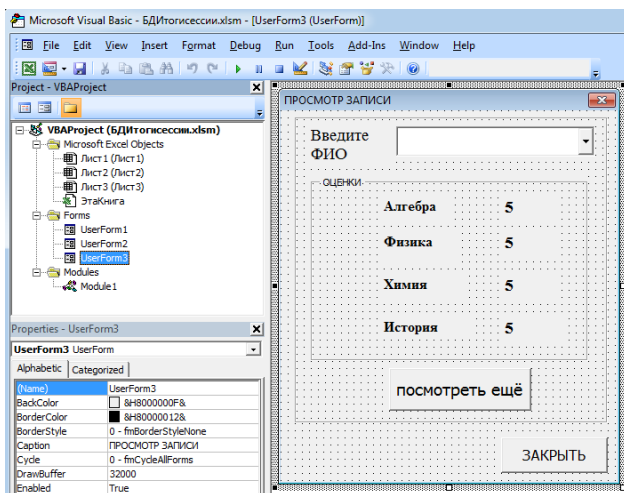


Рис. 5.14. Итоговый вид третьей формы

Поместим на лист с таблицей кнопку с надписью "Работать с таблицей" и свяжем с ней следующую подпрограмму:

```
Sub Кнопка1_Щелчок()
    UserForm1.Show
End Sub
```

Данная кнопка будет вызывать главную форму, когда мы захотим добавить или посмотреть записи в таблице.

После того, как формы спроектированы в режиме конструктора, пришла пора заняться программированием. Следующие подпрограммы определяют функционал первой формы:

' При нажатии кнопки "Добавить запись"

' открывается вторая форма

```
Private Sub CommandButton1_Click()
    UserForm2.Show
End Sub
```

' При нажатии кнопки "Посмотреть запись"

' открывается третья форма

```
Private Sub CommandButton2_Click()
    UserForm3.Show
End Sub
```

' Когда первая форма становится активной, на ней
' обновляются краткие итоги

```
Private Sub UserForm_Activate()  
    Label3.Caption = Str(Cells(1, 1))  
    Label4.Caption = Str(Cells(2, 1))  
End Sub
```

У второй формы кроме подпрограмм есть ещё и переменные:

```
Dim ks As Integer, kot As Integer,  
    alg As Byte, fiz As Byte,  
    him As Byte, ist As Byte,  
    fio As String, ng As Integer
```

' При нажатии кнопки "Сохранить" произойдёт добавление
' записи с данными из текстовых окошек в таблицу

```
Private Sub CommandButton1_Click()  
    fio = TextBox1.Text  
    ng = Val(TextBox2.Text)  
    alg = Val(TextBox3.Text)  
    fiz = Val(TextBox4.Text)  
    him = Val(TextBox5.Text)  
    ist = Val(TextBox6.Text)  
    ks = Cells(1, 1)  
    kot = Cells(2, 1)  
    If (alg = 5) And (fiz = 5) And (him = 5)  
        And (ist = 5) Then kot = kot + 1  
    ks = ks + 1  
    Cells(ks + 10, 3) = ks  
    Cells(ks + 10, 4) = fio  
    Cells(ks + 10, 5) = ng  
    Cells(ks + 10, 6) = alg  
    Cells(ks + 10, 7) = fiz  
    Cells(ks + 10, 8) = him  
    Cells(ks + 10, 9) = ist  
    TextBox1.Text = ""  
    TextBox3.Text = ""  
    TextBox4.Text = ""
```

```

        TextBox5.Text = ""
        TextBox6.Text = ""
        Cells(1, 1) = ks
        Cells(2, 1) = kot
        TextBox1.SetFocus
    End Sub
' При нажатии кнопки "Заккрыть"
' закроется вторая форма
    Private Sub CommandButton2_Click()
        UserForm2.Hide
    End Sub

```

Подпрограммы и переменные третьей формы:

```

Dim fio As String, i As Integer, nf As Integer, ks As Integer
' При нажатии кнопки "Посмотреть ещё"
' очищается окно элемента ComboBox1
    Private Sub CommandButton1_Click()
        ComboBox1.Text = ""
        ComboBox1.SetFocus
        Frame2.Visible = False
    End Sub
' Событие CommandButton1_Enter произойдёт, когда
' кнопка "Посмотреть ещё" получит фокус управления
    Private Sub CommandButton1_Enter()
        fio = ComboBox1.Text
        ks = Cells(1, 1)
        For i = 11 To ks+10
            If fio = Cells(i, 4) Then nf = i
        Next
        Label6.Caption = Str(Cells(nf, 6))
        Label7.Caption = Str(Cells(nf, 7))
        Label8.Caption = Str(Cells(nf, 8))
        Label9.Caption = Str(Cells(nf, 9))
        Frame2.Visible = True
    End Sub
' При нажатии кнопки "Заккрыть"

```

```

' закроется третья форма
Private Sub CommandButton2_Click()
    UserForm3.Hide
End Sub

' При открытии третьей формы выпадающий список
' элемента ComboBox1 заполнится фамилиями из таблицы
' для автоматического заполнения текстового поля
' (рис. 5.15)
Private Sub UserForm_Initialize()
    ks = Cells(1, 1)
    For i = 11 To ks + 10
        fio = Cells(i, 4)
        ComboBox1.AddItem (fio)
    Next
End Sub

```

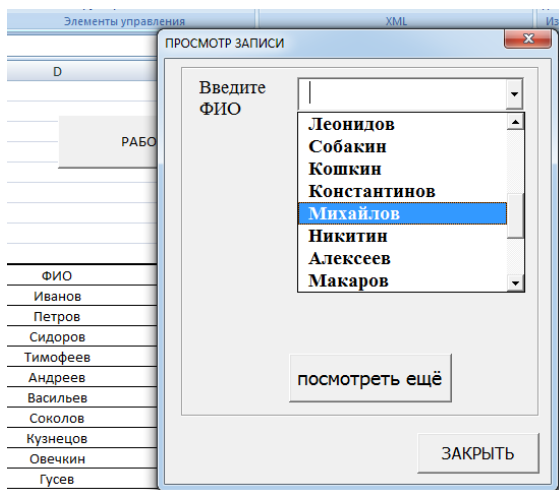


Рис. 5.15. Список элемента ComboBox1

В результате у нас получился проект VBA, структура которого представлена на рис. 5.16.

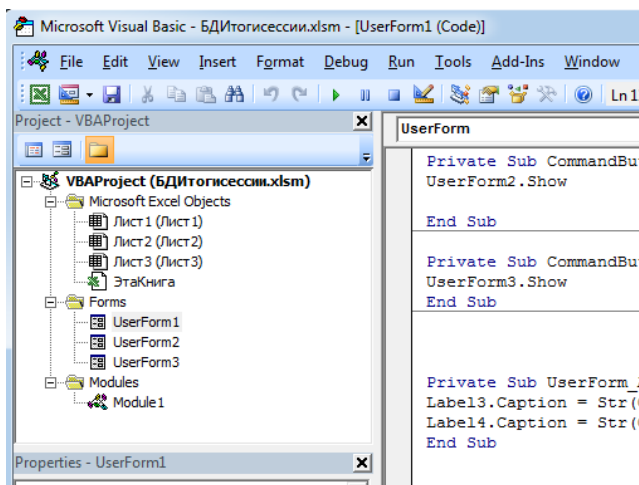


Рис. 5.16. Структура проекта

В книгу Excel добавлена программная надстройка: три формы и один модуль. При работе с формой возможны два состояния главного окна интегрированной среды разработки: **View Code** или **View Object**, соответственно в окне отображается программный код формы или сама форма в режиме конструктора. Элементы управления, которые расположены на форме, – это локальные объекты формы (поэтому их стандартные имена в каждой форме одинаковы, как нумерация квартир в разных домах), подпрограммы тоже локальные. Путаницы не происходит: в каждый момент работы приложения активной может быть только одна форма – её подпрограммы и будут выполняться. Всё это следует из объектно-ориентированной модели формы.

Теперь расскажем, как устроен модуль. В нём содержатся подпрограммы, являющиеся событиями для элементов управления, расположенных непосредственно на листе книги Excel. Сам лист служит формой (объектом, на котором можно располагать элементы управления). Таким образом, режим конструктора, в котором визуально задаются свойства элементов (задать размер, положение и подписать), включается прямо на листе. Для работы с подпрограммами открывается окно **View Code** интегрированной среды разработки. Заметим, что все имена элементов управления листа и событий в модулях русифицированы. В формах все имена англоязычные.

6. РЕШЕНИЕ ЗАДАЧ НА КОМПЬЮТЕРЕ

Решение задач на компьютере состоит из следующих этапов:

1. Создание математической модели задачи.
2. Разработка алгоритма решения задачи.
3. Набор программного кода.
4. Запуск и отладка программы.
5. Поиск и исправление ошибок в программе.
6. Эксплуатация и поддержка программного продукта.

Ниже мы рассмотрим несколько математических задач, которые очень часто приходится решать в научно-производственной сфере.

6.1. Решение нелинейных алгебраических уравнений

Пусть дано уравнение $F(x) = 0$. Решить уравнение – это значит найти все корни уравнения. Корнем уравнения называется такое значение x^* , при подстановке которого в уравнение оно превращается в тождество: $F(x^*) \equiv 0$. Точные методы решения уравнений (по формулам) существуют, если $F(x)$ является полиномом первой или второй степени относительно x . В остальных случаях необходимо применять приближённые методы. Слово "приближённые" означает, что ищется не точное решение x^* , а некое значение $x_{\text{прибл.}}$, достаточно близкое к x^* . Абсолютная величина разности между точным и приближённым решениями $|x_{\text{прибл.}} - x^*|$ называется погрешностью приближённого решения. Говорят, что $x_{\text{прибл.}}$ является решением уравнения $F(x) = 0$ с точностью (или погрешностью) ε , если выполняется неравенство $|x_{\text{прибл.}} - x^*| \leq \varepsilon$.

Для нахождения приближённых значений корней уравнений применяют **итерационные методы**. Смысл этих методов заключается в построении итерационной последовательности $x_0, x_1, x_2, \dots, x_n, \dots$, которая стремится к неизвестному точному решению x^* [8].

При приближённом решении заранее определяют точность – задают значение ε , после этого находят начальное при-

ближение x_0 , с помощью него вычисляют первую итерацию – x_1 , с помощью которой вычисляют вторую итерацию – x_2 . В этом и состоит итерационный процесс: последующая итерация вычисляется через предыдущую. Допустим, что $(n - 1)$ -я итерация не удовлетворяет требуемой точности: $|x_{n-1} - x^*| > \varepsilon$, а следующая n -я удовлетворяет: $|x_n - x^*| \leq \varepsilon$, тогда x_n будет приближённым решением уравнения с точностью ε . Таким образом на каждой итерации уменьшается погрешность.

Скорость сходимости – это характеристика итерационного метода, она тем выше, чем меньше итераций требуется для достижения определённой точности. **Трудоёмкость** метода определяется количеством арифметических операций, требующихся для вычисления одной итерации.

Интервал изоляции корня уравнения – это числовой промежуток, внутри которого находится ровно один корень уравнения.

Начальное приближение (нулевая итерация, оно необходимо для начала итерационного процесса) – это любое значение из интервала изоляции.

Итак, чтобы решить уравнение $F(x) = 0$, необходимо:

1. Определить, сколько у данного уравнения существует корней.
2. Для каждого корня определить интервал изоляции.
3. Для каждого корня определить начальное приближение.
4. Для каждого корня выбрать значение погрешности.
5. Для каждого корня запустить итерационный процесс до достижения приближённым решением заданной погрешности.

Первый этап можно выполнить разными способами – графическим, табличным или аналитическим. Графический заключается в построении графика функции $y = F(x)$, абсциссы точек пересечения графика с осью абсцисс – это значения корней. По графику их точные значения узнать нельзя, но границы интервалов изоляции можно.

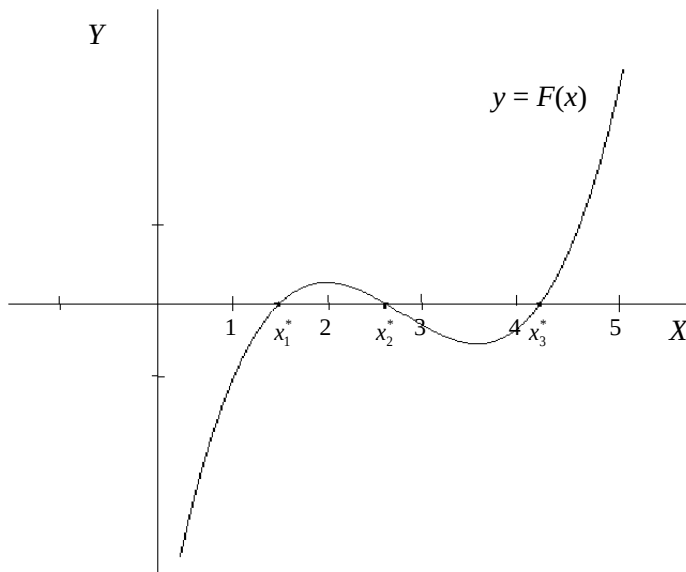


Рис. 6.1. Определение интервалов изоляции

Пример. Пусть $F(x)$ – полином третьей степени и график функции $y = F(x)$ такой, как на рис. 6.1. Понятно, что график полинома третьей степени не может пересекать ось абсцисс больше чем в трёх точках, следовательно, у уравнения $F(x) = 0$ три корня. Разметка оси позволяет определить интервалы изоляции: $[1; 2]$, $[2; 3]$ и $[4; 5]$.

Самый простой итерационный метод решения уравнений – это метод дихотомии или деления отрезка пополам. Интервал изоляции делится пополам, на два числовых промежутка, корень остаётся внутри одного из них. Этот отрезок будем делить пополам на следующем шаге и т.д. Можно сказать, что граница интервала изоляции – это приближённое значение корня с погрешностью не больше длины интервала изоляции. За один шаг длина интервала уменьшается в 2 раза; чтобы погрешность уменьшилась в 1000 раз, необходимо выполнить 10 делений пополам. Скорость сходимости у данного метода не высокая, зато трудоёмкость низкая: чтобы произвести итерацию, нужно вы-

числить среднее арифметическое двух чисел и одно значение полинома третьей степени.

6.2. Решение систем линейных алгебраических уравнений

Многие виды математических задач сводятся к решению систем линейных алгебраических уравнений (СЛАУ).

Вспомним линейную алгебру. Общий вид СЛАУ:

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \dots + a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + \dots + a_{2n}x_n = b_2 \\ \vdots \\ a_{m1}x_1 + a_{m2}x_2 + \dots + a_{mn}x_n = b_m \end{cases} \quad (6.1)$$

СЛАУ можно записать в матричной форме:

$$\begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{pmatrix}. \quad (6.2)$$

В сокращённом виде

$$Ax = b, \quad (6.3)$$

где A – матрица коэффициентов системы, x – вектор неизвестных, b – столбец правых частей уравнений.

Решить СЛАУ – значит определить такие значения неизвестных, подстановка которых в систему превратит каждое уравнение в тождество.

В общем случае СЛАУ может иметь единственное решение, может иметь множество решений, а может решений не иметь. Системы, которые имеют хотя бы одно решение, называются совместными. Необходимое и достаточное условие совместности системы устанавливает теорема Кронекера – Капелли [7]. Системы, не имеющие ни одного решения, называют несовместными.

Простой случай $m = n$, тогда матрица СЛАУ квадратная и количество переменных совпадает с количеством неизвестных:

$$A = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{pmatrix}. \quad (6.4)$$

Для того, чтобы у такой системы было единственное решение, определитель матрицы A должен быть не равен нулю:

$$\|A\| \neq 0. \quad (6.5)$$

В линейной алгебре такие СЛАУ решают методом Крамера [7]. Это точный математический метод, значения неизвестных величин определяются в результате вычислений по формулам в предположении абсолютно точного выполнения арифметических операций над числами. Но в данном случае не учитывается время выполнения арифметических операций. Получается, что с точки зрения математики точное единственное решение системы уравнений (6.1) при условиях (6.4) и (6.5) существует, но на практике его поиск не всегда завершается успехом. Действительно, для реализации метода Крамера необходимо вычислять определитель матрицы A , что составляет порядка $n!$ арифметических действий. При $n = 1000$ это уже не представляется возможным, хотя СЛАУ совсем не большого размера. Поэтому для решения СЛАУ применяют итерационные методы вычисления приближённых значений неизвестных с заданной точностью.

Парадоксальный факт — оказывается, компьютер может ошибаться! В том смысле, что абсолютная точность выполнения арифметических операций на компьютере не всегда соблюдается. Именно при решении СЛАУ достаточно больших размеров обнаружился эффект накопления вычислительных погрешностей, связанных с машинным округлением значений в разрядах арифметического устройства процессора ЭВМ и ячейках памяти. Например, если перемножить два числа, у которых после запятой (в дробной части) находится 10 десятых цифр, то у произведения будет уже 20 цифр после запятой, из которых будут отброшены последние 10 при помещении в ячейку, предусматривающую хранение только 10 цифр после запятой. С этой проблемой не справляется увеличение памяти для хранения числовых значе-

ний: при умножении количество знаков после запятой всегда удваивается.

На скорость накопления погрешностей влияет число обусловленности матрицы A , равное модулю отношения наибольшего по абсолютной величине собственного значения матрицы A к наименьшему [7].

Следующий пример иллюстрирует вычислительную неустойчивость определителя квадратной матрицы, которая состоит в том, что у двух очень похожих матриц определители могут очень сильно отличаться.

Пример. Квадратная матрица A состоит из 2000 строк и 2000 столбцов. Все элементы матрицы A , за исключением расположенных непосредственно над главной диагональю, равны нулю, а каждый из ненулевых равен 10.

Матрица B отличается от матрицы A значением всего лишь одного элемента в левом нижнем углу: $B_{2000,1} = 10^{-1000}$.

$$A = \begin{pmatrix} 0 & 10 & 0 & \dots & 0 & 0 \\ 0 & 0 & 10 & \dots & 0 & 0 \\ 0 & 0 & 0 & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & 0 & 10 \\ 0 & 0 & 0 & \dots & 0 & 0 \end{pmatrix} \quad (6.6)$$

$$B = \begin{pmatrix} 0 & 10 & 0 & \dots & 0 & 0 \\ 0 & 0 & 10 & \dots & 0 & 0 \\ 0 & 0 & 0 & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & 0 & 0 \\ 10^{-1000} & 0 & 0 & \dots & 0 & 0 \end{pmatrix} \quad (6.7)$$

Несложно увидеть, что определители матриц A и B совершенно различны, несмотря на то что разница между матрицами ничтожна: $\det(A) = 0$, $\det(B) = 10^{999}$.

6.3. Интерполяция

Существует множество математических задач, у которых решение представляется в виде функции. Например, задача Коши для обыкновенного дифференциального уравнения, решением которой будет функция, удовлетворяющая этому уравнению и начальным условиям. Или краевая задача, когда функция должна удовлетворять краевым условиям.

Допустим, требуется определить форму прогиба упругой балки под действием нагрузки (рис. 6.2).

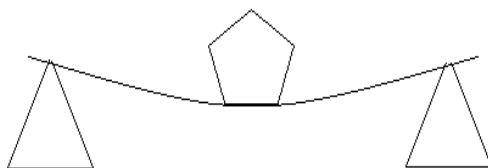


Рис. 6.2. Прогиб балки

Форма балки может быть описана в виде функции одной переменной, её график приведён на рис. 6.3.

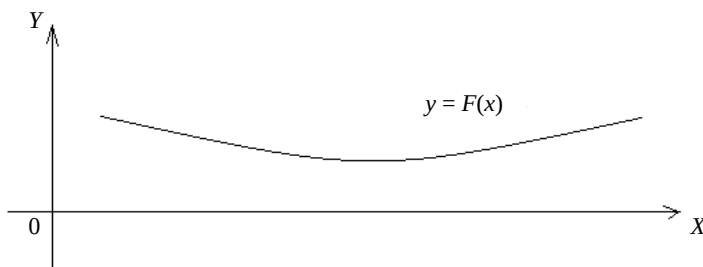


Рис. 6.3. График функции

Функции двух переменных могут описывать форму поверхности тел (рис. 6.4). Актуальной задачей в различных научно-технических областях является расчёт формы объекта исходя из заданных характеристик функционала объекта. Например, какой

формы должен быть обтекатель носовой части подводной лодки, чтобы сопротивление воды и возникающий шум при движении под водой были минимальны. Интересно, что решение этой задачи представляет собой форму, близкую к сферической.

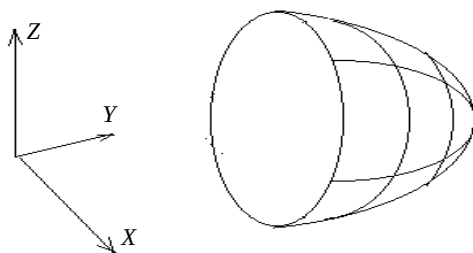


Рис. 6.4. График поверхности

Изучая высшую математику, мы привыкли иметь дело с функциями, заданными аналитически. Что такое функция? Функция – это соответствие элементов одного множества элементам другого, при котором каждому элементу первого множества соответствует не более одного элемента второго множества. Например, непрерывная функция $y = \sin x$ каждому действительному числу ставит в соответствие значение синуса этого числа. Точки плоскости, соответствующие всем таким парам значений, которых бесконечно много, образуют график функции $y = \sin x$.

Теперь представим, что на координатной плоскости провели произвольную линию (рис. 6.5). Эта линия является графиком неизвестной непрерывной функции и тоже состоит из бесконечного множества точек. Эту неизвестную функцию невозможно представить аналитически: чтобы точно её описать, нужно указать все пары координат точек её графика, что физически невозможно.

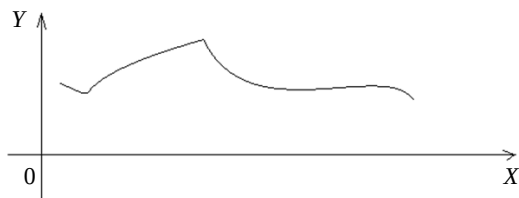


Рис. 6.5. График неизвестной непрерывной функции

Возможный путь – указать конечное количество пар (чем их больше, тем точнее представление неизвестной функции). Таким образом, в реальном мире функция – это совокупность пар числовых значений (рис. 6.6), причём первые значения во всех парах различные, а вторые произвольные (могут и повторяться).

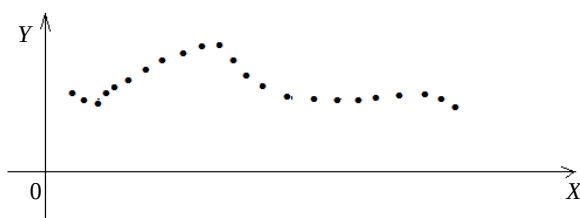


Рис. 6.6. График реальной функции

Такое представление функции можно оформить в виде табл. 6.1.

Таблица 6.1
Табличное представление
функции

X	Y
x_0	y_0
x_1	y_1
x_2	y_2
x_3	y_3
$\vdots$	$\vdots$
x_{n-1}	y_{n-1}
x_n	y_n

Можно назвать такое представление функции цифровым. Таблицу числовых значений легко записать в память компьютера для хранения и обработки.

Сеткой называют структуру, которая возникает, если непрерывные оси координат заменить на дискретные значения; в двумерном случае она напоминает рыболовную сеть (рис. 6.7). Мы будем рассматривать одномерный случай, функции одной переменной. Совокупность значений аргумента $x_0, x_1, x_2, \dots, x_n$ называют узлами сетки, а соответствующие им значения $y_0, y_1, y_2, \dots, y_n$ называют сеточными значениями и определяют сеточную функцию (рис. 6.8). В таком случае говорят, что функция задана на сетке. Сетка называется **равномерной**, если между узлами одинаковое расстояние $h = (x_n - x_0)/n$, тогда $x_i = x_0 + i \cdot h$, где $i = 0 \dots n$ [8].

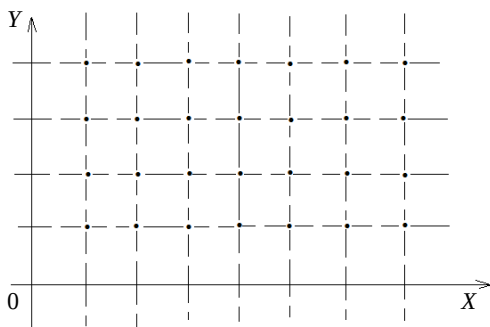


Рис. 6.7. Двумерная сетка

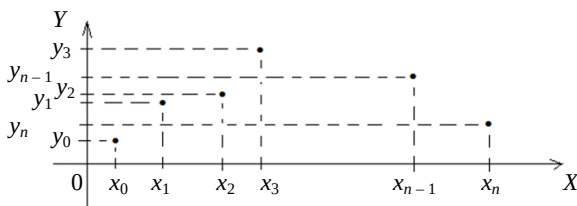


Рис. 6.8. График сеточной функции

Задача **интерполяции** заключается в следующем [8]: пусть заданы значения неизвестной функции $y = f(x)$ $y_0, y_1, y_2, \dots, y_n$

в узлах сетки $x_0, x_1, x_2, \dots, x_n$. Требуется вычислить значение этой функции для любого значения аргумента, принадлежащего внутренней области числового промежутка, ограниченного двумя соседними узлами сетки: $z = f(x)$, где $x \in [x_0; x_n]$. Так как таких значений бесконечно много, то требуется именно вычислить по формулам. Решение задачи интерполяции будет приближённым, поскольку поведение функции между узлами неизвестно (Например, каждый час производят измерение температуры на протяжении суток. Требуется построить непрерывный график изменения температуры.).

Локальная интерполяция характерна тем, что решения на каждом из межузловых промежутков строятся независимо друг от друга. **Глобальная интерполяция** предполагает построение единого решения на всей сетке.

Кусочно-постоянная интерполяция слева предусматривает распространение на весь межузловой промежуток сеточного значения функции в левом узле (рис. 6.9). Аналогично, но в противоположном направлении происходит **кусочно-постоянная интерполяция справа** (рис. 6.10).

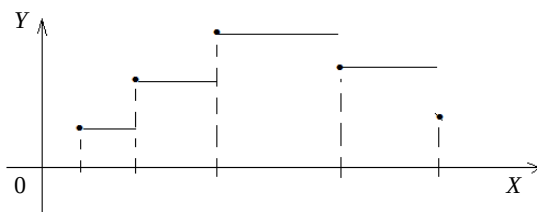


Рис. 6.9. Кусочно-постоянная интерполяция слева

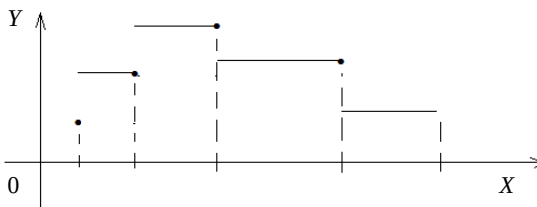


Рис. 6.10. Кусочно-постоянная интерполяция справа

Кусочно-линейная интерполяция заключается в том, что на каждом межузловом промежутке решение строится в виде линейной функции, принимающей в узлах сеточные значения (рис. 6.11). Известно, что через две точки можно провести только одну прямую.

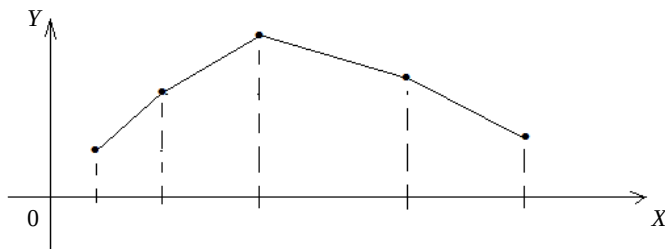


Рис. 6.11. Кусочно-линейная интерполяция

Кусочно-постоянная интерполяция – грубый метод, его погрешность равна разности соседних сеточных значений, зато он не требует вычислений, только сравнения чисел. Кусочно-линейная точнее, но при этом требуется больше вычислений.

Путь уменьшения погрешности – увеличение количества узлов сетки. Рисунок 6.12 иллюстрирует уменьшение погрешности при увеличении количества узлов в 2 раза. Трудоемкость метода при этом тоже возрастает в 2 раза.

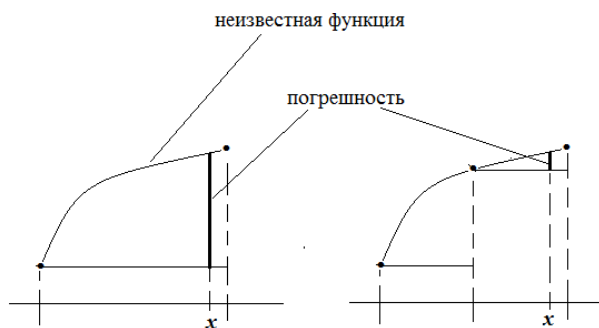


Рис. 6.12. Уменьшение погрешности интерполяции

У подобных методов есть характеристика *порядок точности*, показывающая, как зависит уменьшение погрешности (в данном случае абсолютного значения разности между неизвестным точным значением функции и результатом интерполяции при значении аргумента, равном x) от уменьшения длины межузлового промежутка. В случае кусочно-постоянной или кусочно-линейной интерполяции уменьшение длины межузлового промежутка h в 2 раза приводит к уменьшению погрешности тоже в 2 раза, значит, порядок точности равен 1.

Порядок точности – это показатель степени, в которую необходимо возвести относительное уменьшение длины h (или относительное увеличение количества узлов сетки), чтобы получить относительное уменьшение погрешности.

Недостаток локальных интерполяций заключается в том, что решения, которые дают эти методы, имеют разрывы функции, а также её производных в узлах сетки. В природе, если всё хорошо, то различные процессы описываются достаточно гладкими, дифференцируемыми функциями. А если физические величины изменяются скачкообразно, тогда происходят катастрофы и различные неблагоприятные явления (ударные волны, взрывы и т.д.).

Интерполяция полиномом Лагранжа лишена этого недостатка и даёт решение достаточной гладкости. Известно, что существует единственный полином n -й степени, график которого проходит через $n + 1$ заданную точку на плоскости. При малых значениях n интерполяционный полином Лагранжа даёт хорошую точность интерполяции. Однако при больших значениях n вычислительные погрешности, накапливаемые при расчётах, не позволяют решить задачу интерполяции.

На практике применяют **интерполяцию кубическим сплайном** [8]. Идея этого метода состоит в следующем: на каждом межузловом интервале строится полином третьей степени, в узлах сетки эти полиномы "гладко сшиваются" таким образом, чтобы при переходе через узел первая и вторая производные не испытывали разрыв.

В системах автоматизированного проектирования (САПР) интерполяция является одним из этапов решения сложных расчётных задач, на котором полученная в результате вычислений сеточная функция получает аналитическое представление. Например, при лазерном раскрое листовых материалов сначала формируется точечный контур, после чего с помощью интерполяции получают непрерывную линию, по которой движется лазерный резец.

Моделирование сложных поверхностей полигональными сетками также относится к методам интерполяции.

6.4. Аппроксимация функций

Термин "аппроксимация" означает "приближение". Исходя из данных о неизвестной функции, стараются получить функцию (именно в аналитическом виде), которая бы не сильно отличалась от неизвестной.

Один такой способ – интерполяцию – мы описали выше. При интерполяции должно жёстко соблюдаться условие – значения интерполяции в узлах сетки должны совпадать с исходными сеточными значениями. При аппроксимации функции выполнение этого условия не обязательно. Таким образом, задача **аппроксимации функции** состоит в том, чтобы по исходным данным о неизвестной функции построить функцию, по возможности наиболее похожую на неизвестную.

Рассмотрим аппроксимацию по **методу наименьших квадратов** [8]. Основное в данном методе – это числовой критерий, позволяющий оценить степень схожести функций.

Исходными данными задачи аппроксимации функции являются n значений неизвестной функции $y = f(x)$ $y_1, y_2, \dots, y_n$ при соответствующих значениях аргумента $x_1, x_2, \dots, x_n$, представленные в табл. 6.2.

Таблица 6.2

Исходные данные задачи аппроксимации

X	x_0	x_1	x_2	x_3	$\dots$	x_{n-1}	x_n
Y	y_0	y_1	y_2	y_3	$\dots$	y_{n-1}	y_n

Исходные данные задачи аппроксимации функции могут быть представлены графически (рис. 6.13).

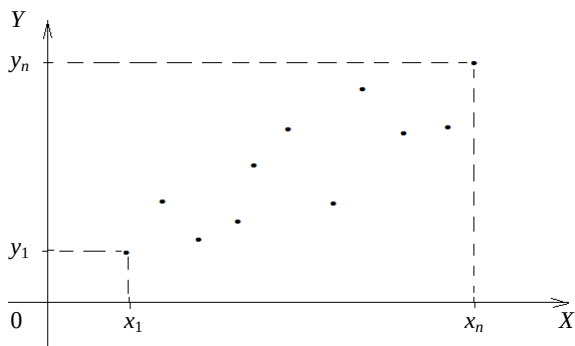


Рис. 6.13. Исходные данные задачи аппроксимации

Критерий метода наименьших квадратов – это сумма квадратов отклонений значений решения при значениях аргумента $x_1, x_2, \dots, x_n$ от соответствующих значений неизвестной функции $y_1, y_2, \dots, y_n$. Отклонения возводятся в квадрат, чтобы отклонения разного знака не компенсировали друг друга.

Задача аппроксимации функции по методу наименьших квадратов решается следующим образом: выбирается класс функций, внутри которого ведётся поиск решения, например линейные функции (полиномы первой степени) вида $P_1(x) = k \cdot x + b$.

Сумма квадратов отклонений обозначается как функционал $G_1(k, b)$, потому что его значение зависит от параметров полинома k и b следующим образом:

$$G_1(k, b) = \sum_{i=1}^n (P_1(x_i) - y_i)^2 = \sum_{i=1}^n (kx_i + b - y_i)^2. \quad (6.8)$$

Теперь необходимо отыскать такие значения параметров k и b , при которых значение функционала $G_1(k, b)$ будет наименьшим. Из курса математического анализа известно, что при минимальных значениях дифференцируемой функции её производная равна нулю. Следовательно, чтобы определить искомые значения параметров, необходимо приравнять нулю частные производные функционала суммы квадратов отклонений:

$$\begin{cases} \frac{\partial G_1(k, b)}{\partial k} = 0, \\ \frac{\partial G_1(k, b)}{\partial b} = 0. \end{cases} \quad (6.9)$$

После подстановки в равенства (6.9) выражения для функционала (6.8) получается:

$$\begin{cases} \frac{\partial \sum_{i=1}^n (kx_i + b - y_i)^2}{\partial k} = 0, \\ \frac{\partial \sum_{i=1}^n (kx_i + b - y_i)^2}{\partial b} = 0. \end{cases} \quad (6.10)$$

Берётся производная, получается система двух (по числу параметров) линейных алгебраических уравнений относительно неизвестных параметров k и b :

$$\begin{cases} nb + (\sum_{i=1}^n x_i)k = \sum_{i=1}^n y_i, \\ (\sum_{i=1}^n x_i)b + (\sum_{i=1}^n x_i^2)k = \sum_{i=1}^n x_i y_i. \end{cases} \quad (6.11)$$

Решение СЛАУ (6.11) – искомые значения параметров k^* и b^* . Линейная функция $P_1^*(x) = k^* \cdot x + b^*$ является решением задачи аппроксимации функции по методу наименьших квадратов (рис. 6.14), а $G_1^* = G_1(k^*, b^*)$ – наименьшая сумма квадратов отклонений среди всех полиномов первой степени. Данное решение позволяет утверждать, что полученная функция – лучшая по приближению неизвестной только среди полиномов первой степени.

Можно решить задачу аппроксимации функции для нескольких классов, например полиномов второй, третьей и т.д. степеней. Решив задачу аппроксимации для каждого класса полиномов, в качестве решения необходимо выбрать тот, для которого сумма квадратов отклонений G_i^* окажется минимальной (рис. 6.15 и рис. 6.16).

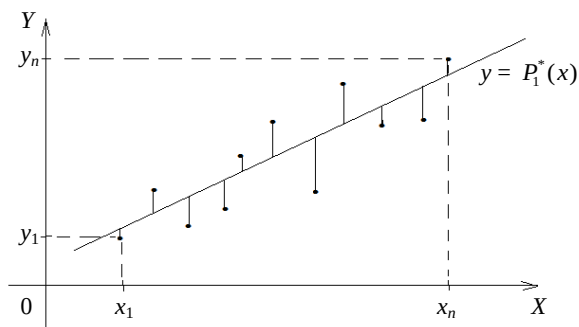


Рис. 6.14. Решение – полином первой степени

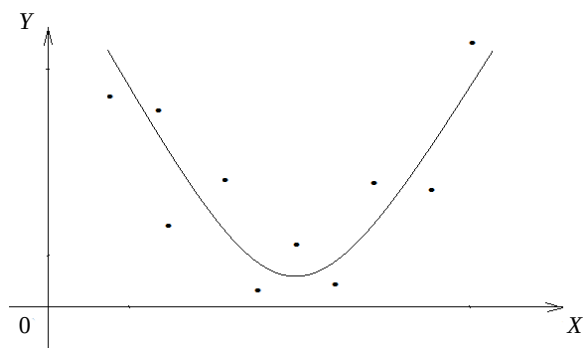


Рис. 6.15. Решение – полином второй степени

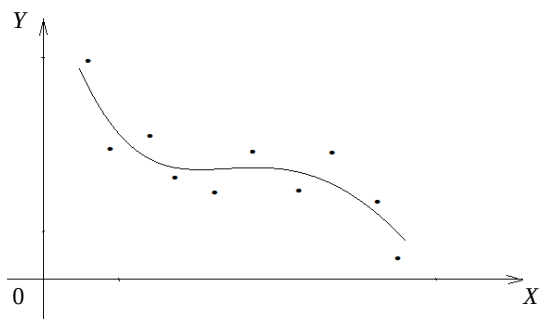


Рис. 6.16. Решение – полином третьей степени

Задача аппроксимации функции методом наименьших квадратов возникает при обработке экспериментальных данных или результатов наблюдений, которые содержат некую составляющую, которая искажает основную зависимость наблюдаемых величин, её называют шум или случайная ошибка. Решение задачи интерполяции, проходящее через все точки исходных данных, в таком случае будет иметь излишнюю сложность. Часто при анализе данных требуется выявить систематическую тенденцию зависимости данных, которая описывается просто, допустим, как линейная, квадратичная или экспоненциальная. Например, строительная организация имеет план построить последовательно за год 12 домов, значит, к первому числу очередного месяца должен быть готов очередной дом. График выполнения работ в идеале – прямая линия под углом 45° . На практике будут небольшие отклонения – даже если всё идёт по плану, один дом могут достроить на два дня позже (шли дожди), другой, наоборот – на день раньше. Экспериментальными данными будут даты постройки каждого дома, для наглядности они могут быть представлены графически.

Рассмотрим два варианта. В первом случае (рис. 6.17) аппроксимация дала линейное решение, совпадающее с идеальным планом потому, что отклонения были небольшими и носили случайный характер.

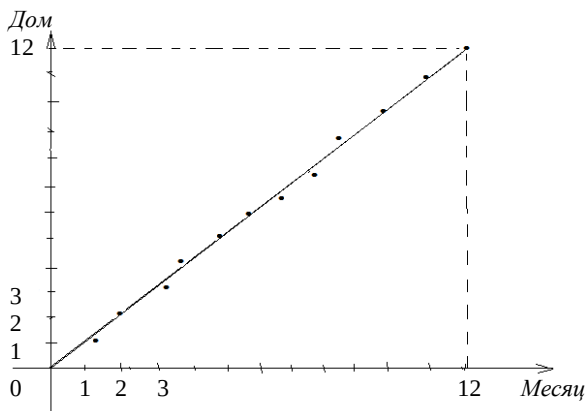


Рис. 6.17. Линейный тренд

Во втором случае (рис. 6.18) аппроксимация выявила нелинейный характер темпов строительства, поэтому стоит задуматься о качестве работ.

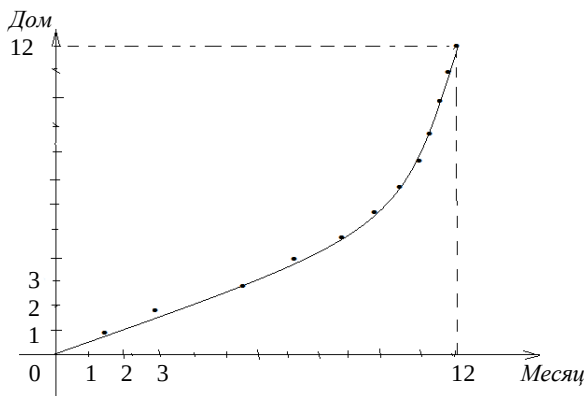


Рис. 6.18. Нелинейный тренд

ЗАКЛЮЧЕНИЕ

В данном учебном пособии авторы сделали попытку показать значение информатики для формирования инженерного подхода. Авторы уверены в том, что программирование помогает развивать логическое и абстрактное мышление, позволяющее выявлять причинно-следственные связи и сосредоточивать внимание на важных аспектах сложных явлений и объектов окружающего мира.

В настоящее время невозможно создание и последующая модернизация сложных систем без применения объектно-ориентированного подхода к проектированию. Примеры, изложенные в данном учебном пособии, помогут в освоении принципов объектно-ориентированного программирования на практике.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Виноградов, С. Н. Логика / С. Н. Виноградов, Ф.Ф. Кузьмин. – Москва : Учпедгиз, 1954. – 176 с.
2. Кнут, Д. Э. Искусство программирования. Т. 1 : Основные алгоритмы / Д. Э. Кнут. – 3-е изд. – Москва : Вильямс, 2015. – 720 с.
3. Кнут, Д. Э. Искусство программирования для ЭВМ : [в 7 т.] : Т. 3 : Сортировка и поиск / Д. Э. Кнут. – Москва : Мир, 1978. – 848 с.
4. Мейер, Б. Почувствуй класс / Б. Мейер. – Москва : БИНОМ. Лаборатория знаний, 2011. – 776 с.
5. Вайсфельд, М. Объектно-ориентированное мышление / М. Вайсфельд. – Санкт-Петербург : Питер, 2014. – 304 с.
6. Объектно-ориентированный анализ и проектирование с примерами приложений / Г. Буч, Р. А. Максимчук, М. У. Энгл [и др.]. – Москва : Вильямс, 2008. – 720 с.
7. Курош, А. Г. Курс высшей алгебры / А. Г. Курош. – Москва : Наука, 1968. – 432 с.
8. Бедарев, И. А. Методы вычислений : учеб. пособие / И. А. Бедарев, Ю. В. Кратова, Н. Н. Федорова ; Новосиб. гос. архитектур.-строит. ун-т (Сибстрин). – Новосибирск : НГАСУ (Сибстрин), 2009. – 116 с.
9. Китайский восточный календарь. – URL: <http://calendar.yoip.ru/china-years-calendar.html> (дата обращения: 25.03.2019). – Текст : электронный.
10. История развития ЭВМ. – Текст : электронный // Викиучебник : веб-сайт. – URL: wikibooks.org/wiki/История_развития_ЭВМ (дата обращения: 27.03.2019).
11. Архитектура фон Неймана. – Текст : электронный // Википедия : веб-сайт. – URL: ru.wikipedia.org/wiki/Архитектура_фон_Неймана (дата обращения: 02.04.2019).
12. БЭСМ-6. – Текст : электронный // Википедия : веб-сайт. – URL: <https://ru.wikipedia.org/wiki/БЭСМ-6> (дата обращения: 25.03.2019).
13. Виртуальный компьютерный музей. – URL: <http://www.computer-museum.ru/histussr/es1020f.htm> (дата обращения: 25.03.2019). – Текст : электронный.
14. Космический корабль Буран. – Текст : электронный // Википедия : веб-сайт. – URL: https://ru.wikipedia.org/wiki/Буран_космический_корабль (дата обращения: 08.04.2019).
15. Boeing X-37. – Текст : электронный // Википедия : веб-сайт. – URL: https://ru.wikipedia.org/wiki/Boeing_X-37 (дата обращения: 08.04.2019).

Учебное издание

Воробьева Алла Петровна
Литвинов Леонид Алексеевич

ОСНОВЫ ИНФОРМАТИКИ

Учебное пособие

Под редакцией Л.А. Литвинова

ISBN 978-5-7795-0879-7



Темплан 2019 г.

Редактор А.В. Тренина

Санитарно-эпидемиологическое заключение
№ 54.НС.05.953.П.006252.06.06 от 26.06.2006 г.

Подписано к печати 18.09.2019. Формат 60 × 84 ¹/₁₆ д.л.

Гарнитура Таймс. Бумага офсетная. Печать цифровая.

Объем 4,4 уч.-изд.л.; 8,75 п.л. Тираж 150 экз. Заказ №

Новосибирский государственный
архитектурно-строительный университет (Сибстрин)
630008, Новосибирск, ул. Ленинградская, 113

Отпечатано