« Виды языков программирования»

 СОДЕРЖАНИЕ

# ВВЕДЕНИЕ с.3

# Глава 1. 1.Теоретическая часть с.4

 1.1.Что такое язык программирования - с.4

 1.2. Для чего нужны языки программирования - с.5

 1.3. Какие существуют языки программирования - с.7

 1.3.1.Фортран - с.7

 1.3.2. Алгол - с.7

 1.3.3. Кобол - с.8

 1.3.4. Лисп - с.8

 1.3.5. Бейсик - с.9

 1.3.6. Форт - с.10

 1.3.7. Паскаль – с.10

 1.3.8. Аdа - с.11

 1.3.9. Си - с.12

 1.3.10. Пролог - с.12

 1.3.11.Java - с.14

 1.3.12. Object Pascal -с.14

 1.3.13 Система визуального объектно-проектированного проектирования Delphi - с.15

ГЛАВА 2. ПРАКТИЧЕСКАЯ ЧАСТЬ - С.19

 2.1.Создание программы «Простейший калькулятор» - с.19

 2.2.Вычисление значения выражений - с.23

 2.3.Вычисление сумм ряда чисел - с.25

 2.4.Вычисление системы функций «Логическая развилка» - с.29

ЗАКЛЮЧЕНИЕ.С. 35

#  Введение

 Внедрение ЭВМ во все сферы человеческой деятельности требует от специалистов разного профиля овладения навыками использования вычислительной техники. Повышается уровень подготовки студентов вузов, которые уже с первых курсов приобщаются к использованию ЭВМ и простейших численных методов, не говоря уже о том, что при выполнении курсовых и дипломных проектов применение вычислительной техники становится нормой в подавляющем большинстве вузов.

Вычислительная техника используется сейчас не только в инженерных расчетах и экономических науках, но и таких традиционно нематематических специальностях, как медицина, лингвистика, психология. В связи с этим можно констатировать, что применение ЭВМ приобрело массовый характер. Возникла многочисленная категория специалистов - пользователей ЭВМ, которым необходимы знания по применению ЭВМ в своей отрасли - навыки работы с уже имеющимся программным обеспечением, а так же создания своего собственного ПО, приспособленного для решения конкретной задачи. И здесь на помощь пользователю приходят описания языков программирования.

# Глава 1. 1.Теоретическая часть.

 **1.1.Что такое язык программирования.**

**Язык программирования** — формальная знаковая система, предназначенная для описания алгоритмов в форме, которая удобна для исполнителя (например, компьютера). Язык программирования определяет набор лексических, синтаксических и семантических правил, используемых при составлении компьютерной программы. Он позволяет программисту точно определить то, на какие события будет реагировать компьютер, как будут храниться и передаваться данные, а также какие именно действия следует выполнять над этими при различных обстоятельствах.

Со времени создания первых программируемых машин человечество придумало уже более двух с половиной тысяч языков программирования. Каждый год их число пополняется новыми. Некоторыми языками умеет пользоваться только небольшое число их собственных разработчиков, другие становятся известны миллионам людей. Профессиональные программисты иногда применяют в своей работе более десятка разнообразных языков программирования.

Создатели языков по-разному толкуют понятие *язык программирования*. Среди общиx мест, признаваемых большинством разработчиков, находятся следующие:

* *Функция:* язык программирования предназначен для написания компьютерных программ, которые применяются для передачи компьютеру инструкций по выполнению того или иного вычислительного процесса и организации управления отдельными устройствами.
* *Задача:* язык программирования отличается от естественных языков тем, что предназначен для передачи команд и данных от человека компьютеру, в то время как естественные языки используются лишь для общения людей между собой. В принципе, можно обобщить определение "языков программирования" - это способ передачи команд, приказов, чёткого руководства к действию; тогда как человеческие языки служат также для обмена информацией.
* *Исполнение:* язык программирования может использовать специальные конструкции для определения и манипулирования структурами данных и управления процессом вычислений.

**1.2. Для чего нужны языки программирования**

 Процесс работы компьютера заключается в выполнении программы, то есть набора вполне определённых команд во вполне определённом порядке. Машинный вид команды, состоящий из нулей и единиц, указывает, какое именно действие должен выполнить центральный процессор. Значит, чтобы задать компьютеру последовательность действий, которые он должен выполнить, нужно задать последовательность двоичных кодов соответствующих команд. Программы в машинных кодах состоят из тысячи команд. Писать такие программы – занятие сложное и утомительное. Программист должен помнить комбинацию нулей и единиц двоичного кода каждой программы, а также двоичные коды адресов данных, используемых при её выполнении. Гораздо проще написать программу на каком-нибудь языке, более близком к естественному человеческому языку, а работу по переводу этой программы в машинные коды поручить компьютеру. Так возникли языки, предназначенные специально для написания программ, - **языки программирования.**

Имеется много различных языков программирования. Вообще-то для решения большинства задач можно использовать любой из них. Опытные программисты знают, какой язык лучше использовать для решения каждой конкретной задачи, так как каждый из языков имеет свои возможности, ориентацию на определённые типы задач, свой способ описания понятий и объектов, используемых при решении задач.

Всё множество языков программирования можно разделить на две группы: **языки низкого уровня** и **языки высокого уровня.**

 К языкам низкого уровня относятся языки ассемблера (от англ. to assemble – собирать, компоновать). В языке ассемблера используются символьные обозначения команд, которые легко понятны и быстро запоминаются. Вместо последовательности двоичных кодов команд записываются их символьные обозначения, а вместо двоичных адресов данных, используемых при выполнении команды, - символьные имена этих данных, выбранные программистом. Иногда язык ассемблера называют мнемокодом или автокодом.

 Большинство программистов пользуются для составления программ языками высокого уровня. Как и обычный человеческий язык, такой язык имеет свой алфавит – множество символов, используемых в языке. Из этих символов составляются так называемые ключевые слова языка. Каждое из ключевых слов выполняет свою функцию, так же как в привычном нам языке нам языке слова, составленные из букв алфавита данного языка, могут выполнять функции разных частей речи. Ключевые слова связываются друг с другом в предложения по определённым синтаксическим правилам языка. Каждое предложение определяет некоторую последовательность действий, которые должен выполнить компьютер.

 Язык высокого уровня выполняет роль посредника между человеком и компьютером, позволяя человеку общаться с компьютером более привычным для человека способом. Часто такой язык помогает выбрать правильный метод решения задачи.

Перед тем как писать программу на языке высокого уровня, программист должен составить *алгоритм* решения задачи, то есть пошаговый план действий, который нужно выполнить для решения этой задачи. Поэтому языки, требующие предварительного составления алгоритма, часто называют *алгоритмическими языками.*

**1.3. Какие существуют языки программирования**

**1.3.1.Фортран**

Языки программирования стали появляться уже с середины 50-х годов. Одним из первых языков такого типа стал язык Фортран (англ. FORTRAN от FORmula TRANslator – переводчик формул), разработанный в 1957 году. Фортран применяется для описания алгоритма решения научно-технических задач с помощью ЦВМ. Так же, как и первые вычислительные машины, этот язык предназначался, в основном, для проведения естественно-научных и математических расчётов. В усовершенствованном виде этот язык сохранился до нашего времени. Среди современных языков высокого уровня он является одним из наиболее используемых при проведении научных исследований. Наиболее распространены варианты Фортран-II, Фортран-IV, EASIC Fortran и их обобщения.

**1.3.2. Алгол**

 После Фортрана в 1958-1960 годах появился язык Алгол (Алгол-58, Алгол-60) (англ. ALGOL от ALGOrithmic Language – алгоритмический язык). Алгол был усовершенствован в 1964-1968 годах – Алгол-68. Алгол был разработан комитетом, в который входили европейские и американские учёные. Он относится к языкам высокого уровня (high-level language) и позволяет легко переводить алгебраические формулы в программные команды. Алгол был популярен в Европе, в том числе СССР, в то время как сравнимый с ним [Фортран](http://ru.wikipedia.org/wiki/%D0%A4%D0%BE%D1%80%D1%82%D1%80%D0%B0%D0%BD) был распространен в США и Канаде. Алгол оказал заметное влияние на все разработанные позднее языки программирования, и, в частности, на язык Pascal. Этот язык так же, как и Фортран, предназначался для решения научно-технических задач. Кроме того, этот язык применялся как средство обучения основам программирования – искусства составления программ.

Обычно под понятием Алгол подразумевается язык *Алгол-60*, в то время как *Алгол-68* рассматривается как самостоятельный язык. Даже когда язык Алгол почти перестал использоваться для программирования, он ещё оставался официальным языком для публикации алгоритмов.

**1.3.3. Кобол**

В 1959 – 1960 годах был разработан язык Кобол (англ. COBOL от COmmom Business Oriented Language – общий язык, ориентированный на бизнес). Это язык программирования третьего поколения, предназначенный, в первую очередь, для разработки бизнес приложений. Также Кобол предназначался для решения экономических задач, обработки данных для банков, страховых компаний и других учреждений подобного рода. Разработчиком первого единого стандарта Кобола являлась Грейс Хоппер (*бабушка* *Кобола*).

 Кобол обычно критикуется за многословность и громоздкость, поскольку одной из целей создателей языка было максимально приблизить конструкции к английскому языку. (До сих пор Кобол считается языком программирования, на котором было написано больше всего строк кода). В то же время, Кобол имел прекрасные для своего времени средства для работы со структурами данных и файлами, что обеспечило ему долгую жизнь в бизнес приложениях, по крайней мере, в США.

**1.3.4. Лисп**

 Почти одновременно с Коболом (1959 – 1960 гг.) в Массачусетском технологическом институте был создан язык Лисп (англ. LISP от LISt Processing – обработка списков). Лисп основан на представлении программы системой линейных списков символов, которые притом являются основной структурой данных языка. Лисп считается вторым после Фортрана старейшим высокоуровневым языком программирования. Этот язык широко используется для обработки символьной информации и применяется для создания программного обеспечения, имитирующего деятельность человеческого мозга.

Любая программа на Лиспе состоит из последовательности *выражений* (форм). Результат работы программы состоит в вычислении этих выражений. Все выражения записываются в виде *списков* — одной из основных структур Лиспа, поэтому они могут легко быть созданы посредством самого языка. Это позволяет создавать программы, изменяющие другие программы или макросы, позволяющие существенно расширить возможности языка.

Основной смысл Лисп-программы "жизнь" в символьном пространстве: перемещение, творчество, запоминание, создание новых миров и т.д. Лисп как метафора мозга, символ, метафора сигнала: "Как происходит биологический анализ сигналов мозгом, как внешний фактор - физическое и химическое воздействие, являющееся для организма раздражителем превращается в биологически значимый сигнал, зачастую жизненно важный, определяющий все поведение человека или животного; и как происходит разделение разных сигналов на положительные, отрицательные и безразличные, индифферентные. Сигнал это уже интегративное понятие. Он представляет собой опознавательный знак группы, комплексных раздражителей, связанных между собой общей историей и причинно следственными отношениями. В этом комплексе, системе раздражителей, сигнальный стимул сам является также составляющим элементом и при иных обстоятельствах его роль может принадлежать другому стимулу из комплекса. В сигнале концентрируется весь прошлый опыт животного или человека.

 **1.3.5. Бейсик**

 В середине 60-х годов (1963 г.) в Дартмутском колледже (США) был создан язык Бейсик (англ. BASIC от Beginner’s Allpurpose Instruction Code – всецелевой символический код инструкций для начинающих). Со временем, когда стали появляться другие диалекты, этот «изначальный» диалект стали называть Dartmouth BASIC. Язык был основан частично на [Фортран II](http://ru.wikipedia.org/w/index.php?title=%D0%A4%D0%BE%D1%80%D1%82%D1%80%D0%B0%D0%BD_II&action=edit) и частично на [Алгол-60](http://ru.wikipedia.org/w/index.php?title=%D0%90%D0%BB%D0%B3%D0%BE%D0%BB-60&action=edit), с добавлениями, делающими его удобным для работы в режиме разделения времени и, позднее, обработки текста и матричной арифметики. Первоначально Бейсик был реализован на мейнфрейме GE-265 с поддержкой множества терминалов. Вопреки распространённому убеждению, в момент своего появления это был [компилируемый](http://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D0%B8%D0%BB%D1%8F%D1%82%D0%BE%D1%80) язык.

Бейсик был спроектирован так, чтобы студенты могли писать программы, используя терминалы с разделением времени. Он создавался как решение для проблем, связанных со сложностью более старых языков. Он предназначался для более «простых» пользователей, не столько заинтересованных в скорости программ, сколько просто в возможности использовать компьютер для решения своих задач. В силу простоты языка Бейсик многие начинающие программисты начинают с него свой путь в программировании.

 **1.3.6. Форт**

 В конце 60-х – начале 70-х годов появился язык Форт (англ. FOURTH – четвёртый). Этот язык стал применяться в задачах управления различными системами после того, как его автор Чарльз Мур написал на нём программу, предназначенную для управления радиотелескопом Аризонской обсерватории.

Ряд свойств, а именно интерактивность, гибкость и простота разработки делают Форт весьма привлекательным и эффективным языком в прикладных исследованиях и при создании инструментальных средств. Очевидными областями применения этого языка являются встраиваемые системы управления. Также находит применение при программировании компьютеров под управлением различных операционных систем.

**1.3.7. Паскаль**

Появившийся в 1972 году язык Паскаль был назван так в честь великого французского математика XVII века, изобретателя первой в мире арифметической машины Блеза Паскаля. Этот язык был создан швейцарским учёным, специалистом в области информатики Никлаусом Виртом как язык для обучения методам программирования. Паскаль – это язык программирования общего назначения.

Особенностями языка являются строгая типизация и наличие средств структурного (процедурного) программирования. Паскаль был одним из первых таких языков. По мнению Н. Вирта, язык должен способствовать дисциплинированию программирования, поэтому, наряду со строгой типизацией, в Паскале сведены к минимуму возможные синтаксические неоднозначности, а сам синтаксис интуитивно понятен даже при первом знакомстве с языком.

 Язык Паскаль учит не только тому, как правильно написать программу, но и тому, как правильно разработать метод решения задачи, подобрать способы представления и организации данных, используемых в задаче. С 1983 года язык Паскаль введён в учебные курсы информатики средних школ США.

**1.3.8. Аdа**

В 1983 году под эгидой Министерства Обороны США был создан язык Ada. Язык замечателен тем, что очень много ошибок может быть выявлено на этапе компиляции. Кроме того, поддерживаются многие аспекты программирования, которые часто отдаются на откуп операционной системе (параллелизм, обработка исключений). В 1995 году был принят стандарт языка Ada 95, который развивает предыдущую версию, добавляя в нее объекно-ориентированность и исправляя некоторые неточности. Оба этих языка не получили широкого распространения вне военных и прочих крупномасштабных проектов (авиация, железнодорожные перевозки). Основной причиной является сложность освоения языка и достаточно громоздкий синтаксис.

**1.3.9. Си**

В настоящее время популярным среди программистов является язык Си (С – буква английского алфавита). Язык Си берёт своё начало от двух языков - BCPL и B. В 1967 году Мартин Ричардс разработал BCPL как язык для написания системного программного обеспечения и компиляторов. В 1970 году Кен Томпсон использовал В для создания ранних версий операционной системы UNIX на компьютере DEC PDP-7. Как в BCPL, так и в В переменные не разделялись на типы - каждое значение данных занимало одно слово в памяти и ответственность на различение, например, целых и действительных чисел целиком ложилась на плечи программиста.

Язык Си был разработан (на основе В) Деннисом Ритчи из Bell Laboratories и впервые был реализован в 1972 году на компьютере DEC PDP-11. Известность Си получил в качестве языка ОС UNIX. Сегодня практически все основные операционные системы были написаны на Си или С++. По прошествии двух десятилетий Си имеется в наличии на большинстве компьютеров. Он не зависит от аппаратной части.

В конце 70-х годов Си превратился в то, что мы называем «традиционный Си». В 1983 году Американским комитетом национальных стандартов в области компьютеров и обработки информации был учрежден единый стандарт этого языка.

 Этот язык имеет богатые средства, позволяет писать гибкие программы, использующие все возможности современных персональных компьютеров.

**1.3.10. Пролог**

 Ещё один язык, который считается языком будущего, был создан в начале 70-х годов группой специалистов Марсельского университета. Это язык Пролог. Своё название он получил от слов «ПРОграммирование на языке ЛОГики». В основе этого языка лежат законы математической логики. Как и язык Лисп, Пролог применяется, в основном, при проведении исследований в области программной имитации деятельности мозга человека. В отличие от описанных выше языков, этот язык не является алгоритмическим. Он относится к так называемым **дескриптивным**(от англ. descriptive – описательный) – описательным языкам. Дескриптивный язык не требует от программиста разработки всех этапов выполнения задачи. Вместо этого, в соответствии с правилами такого языка, программист должен описать базу данных, соответствующую решаемой задаче, и набор вопросов, на которые нужно получить ответы, используя данные из этой базы.

В последние десятилетия в программировании возник и получил существенное развитие **объектно-ориентированный** подход. Это метод программирования, имитирующий реальную картину мира: информация, используемая для решения задачи, представляется в виде множества взаимодействующих объектов. Каждый из объектов имеет свои свойства и способы поведения. Взаимодействие объектов осуществляется при помощи передачи сообщений: каждый объект может получать сообщения от других объектов, запоминать информацию и обрабатывать её определённым способом и, в свою очередь, посылать сообщения. Так же, как и в реальном мире, объекты хранят свои свойства и поведение вместе, наследуя часть из них от родительских объектов.

Объектно-ориентированная идеология используется во всех современных программных продуктах, включая операционные системы.

Первый объектно-ориентированный язык *Simula-67* был создан как средство моделирования работы различных приборов и механизмов. Большинство современных языков программирования – объектно-ориентированные. Среди них последние версии языка *Turbo-Pascal, C++, Ada* и другие.

В настоящее время широко используются системы **визуального программирования** *Visual Basic, Visual C++, Delphi* и другие. Они позволяют создавать сложные прикладные пакеты, обладающие простым и удобным пользовательским интерфейсом.

**1.3.11.Java**

 С 1995 года стал широко распространяться новый объектно-ориентированный язык программирования Java, ориентированный на сети компьютеров и, прежде всего, на Internet. Синтаксис этого языка напоминает синтаксис языка C++, однако эти языки имеют мало общего. Java интерпретируемый язык: для него определены внутреннее представление (bytecode) и интерпретатор этого представления, которые уже сейчас реализованы на большинстве платформ. Интерпретатор упрощает отладку программ, написанных на языке Java, обеспечивает их переносимость на новые платформы и адаптируемость к новым окружениям. Он позволяет исключить влияние программ, написанных на языке Java, на другие программы и файлы, имеющиеся на новой платформе, и тем самым обеспечить безопасность при выполнении этих программ. Эти свойства языка Java позволяют использовать его как основной язык программирования для программ, распространяемых по сетям (в частности, по сети Internet).

**1.3.12. Object Pascal**

Object Pascal создавался сотрудниками компании Apple Computer (некоторые из которых были участниками проекта Smalltalk) совместно с Никлаусом Виртом (Niklaus Wirth), создателем языка Pascal. Object Pascal известен с 1986 года и является первым объектно-ориентированным языком программирования, который был включен в Macintosh Programmer's Workshop (MPW), среду разработки для компьютеров Macintosh фирмы Apple.

В этом языке нет методов класса, переменных класса, множественного наследования и метаклассов. Эти механизмы исключены специально, чтобы сделать язык простым для изучения начинающими "объектными" программистами.

**1.3.13. Система визуального объектно-проектированного проектирования Delphi**

 Появление Delphi не могло пройти незамеченным среди многочисленных пользователей компьютера. Оценки экспертов, изучающих возможности этого нового продукта фирмы Borland, обычно окрашены в восторженные тона. Основное достоинство Delphi состоит в том, что здесь реализованы идеи визуального программирования. Среда визуального программирования превращает процесс создания программы в приятное и легко понимаемое конструирование приложения из большого набора графических и структурных примитивов.

 Система Delphi позволяет решать множество задач, в частности:

* Создавать законченные приложения для Windows самой различной направленности: от чисто вычислительных и логических, до графических и мультимедиа.
* Быстро создавать (даже начинающим программистам) профессионально выглядящий оконный интерфейс для любых приложений.
* Создавать мощные системы работы с локальными и удаленными базами данных
* Создавать справочные системы (файлы .hlp) для своих приложений и мн. др.

Delphi – чрезвычайно быстро развивающаяся система. Первая версия – Delphi 1.0 была выпущена в феврале 1995 г. А затем новые версии выпускались ежегодно.

Каждая последующая версия Delphi дополняла предыдущую.Большинство версий Delphi выпускается в нескольких вариантах: Standart – стандартном, Professional – профессиональном, Client/Server – клиент/сервер, Enterprise – разработка баз данных предметных областей. Различаются варианты в основном разным уровнем доступа к системам управления базами данных. Последние варианты - Client/Server и Enterprise, в этом отношении наиболее мощные.

Delphi - это комбинация нескольких важнейших технологий:

- Высокопроизводительный компилятор в машинный код

- Объектно-ориентированная модель компонент

- Визуальное (а, следовательно, и скоростное) построение приложений из программных прототипов

- Масштабируемые средства для построения баз данных

**Структура экрана в среде Delphi.**

После вызова Delphi в Windows появляются несколько окон (рис 1.):

* главное окно,
* окно формы,
* окно инспектора объектов,
* окно дерева объектов,
* окно кода программ

 Рассмотрим расположенное в верхней части экрана графическое меню системы Delphi, составленное из пиктограмм.В левой части графического меню находится панель инструментов. Инструменты выполняют некоторые команды главного меню - такое дублирование часто практикуется в инструментальных средах.На этой панели есть, в частности, кнопка сохранения проекта на диске, кнопка открытия проекта, кнопка запуска программы на выполнение.



Рис1. Структура экрана в среде Delphi.

 Следующая часть графического меню - палитра компонентов, устроенная в виде наборов пиктограмм. Совокупность наборов составляет библиотеку визуальных компонентов (VCL). Имеется несколько категорий компонентов, каждая из которых связана со своей закладкой. С помощью палитры компонентов мы будем создавать экземпляры компонентов (или объекты) в форме.

Для того чтобы разместить объект в форме, нужно "щелкнуть" на соответствующей кнопке палитры и затем щелкнуть внутри окна формы: в указанное место формы будет вставлен объект - экземпляр компонента выбранного типа.

Окно Object Inspector - это окно, отображающее свойства либо формы, либо размещенного на форме объекта. В нашем случае текущим компонентом является форма, поэтому на рисунке окно свойств показывает свойства формы.

Окно свойств имеет две закладки - Properties и Еvents, с помощью которых можно получить в окне строки (поля) для задания, соответственно, свойств компонента (т. е. объекта или формы) и его реакции на различные события. Свойство определяет атрибут компонента, например, размер кнопки или шрифт метки. Событие означает, например, такие действия, как щелчок мыши на кнопке или закрытие окна.

Окно дерева объектов появилось в версии 6 и предназначено для наглядного отображения связей между отдельными объектами, размещенными на активной форме или в активном модуле данных.

Окно кода программы предназначено для создания и редактирования текста программы. Первоначально оно содержит минимальный исходный текст.

**Проекты Delphi.** Проект Delphiсостоит изформ, модулей, установок параметров проекта, ресурсов и т.д.Вся эта информация размещается в файлах. Многие из этих файлов автоматически создаются Delphi, когда вы строите ваше приложение. Ресурсы, такие как битовые матрицы, пиктограммы и т.д., находятся в файлах, которые вы получаете из других источников или создаете при помощи многочисленных инструментов и редакторов ресурсов, имеющихся в вашем распоряжении. Кроме того, компилятор также создает файлы.

Создающиеся в процессе проектирования файлы показаны в табл. 3.

Главной частью приложения является файл проекта (.dpr), содержащий код на языке Object Pascal, с которого начинается выполнение программы и который обеспечивает инициализацию других модулей. Он создается и модифицируется Delphi автоматически в процессе разработки приложения.

 Имя, которое дается файлу проекта в процессе сохранения, становится именем исполняемого файла.

В настоящее время вышла уже 7-я версия системы Delphi. За рекордно короткий срок она стала одной из самых популярных систем программирования в мире. Многие разработчики в мире твердо ориентируются на использование Delphi как на инструмент, позволяющий создавать высокоэффективные клиент-серверные приложения.

**ГЛАВА 2. ПРАКТИЧЕСКАЯ ЧАСТЬ.**

**2.1.Создание программы «Простейший калькулятор»**

 Калькулятор (лат. calculātor «счётчик») - электронное вычислительное устройство для выполнения операций над числами или алгебраическими формулами.



Рис. 2 – Программа калькулятор до запуска. Изначальный вид.

 Для начала я создаю примерную форму, как будет выглядеть в дальнейшем мой калькулятор. Добавляю в форму 8 кнопки (Button1,2,3,4,56,7,8). Они будут выполнять функции (сложения, вычитания, умножения, деления, нахождения синуса, косинуса квадрата и корня квадратного). Так же в форме есть две кнопки (Label1,2). Они предназначены для вывода готовых данных на экран. Кнопки Edit1,2 отвечают за ввод числовых данных, которые необходимо рассчитать.

 Для того, чтобы заставить кнопку «+» складывать данные, кнопку «sin» находить угол альфа, необходимо знать коды программирования.

 Для операции «сложение» программный код будет выглядеть так:

 Procedure TForm1.Button1.Click(Sender: TObject);

 var a, b, c: Single;

 begin

a:= StrToFloat (Edit1.Text);

b:= StrToFloat (Edit2.Text);

c:= a + b;

Label2. Caption:= FloatToStr (c);

end;

Для операции «вычитание» программный код будет выглядеть так:

Procedure TForm1.Button1.Click(Sender: TObject);

var a, b, c: Single;

begin

a:= StrToFloat (Edit1.Text);

b:= StrToFloat (Edit2.Text);

c:= a-b;

Label2. Caption:= FloatToStr (c);

end;

Для операции «произведение» программный код будет выглядеть так:

Procedure TForm1.Button1.Click(Sender: TObject);

var a, b, c: Single;

begin

a:= StrToFloat (Edit1.Text);

b:= StrToFloat (Edit2.Text);

c:= a\*b;

Label2. Caption:= FloatToStr (c);

end;

Для нахождения «sin» угла код будет выглядеть так:

Procedure TForm1.Button1.Click(Sender: TObject);

var a, c: Single;

begin

a:= StrToFloat (Edit1.Text);

c:= sin(a);

Label2. Caption:= FloatToStr (c);

end;

Для нахождения «cos» угла код будет выглядеть так:

Procedure TForm1.Button1.Click(Sender: TObject);

var a, c: Single;

begin

a:= StrToFloat (Edit1.Text);

c:= cos(a);

Label2. Caption:= FloatToStr (c);

end;

Для нахождения «квадрата» код будет выглядеть так:

Procedure TForm1.Button1.Click(Sender: TObject);

var a, c: Single;

begin

a:= StrToFloat (Edit1.Text);

c:=SQR(a);

Label2. Caption:= FloatToStr (c);

end;

Для нахождения «квадратного корня» код будет выглядеть так:

Procedure TForm1.Button1.Click(Sender: TObject);

var a, c: Single;

begin

a:= StrToFloat (Edit1.Text);

c:= SQRT(a);

Label2. Caption:= FloatToStr (c);

end;

 Необходимо обратить внимание на команду «Single». Она считает целые числа. А команда «Real» помимо целых, считает еще и дробные числа. Так же следует заметить, что после каждой команды не стоит забывать ставить знак «;», иначе при компиляции будет выходить синтаксическая ошибка.

В Lazarus-e также можно менять свойства объектов:

1. Name - (программное имя объекта);
2. Height – (ширина объекта);
3. Left, Top – (положение объекта относительно границы; влево и вверх);
4. Caption – (редактируемое имя объекта);
5. Font – редактирование (размера, шрифта, курсива, подчеркивания и т.д.) текста;
6. Text (редактируемая надпись на объекте;
7. Image (вставка картинки по выбору пользователя).

 После завершения редактирования и написания исходного кода программы необходимо сделать компиляцию. Делается она при помощи нажатия кнопки «F9». Если все сделано правильно, компиляция завершится успешно и откроется созданная пользователем программа.

 Если же компиляция по какой-то причине завершается некорректно, программа сама выделит в диалоговом окне предположительное место, где скорее всего допущена ошибка. Поэтому следует быть внимательным при составлении программы. Любая синтаксическая ошибка (отсутствие точки, точки с запятой, лишний знак, либо цифра в исходном коде) ведет к неправильной компиляции.





Рис. 3 – Программа «Калькулятор» после запуска.

 Программа составлена правильно, компилирование объекта прошло успешно. Сейчас, для того, чтобы вычислить сумму, разность, произведение или деление чисел, необходимо задать свои значения, и, созданная программа «Калькулятор» быстро и правильно рассчитает ответ.

**2.2.Вычисление значения выражений**

 В среде Lazarus можно также вычислять значения сложных математических выражений.

 Все, что нам необходимо сделать, это правильно составить формулу, чтобы Lazarus смог скомпилировать ее, а затем и решить.



Рис. 4 – Программа «вычисление значения выражений» до запуска

Для начала при составлении программы между «procedure» и «begin» вводим команду var alfa………y:real; она необходима для расчета десятичных чисел. Также нужно ввести команду «math» в «uses», иначе некоторые функции в программе работать не будут.

uses

Classes, SysUtils, FileUtil, LResources, Forms, Controls, Graphics, Dialogs, ExtCtrls, StdCtrls, Math;

Вот так выглядит код программы «вычисление значения выражений» в Lazarus-е:

procedure TForm1.SpeedButton1Click(Sender: TObject);

var x,y: Single;

begin

x:= StrToFloat (Edit1.Text);

y:= ((sin(x))/2)+3;

Label3.Caption:=FloatToStr(y);

end;





Рис. 5 – Программа «Вычисление выражений» после запуска.

Программа составлена правильно, интерпретация прошла успешно. Сейчас, для того, чтобы рассчитать функцию «y», необходимо задать в формулу свои значения.

**2.4.Вычисление сумм ряда чисел**

С помощью суммы рядов чисел можно: - разложить функцию в степенной ряд; - выполнить приближенные вычисления значений функции; - выполнить вычисления пределов; - выполнить вычисление определенных интегралов; - выполнить вычисление логарифмов; - выполнить интегрирование дифференциальных уравнений; - решить уравнение первого порядка итерационным методом.

 Итерация – это повторяемое выполнение некоторого действия до тех пор, пока не будет удовлетворено некоторое условие. Ряд считается заданным, если дан закон, по которому можно вычислить любой член ряда, и известен порядковый номер этого числа. Среди рядов есть сходящиеся ряды и расходящиеся. Если значение частичных сумм Sn при неограниченном возрастании n стремится к некоторому числу А, ряд называется сходящимся, а число А при этом называют суммой. Таким образом, при неограниченном возрастании n значение Sn сколь угодно мало отличается от А, т.е. число А предел последовательности Sn.

 В данном задании необходимо рассчитать сумму чисел и вычислить арифметическое выражение по формуле:





Рис. 6 – Программа «Вычисление сумм ряда чисел» до запуска

Код программы «Вычисление сумм ряда чисел» будет выглядеть следующим образом:

uses

Classes, SysUtils, FileUtil, LResources, Forms, Controls, Graphics, Dialogs, ExtCtrls, StdCtrls, Math;

type

{ TForm1 }

TForm1 = class (TForm)

Button1: TButton;

Edit1: TEdit;

Image1: TImage;

Label1: TLabel;

Label2: TLabel;

Label3: TLabel;

Label4: TLabel;

Label5: TLabel;

procedure Button1Click(Sender: TObject);

private

{ private declarations }

public

{ public declarations }

end;

var

Form1: TForm1;

implementation

{ TForm1 }

procedure TForm1.Button1Click(Sender: TObject);

var n, factorial: integer; x, y, s: real;

begin

x:=StrToFloat(Edit1.Text);

s:=0;

factorial:=1;

for n:=1 to 25 do

begin

s:=s + power(x,(n-1))/factorial;

factorial:=factorial\*(n+1);

end;

Label4.Caption:=FloatToStr(s);

y:=(power(2.76,x)-1)/x;

Label5.Caption:=FloatToStr(y);

end;

initialization

{$I unit1.lrs}

end.





Рис. 7 – Программа «Вычисление сумм ряда чисел» после запуска

Программа составлена правильно, компилирование объекта прошло успешно. Сейчас, для того, чтобы вычислить сумму ряда чисел, необходимо задать в формулу свои значения, и, созданная программа, аналогично калькулятору, рассчитает ответ.

**2.3.Вычисление системы функций «Логическая развилка»**

На практике решение большинства задач не удается описать с помощью программ линейной структуры. При этом после проверки некоторого условия выполняется та или иная последовательность операторов, однако происходит нарушение естественного порядка выполнения операторов. Для этих целей используют управляющие операторы. Условный оператор используется для реализации разветвлений в программе, которые происходят при выполнении некоторого условия и имеет следующую структуру:

IF <логическое выражение> THEN серия1 ELSE серия2;

 Если логическое выражение, выступающее в качестве условия, принимает значение False, то выполняются операторы, расположенные после else (серия2), если True, — операторы, следующие за then. При записи логического выражения следует избегать знака = (равно) для действительных переменных, так как они представляются неточно, а поэтому может не произойти совпадений значений выражений, стоящих слева и справа от знака равно. Для устранения указанного недостатка следует требовать выполнения условия с заданной точностью, т.е. вместо отношения X = Y рекомендуется, например: Abs(X - Y) < 1E-8.

 Поскольку развилка может быть неполной, то возможна и неполная форма записи условного оператора:

IF <логическое выражение> THEN серия;

 Условный оператор реализует разветвление вычислительного процесса по двум направлениям, одно из которых осуществляется при выполнении условия, другое — в противном случае. Для реализации разветвлений более чем по двум направлениям необходимо использовать несколько условных операторов.

 В данном задании нужно вычислить значение функции y(x), где:



Ser ( ) = a2

Power (3,5) = 35 (степень)

Sqrt ( ) = (квадратный корень)

Abs ( ) = | | (модуль)

:= sin(a) = sin (синус)

:= cos(x) = cos (косинус)



Рис. 8 – Программа «Логическая развилка» до запуска

Аналогично предыдущим заданиям составляем программу. Код программы «Логическая развилка» на языке Pascal будет выглядеть следующим образом:

unit Unit1;

{$mode objfpc}{$H+}

interface

uses

Classes, SysUtils, FileUtil, LResources, Forms, Controls, Graphics, Dialogs, ExtCtrls, StdCtrls;

type

{ TForm1 }

TForm1 = class(TForm)

Button1: TButton;

Edit1: TEdit;

Image1: TImage;

Label1: TLabel;

Label2: TLabel;

Label3: TLabel;

procedure Button1Click(Sender: TObject);

private

{ private declarations }

public

{ public declarations }

end;

var

Form1: TForm1;

implementation

{ TForm1 }

procedure TForm1.Button1Click(Sender: TObject);

var x, y: real;

begin

x:=StrToFloat(Edit1.Text);

if x<-5 then

y:=ln(1+abs(cos(x)))/sqr(x) else

if x>=2 then

y:=cos(x) else

y:=(sin(x)-2\*x)/5;

Label3.Caption:=FloatToStr(y);

end;

initialization

{$I unit1.lrs}

end.

Компилятор – (англ. – compiler – составитель, собиратель) читает всю программу целиком, делает ее перевод и создает законченный вариант программы на машинном языке, который затем и выполняется.

Интерпретатор – (англ. – interpreter – истолковыватель, устный переводчик) переводит и выполняет команду строку за строкой.

После того, как программа откомпилирована, ни сама исходная программа, ни компилятор более не нужны. В то же время, программа, обрабатываемая интерпретатором, должна заново переводится на машинный язык при каждом очередном запуске программы. Откомпилированные программы работают быстрее, но интерпретируемые проще исправлять и изменять.





Рис. 9 – Программа ««Логическая развилка» после запуска.

##  ЗАКЛЮЧЕНИЕ

 Программирование и сама наука информатика необходима человеку и обществу для технического прогресса и совершенствования во всех абсолютно во всех научных областях

 В последние годы наблюдается быстрое развитие компьютерных технологий. Компьютер внедряется практически во все сферы нашей жизни, а во многих из них становится просто незаменимым.

 Изобретение языков программирования высшего уровня, а также их постоянное совершенствование и развитие позволило человеку не только общаться с машиной и понимать ее, но и использовать ЭВМ для сложнейших расчетов в самых различных областях жизнедеятельности человека.

 В данной работе были рассмотрены самые распространенные зяыки программирования, которые используются для научных вычислений, для обучения программированию.

 Несмотря на то, что современный уровень развития языков программирования находится на достаточно высоком уровне, тенденция их развития, а также развития информационных технологий в целом складывается таким образом, что можно предположить, что в ближайшем будущем человеческие познания в этой сфере помогут произвести на свет языки, умеющие понимать, обрабатывать и передавать информацию в виде мысли, слова, звука и жеста.

 **БИБЛИОГРАФИЧЕСКИЙ СПИСОК**

1. Акулов О.А., Медведев Н.В. Информатика, М.:АСТ-ПРЕСС, Инфорком-Пресс, 2006 г;
2. Алексеев А.П. Информатика и ИКТ 2001.-М.: СОЛОН-Р, 2001.-364 с. … В.А. Лебедев — Уфа, 2007г;
3. Гейн А. Г., Григорьев С. Г. Информатика и информационные технологии. / Урал., 2008 г;
4. Информатика: Учеб. пособие для студ. вузов/ А.В.Могилев, Н.И.Пак, Е.К.Хённер; Под ред. Е.К.Хённера. – М., 2003 г. – 816 с;
5. Информатика и информационно-коммуникационные технологии Учебник / Под ред. проф. Н.В.Макаровой.-М.: 2007 г;
6. Информатика и компьютерная техника, Базовый курс/С.В. Симонович и др. - СПБ.: Питер, 2009 640с Информатика. Вычислительная техника. Крылов С.С., Лешинер И.Р., Якушкин П.А., 2009 г;
7. Компьютерные сети, СПб., "Питер", 2001 г. Н.А. Олифер, Н.В. Макарова.
8. Угринович Н.Д. Информатика и информационные технологии.М.:БИНОМ, Лаборатория знаний, 2008г;
9. Острейковский В.А., Информатика, М., "Высшая школа", 2000 г. В.Г.Олифер;
10. Щауцукова Л.З., Информатика, под ред. Т.А. Бурмистрова. М: Просвещение, 2003