ГОСУДАРСТВЕННЫЙ КОМИТЕТ СВЯЗИ, ИНФОРМАТИЗАЦИИ И ТЕЛЕКОММУНИКАЦИОННЫХ ТЕХНОЛОГИИ РЕСПУБЛИКИ УЗБЕКИСТАН

НУКУССКИЙ ФИЛИАЛ ТАШКЕНТСКОГО УНИВЕРСИТЕТА ИНФОРМАЦИОННЫХ ТЕХНОЛОГИИ

Факультет:

Компьютер инжиниринг

КУРСОВАЯ РАБОТА

По предмету:Программирование на языкеС++

На тему: Составление программы на языке с++ для решение уравнение

Выполнила:студентка 2в курса Компьютер инжиниринга

Тилепова А.

Принял:Ядгаров Ш.

НУКУС 2015

ПЛАН:

- **Введение**
- **Основная часть**
  - **Создание C #**
  - **Описание использованных програмних средств**
  - **Решение квадратного уравнения в C++**
- **Заключение**
- **Прикладной работа**
- **Список литературы**

Введение

На практике в большинстве случаев найти точной решение возникшей математической задачи не удается. Это происходит главным образом не потому, что мы не умеем этого сделать, а поскольку искомое решение обычно не выражается в привычных для нас элементарных или других известных функциях. Поэтому большое значение приобрели численные методы, особенно в связи с возрастанием роли математических методов в различных областях науки и техники и с появлением высокопроизводительных ЭВМ.

В настоящей курсовой работе рассмотрена важная, с точки зрения прикладных задач: метод Крамера для решение линейных алгебраических уравнений.

Язык C# был создан корпорацией Microsoft для поддержки среды .NET Framework и опирается на богатое наследие в области программирования. Его главным разработчиком был Андерс Хейльсберг (Anders Hejlsberg) — известнейший специалист по программированию. C# происходит напрямую от двух самых удачных в области программирования языков: C и C++. От языка C он унаследовал синтаксис, многие ключевые слова и операторы, а от C++ — усовершенствованную объектную модель.

Создание C

Языки программирования не существуют в пустоте. Напротив, они тесно связаны друг с другом таким образом, что на каждый новый язык оказывают в той или иной форме влияние его предшественники. Это процесс, в ходе которого свойства одного языка приспосабливаются к другому языку, полезные нововведения внедряются в существующий контекст, а устаревшие конструкции удаляются. Таким путем развиваются языки программирования и совершенствуется искусство программирования. И в этом отношении C# не является исключением. У языка программирования C# “богатое наследство”. Он является наследником двух самых удачных языков программирования: C и C++.

Создание C знаменует собой начало современной эпохи программирования. Язык C разработан Деннисом Ритчи в 1970-е годы для программирования на мини-ЭВМ DEC PDP-11 под управлением операционной системы Unix. C появился в результате революции в структурном программировании в 1960-е годы. До появления структурного программирования писать большие программы было трудно, поскольку логика программы постепенно вырождалась в так называемый «макаронный» код – запутанный клубок безусловных переходов, вызовов и возвратов, которые трудно отследить. В структурированных языках программирования этот недостаток устранялся путем ввода строго определенных управляющих операторов, подпрограмм с локальными переменными, и других усовершенствований. Благодаря применению методов структурного программирования сами программисты стали более организованными, надежными и управляемыми.

И хотя в то время существовали и другие структурированные языки программирования, именно в C впервые удалось добиться удачного сочетания эффективности, изящества и выразительности. Благодаря своему краткому, но простому синтаксису в сочетании с принципом, ставившим во

главу угла программиста, а не сам язык, С быстро завоевал многих сторонников. Сейчас уже нелегко представить себе, что С оказался своего рода «струей свежего воздуха», которого так не хватало программистам. В итоге С стал самым распространенным языком программирования в 1980-е годы.

Но даже у такого достойного языка, как С, имелись свои ограничения. К числу самых труднопреодолимых его ограничений относится неспособность справиться с большими программами. Как только проект достигает определенного масштаба, язык С тут же ставит предел, затрудняющий понимание и сопровождение программ при их последующем разрастании. Конкретный предел зависит от самой программы, программиста и применяемых инструментальных средств, тем не менее, всегда существует «порог», за которым программа на С становится неуправляемой.

К концу 1970-х годов масштабы многих проектов приблизились к пределам, с которыми уже не могли справиться методики структурного программирования вообще и язык С в частности. Для решения этой проблемы было открыто новое направление в программировании – так называемое объектно-ориентированное программирование(ООП). Применяя метод ООП, программист мог работать с более крупными программами. Но главная трудность заключалась в том, что С, самый распространенный в то время язык, не поддерживал ООП. Стремление к созданию объектно-ориентированного варианта С в конечном итоге привело к появлению С++.

Язык С++ был разработан в 1979 году Бьярне Страуструпом, работавшим в компании Bell Laboratories, базировавшейся в Мюррей-Хилл, штат Нью-Джерси. Первоначально новый язык назывался «С с классами», но в 1983 году он был переименован в С++. Язык С полностью входит в состав С++, а следовательно, С служит основанием, на котором зиждется С++. Большая часть дополнений, введенных Страуструпом, обеспечивала плавный переход к ООП. И вместо того, чтобы изучать совершенно новый язык,

программирующему на C требовалось лишь освоить ряд новых свойств, чтобы воспользоваться преимуществами методики ООП.

В течении 1980-х годов C++ все еще оставался в тени, интенсивно развиваясь, но к началу 1990-х годов, когда он уже был готов для широкого применения, его популярность в области программирования заметно возросла. К концу 1990-х годов он стал наиболее широко распространенным языком программирования и в настоящее время по-прежнему обладает неоспоримыми преимуществами языка разработки высокопроизводительных программ системного уровня.

Важно понимать, что разработка C++ не была попыткой создать совершенно новый язык программирования. Напротив, это была попытка усовершенствовать уже существовавший довольно удачный язык. Такой подход к разработке языков программирования, основанный на уже сущетсвующем языке и совершенствующий его далее, превратился в упрочившуюся тенденцию, которая продолжается до сих пор.

Следующи важным шагом в развитии языков программирования стала разработка Java. Java представляет собой структурированный, объектно-ориентированный язык с синтаксисом и конструктивными особенностями, унаследованными от C++. Несмотря на то, что в Java успешно решаются многие вопросы переносимости программ в среде Интернета, его возможности все же ограничены. Ему, в частности, не достает межъязыковой возможности взаимодействия, называемой также многоязыковым программированием. Это возможность кода, написанного на одном языке, без труда взаимодействовать с кодом, написанным на другом языке. Межъязыковая возможность взаимодействия требуется для построения крупных, распределенных программных систем. Она желательна также для создания отдельных компонентов программ, поскольку наиболее ценным компонентом считается тот, который может быть использован в самых разных языках программирования и в самом большом числе операционных

сред. Другой возможностью, отсутствующей в Java, является полная интеграция с платформой Windows.

Для удовлетворения этих и других потребностей программирования корпорация Microsoft разработала в конце 1990-х годов язык C# как часть общей стратегии .NET. Впервые он был выпущен в виде альфа-версии в середине 2000 года. Главным разработчиком C# был Андерс Хейльсберг – один из ведущих в мире специалистов по языкам программирования, который может похвалиться рядом достижений в данной области. Достаточно сказать, что в 1980-е годы он был автором очень удачной и имевшей большое значение разработки – языка Turbo Pascal, изящная реализация которого послужила образцом для создания всех последующих компиляторов.

Язык C# непосредственно связан с C, C++ и Java. И это не случайно. Ведь это три самых широко распространенных и признанных во всем мире языка

программирования. Кроме того, на момент создания C# практически все профессиональные программисты уже владели C, C++ и Java. Благодаря тому, что C# построен на столь прочном и понятном основании, перейти на этот язык из C, C++ или Java не представляло особого труда. А поскольку и Хейльсбергу не нужно было изобретать велосипед, то он мог сосредоточиться непосредственно на усовершенствованиях и нововведениях в C#.

В C# имеется немало новых средств, и самое важное из них связано со встроенной поддержкой программных компонентов. В действительности C# может считаться компонентно-ориентированным языком программирования, поскольку в него внедрена встроенная поддержка написания программных компонентов. Например, в состав C# входят средства прямой поддержки таких составных частей программных компонентов, как свойства, методы и события. Но самой важной компонентно-ориентированной особенностью

этого языка, вероятно, является возможность работы в безопасной среде многоязыкового программирования. **Создание C #**

С момента выпуска исходной версии 1.0 развитие C# происходило быстро. Вскоре после версии 1.0 корпорация Microsoft выпустила версию 1.1, в которую было внесено немало корректив, но мало значительных возможностей. Однако ситуация совершенно изменилась после выпуска версии C# 2.0.

Появление версии 2.0 стало поворотным моментом в истории развития C#, поскольку в нее было введено много новых средств, в том числе обобщения, частичные типы и анонимные методы, которые основательно расширили пределы возможностей и область применения этого языка, а также повысили его эффективность. После выпуска версии 2.0 "упрочилось" положение C#. Ее появление продемонстрировало также приверженность корпорации Microsoft к поддержке этого языка в долгосрочной перспективе.

Следующей значительной вехой в истории развития C# стал выпуск версии 3.0. В связи с внедрением многих новых свойств в версии C# 2.0 можно было ожидать некоторого замедления в развитии C#, поскольку программистам требовалось время для их освоения, но этого не произошло. С появлением версии 3.0 корпорация Microsoft внедрила ряд новшеств, совершенно изменивших общее представление о программировании. К числу этих новшеств относятся, среди прочего, лямбда-выражения, язык интегрированных запросов (LINQ), методы расширения и неявно типизированные переменные. Конечно, все эти новые возможности очень важны, поскольку они оказали заметное влияние на развитие данного языка, но среди них особенно выделяются две: язык интегрированных запросов (LINQ) и лямбда-выражения. Язык LINQ и лямбда-выражения вносят совершенно новый акцент в программирование на C# и еще глубже подчеркивают его ведущую роль в непрекращающейся эволюции языков программирования.

Текущей является версия C# 4.0. Эта версия прочно опирается на три предыдущие основные версии C#, дополняя их целым рядом новых средств. Вероятно, самыми важными среди них являются именованные и необязательные аргументы. В частности, именованные аргументы позволяют связывать аргумент с параметром по имени. А необязательные аргументы дают возможность указывать для параметра используемый по умолчанию аргумент. Еще одним важным новым средством является тип dynamic, применяемый для объявления объектов, которые проверяются на соответствие типов во время выполнения, а не компиляции. Кроме того, ковариантность и контравариантность параметров типа поддерживается благодаря новому применению ключевых слов in и out. Тем, кто пользуется моделью COM вообще и прикладными интерфейсами Office Automation API в частности, существенно упрощен доступ к этим средствам. В целом, новые средства, внедренные в версии C# 4.0, способствуют дальнейшей рационализации программирования и повышают практичность самого языка C#.

Еще два важных средства, внедренных в версии 4.0 и непосредственно связанных с программированием на C#, предоставляются не самим языком, а средой .NET Framework 4.0. Речь идет о поддержке параллельного программирования с помощью библиотеки распараллеливания задач (TPL) и параллельном варианте языка интегрированных запросов (PLINQ). Оба эти средства позволяют существенно усовершенствовать и упростить процесс создания программ, в которых применяется принцип параллелизма. И то и другое средство упрощает создание многопоточного кода, который масштабируется автоматически для использования нескольких процессоров, доступных на компьютере. В настоящее время широкое распространение получили компьютеры с многоядерными процессорами, и поэтому возможность распараллеливать выполнение кода среди всех доступных процессоров приобретает все большее значение практически для всех, кто программирует на C#.

Основным понятием C# является объектно-ориентированное программирование (ООП). Методика ООП неотделима от C#, и поэтому все программы на C# являются объектно-ориентированными хотя бы в самой малой степени. В связи с этим очень важно и полезно усвоить основополагающие принципы ООП, прежде чем приступать к написанию самой простой программы на C#.

ООП представляет собой эффективный подход к программированию. Методики программирования претерпели существенные изменения с момента изобретения компьютера, постепенно приспосабливаясь, главным образом, к повышению сложности программ. Когда, например, появились первые ЭВМ, программирование заключалось в ручном переключении на разные двоичные машинные команды с переднего пульта управления ЭВМ. Такой подход был вполне оправданным, поскольку программы состояли всего из нескольких сотен команд. Дальнейшее усложнение программ привело к разработке языка ассемблера, который давал программистам возможность работать с более сложными программами, используя символическое представление отдельных машинных команд. Постоянное усложнение программ вызвало потребность в разработке и внедрении в практику программирования таких языков высокого уровня, как, например, FORTRAN и COBOL, которые предоставляли программистам больше средств для того, чтобы как-то справиться с постоянно растущей сложностью программ. Но как только возможности этих первых языков программирования были полностью исчерпаны, появились разработки языков структурного программирования, в том числе и C.

На каждом этапе развития программирования появлялись методы и инструментальные средства для "обуздания" растущей сложности программ. И на каждом таком этапе новый подход вбирал в себя все самое лучшее из предыдущих, знаменуя собой прогресс в программировании. Это же можно сказать и об ООП. До ООП многие проекты достигали (а иногда и превышали) предел, за которым структурный подход к программированию

оказывался уже неработоспособным. Поэтому для преодоления трудностей, связанных с усложнением программ, и возникла потребность в ООП.

ООП вобрало в себя все самые лучшие идеи структурного программирования, объединив их с рядом новых понятий. В итоге появился новый и лучший способ организации программ. В самом общем виде программа может быть организована одним из двух способов: вокруг кода (т.е. того, что фактически происходит) или же вокруг данных (т.е. того, что подвергается воздействию). Программы, созданные только методами структурного программирования, как правило, организованы вокруг кода. Такой подход можно рассматривать "как код, воздействующий на данные".

Совсем иначе работают объектно-ориентированные программы. Они организованы вокруг данных, исходя из главного принципа: "данные управляют доступом к коду". В объектно-ориентированном языке программирования определяются данные и код, которому разрешается воздействовать на эти данные. Следовательно, тип данных точно определяет операции, которые могут быть выполнены над данными.

Для поддержки принципов ООП все объектно-ориентированные языки программирования, в том числе и C#, должны обладать тремя общими свойствами: инкапсуляцией, полиморфизмом и наследованием. Рассмотрим каждое из этих свойств в отдельности.

Инкапсуляция — это механизм программирования, объединяющий вместе код и данные, которыми он манипулирует, исключая как вмешательство извне, так и неправильное использование данных. В объектно-ориентированном языке данные и код могут быть объединены в совершенно автономный черный ящик. Внутри такого ящика находятся все необходимые данные и код. Когда код и данные связываются вместе подобным образом, создается объект. Иными словами, объект — это элемент, поддерживающий инкапсуляцию.

В объекте код, данные или же и то и другое могут быть закрытыми или же открытыми. Закрытые данные или код известны и доступны только

остальной части объекта. Это означает, что закрытые данные или код недоступны части программы, находящейся за пределами объекта. Если же данные или код оказываются открытыми, то они доступны другим частям программы, хотя и определены внутри объекта. Как правило, открытые части объекта служат для организации управляемого интерфейса с закрытыми частями.

Основной единицей инкапсуляции в C# является класс, который определяет форму объекта. Он описывает данные, а также код, который будет ими оперировать. В C# описание класса служит для построения объектов, которые являются экземплярами класса. Следовательно, класс, по существу, представляет собой ряд схематических описаний способа построения объекта.

Код и данные, составляющие вместе класс, называют членами. Данные, определяемые классом, называют полями, или переменными экземпляра. А код, оперирующий данными, содержится в функциях-членах, самым типичным представителем которых является метод. В C# метод служит в качестве аналога подпрограммы. (К числу других функций-членов относятся свойства, события и конструкторы.) Таким образом, методы класса содержат код, воздействующий на поля, определяемые этим классом.

Полиморфизм, что по-гречески означает "множество форм", — это свойство, позволяющее одному интерфейсу получать доступ к общему классу действий. Простым примером полиморфизма может служить руль автомашины, который выполняет одни и те же функции своеобразного интерфейса независимо от вида применяемого механизма управления автомашиной. Это означает, что руль действует одинаково независимо от вида рулевого управления: прямого действия, с усилением или реечной передачей. Следовательно, при вращении руля влево автомашина всегда поворачивает влево, какой бы вид управления в ней ни применялся. Главное преимущество единообразного интерфейса заключается в том, что, зная, как обращаться с рулем, вы сумеете водить автомашину любого типа.

Тот же самый принцип может быть применен и в программировании. Рассмотрим для примера стек, т.е. область памяти, функционирующую по принципу "последним пришел — первым обслужен". Допустим, что в программе требуются три разных типа стеков: один — для целых значений, другой — для значений с плавающей точкой, третий — для символьных значений. В данном примере алгоритм, реализующий все эти стеки, остается неизменным, несмотря на то, что в них сохраняются разнотипные данные. В языке, не являющемся объектно-ориентированным, для этой цели пришлось бы создать три разных набора стековых подпрограмм с разными именами. Но благодаря полиморфизму для реализации всех трех типов стеков в C# достаточно создать лишь один общий набор подпрограмм. Зная, как пользоваться одним стеком, вы сумеете воспользоваться и остальными.

В более общем смысле понятие полиморфизма нередко выражается следующим образом: "один интерфейс — множество методов". Это означает, что для группы взаимосвязанных действий можно разработать общий интерфейс. Полиморфизм помогает упростить программу, позволяя использовать один и тот же интерфейс для описания общего класса действий. Выбрать конкретное действие (т.е. метод) в каждом отдельном случае — это задача компилятора. Программисту не нужно делать это самому. Ему достаточно запомнить и правильно использовать общий интерфейс.

Наследование представляет собой процесс, в ходе которого один объект приобретает свойства другого объекта. Это очень важный процесс, поскольку он обеспечивает принцип иерархической классификации. Если вдуматься, то большая часть знаний поддается систематизации благодаря иерархической классификации по нисходящей. Например, сорт яблок "Джонатан" входит в общую классификацию сортов яблок, которые, в свою очередь, относятся к классу фруктов, а те — к еще более крупному классу пищевых продуктов. Это означает, что класс пищевых продуктов обладает рядом свойств (съедобности, питательности и т.д.), которые по логике вещей распространяются и на его подкласс фруктов. Помимо этих свойств, класс

фруктов обладает своими собственными свойствами (сочностью, сладостью и т.д.), которыми он отличается от других пищевых продуктов. У класса яблок имеются свои характерные особенности (растут на деревьях, не в тропиках и т.д.). Таким образом, сорт яблок "Джонатан" наследует свойства всех предшествующих классов, обладая в то же время свойствами, присущими только этому сорту яблок, например красной окраской кожицы с желтым бочком и характерным ароматом и вкусом.

Если не пользоваться иерархиями, то для каждого объекта пришлось бы явно определять все его свойства. А если воспользоваться наследованием, то достаточно определить лишь те свойства, которые делают объект особенным в его классе. Он может также наследовать общие свойства своего родителя. Следовательно, благодаря механизму наследования один объект становится отдельным экземпляром более общего класса

Описание использованных програмних средств

Любая программа представляет собой последовательность инструкций в машинных кодах, которые управляют поведением определенного вычислительного устройства [4].

Структура программы

Все программы на языке C++ состоят из одной или нескольких функций. В любом случае программа должна содержать функцию main(), которая при выполнении программы вызывается первой. Определение функции main состоит из заголовка void main(), и последовательности инструкций, заключённых в фигурные скобки. Слово void в заголовке говорит о том, что функция не должна вырабатывать и возвращать значение. Все программы должны иметь функцию с именем main.

Общий вид программы на языке C++ показан в следующем примере. Функции с именами fun1(),…,funn() определяются пользователем [2]. Пример:

```
подключение заголовочных файлов
объявление глобальных переменных
тип_возвращаемого_значения main(список_параметров)
{
 последовательность операторов
}
тип_возвращаемого_значения fun1(список_параметров)
{
 последовательность операторов
}
.
.
.
тип_возвращаемого_значения funn(список_параметров)
{
 последовательность операторов
}
```

Заголовочные файлы

Любая программа на языке C++ содержит подключение заголовочных файлов с помощью директивы препроцессора #include<имя_заголовочного_файла.h>.

Директива #include вынуждает компилятор считать и подставить в исходный файл с заданным именем. Это имя заключается в двойные или угловые скобки.

Кавычки и угловые скобки, в которых указываются имена включаемых файлов, определяют способ их поиска на жестком диске. Если имя файла содержится в угловых скобках, он должен находиться в каталоге, указанном компилятором. Если имя файла заключено в кавычки, как правило, его поиск выполняется в рабочем каталоге. Если файл не найден, то поиск повторяется так, будто имя файла содержалось в угловых скобках.

Например:

```
#include <iostream.h>
#include <conio.h>
#include <iomanip.h>
```

В первом примере с помощью директивы #include подключают заголовок ввода-вывода <iostream.h>. Этот файл содержит информацию, необходимую, чтобы компилировать программу, которая использует cin и cout.

Во втором примере директивы #include подключают заголовок <iomanip.h>, который даёт указание препроцессору C++ включить в программу параметризованный манипулятор потока setw. Он предназначен для манипулирования выходными величинами на экране, например:

```
cout << setw(3)<<year
```

Обращение setw(3) определяет, что следующая выходная величина будет напечатана с шириной (размером) поля 3, т.е. её значение будет содержать по крайней мере 3 символьных позиции.

В третьем примере с помощью директивы #include подключают заголовок <conio.h>, который содержит функцию задержки экрана getch() [4].

Типы данных

В языке C++ существуют 5 типов данных: символ, целое число, число с плавающей запятой, число с плавающей запятой удвоенной точности и переменная, не имеющая значение. Им соответствуют следующие ключевые слова: char, int, float, double, void. Все другие типы данных в языке C++ создаются на основе элементарных типов, указанных выше. Размер переменных и диапазон их значений зависит от типа процессора и компилятора.

К пяти основным типам данных, определенных в языке C, язык C++ добавляет еще такой тип как bool.

Тип void используется для определения функции, не возвращающей никаких значений, либо для создания обобщенного указателя.

Переменные

Переменная – это имя ячейки памяти, которую можно использовать для хранения модифицируемого значения. Все переменные должны быть объявлены до своего использования.

Ниже приведен общий вид объявления переменной.

тип список_переменных;

Например:

int i,j;

При объявлении переменной ей можно присвоить начальное значение. Общий вид инициализации выглядит следующим образом:

тип имя_переменной = значение;

Например:

int a=0;

Глобальные и локальные статические переменные инициализируются только при запуске программы. Локальные переменные инициализируются каждый раз при входе в блок, где они описаны. Неинициализированные локальные переменные имеют неопределенное значение, пока к ним не будет применен оператор присваивания. Неинициализированные глобальные переменные и локальные статические переменные автоматически устанавливаются равными нулю [5].

Практически все символы можно вывести на печать, заключив их в одиночные кавычки. Однако некоторые символы, например, символ перехода на новую строку, невозможно ввести в строку с клавиатуры. Для этого в языке C/C+ + предусмотрены специальные управляющие символьные константы.

Идентификаторы

В языке C/C++ имена переменных, функций, меток и других объектов, определенных пользователем, называются идентификаторами. Идентификаторы могут состоять из одного или нескольких символов. Первый символ идентификатора должен быть буквой или символом подчеркивания, а следующие символы должны быть буквами, цифрами или символами подчеркивания.

В языке C++ нет ограничений на длину идентификаторов, и значащими считаются, по крайней мере, 1024 первых символа.

Символы, набранные в верхнем и нижнем регистре, различаются т. к. язык C/C++ чувствителен к регистру. Следовательно, imin Imin, IMIN – это различные идентификаторы. Переменная с именем, например, a_1 в языке C++

записывается как a1. В противном случае компилятор выдаст сообщение об ошибке.

Ключевые слова нельзя использовать в качестве идентификаторов и они не должны совпадать с именами функций из стандартных библиотек [2].

Операторы

В языке C/C++ предусмотрено большое количество операторов. В этих языках операторам уделяется гораздо больше внимания, чем в большинстве других языков. Операторы разделяются на 4 основные группы: арифметические, сравнения, логические и побитовые.

Оператор присваивания

Оператор присваивания можно использовать в любом корректном выражении. Общий вид оператора присваивания выглядит следующим образом:

имя_переменной = выражение

Здесь выражение может состоять как из отдельной константы, так и комбинации сложных операторов [4].

Арифметические операторы

Операторы +, -, * и / выполняются точно так же, как и в большинстве других языков программирования. Их можно применять практически к любым встроенным типам данных. Если оператор / применяется к целому числу или символу дробная часть отбрасывается.

Оператор %, как и в других языках, возвращает остаток целочисленного деления. Однако он не применим к числам с плавающей точкой.

Унарный минус умножает число на -1.

Оператор ++ добавляет 1 к своему операнду, а оператор – – вычитает ее.

Операторы сравнения и логические операторы имеют более низкий приоритет, чем арифметические операции.

Условные операторы

Оператор условного перехода имеет следующий формат записи:

if (A) оператор 1 ;

else оператор 2 ;

где A - выражение. Если значение этого выражения "истина" (не нуль), то выполняется оператор1, если же оно "ложь", то выполняется оператор2; в

случае, когда выражение ошибочное и отсутствует область else - выполняется следующий оператор [4]. Например:

```
if (a=1) b=a*2;
else b=0;
```

Операторы цикла

В языке C/C++ операторы цикла предназначены для выполнения повторяющихся инструкций, пока действует определенное правило. Это условие может быть как задано заранее(в цикле for), так и меняться во время выполнения цикла(в операторах while и_do-while)

В программе широко используется цикл for.

Оператор цикла for имеет следующую форму записи:

```
for ( [выражение1;] [выражение2;] [выражение3] ) оператор;
```

где выражение1 - выражение инициализации, которое обычно используется для установки начального значения; это выражение присваивания (необязательный параметр); выражение2 - выражение условия, которое определяет, при каком условии цикл будет повторяться (необязательный параметр); выражение3 - выражение итерации, которая определяет шаг изменения сменных, которые руководят циклом после каждого выполнения (необязательный параметр) [2].

Этот оператор реализуется таким образом:

- сначала выполняется выражение инициализации;
- вычисляется условное выражение;
- если результат условного выражения "истина" (не равняется нулю), то выполняется оператор цикла;
- вычисляется выражение итерации;
- снова проверяется условие;

- как только условное выражение становится равным нулю ("неправда"), управления передается оператору, который располагается за оператором цикла for.

Поскольку проверка условия производится перед циклом, то цикл может ни разу не выполняться, если условие сразу будет ошибочным.

Пример использования цикла for:

```
for(i=0;i<3;i++)

for(j=0;j<3;j++)

cin>>A[i][j];
```

Операторы ввода-вывода

При вводе-выводе потока все данные рассматриваются как поток отдельных байтов.

Чтобы использовать функции ввода-вывода необходимо подключить библиотеку ввода-вывода

```
#include <iostream.h>
```

Формат записи операторов ввода-вывода:

```
cin [ >> values ];

cout << data [ << data << "\n"] [5];
```

Например:

```
cin>>A[i][j];
cout<<"x1="<<x1<<endl;
```

Массивы и указатели

Массив – это последовательная группа ячеек памяти, которые имеют одинаковый тип и имеют общее имя. Доступ к отдельному элементу массива осуществляется с помощью индекса.

В языке C/C++ предусмотрены многомерные массивы. Простейшим из них является двумерный. Объявление двумерного массива A, состоящего из 3 строк и 3 столбцов выглядит следующим образом:

```
double A[3][3];
```

Обращение к элементу двумерного массива выглядит так:

A[0][0], A[0][1] и т.д.

В языке C/C++ указатели и массивы тесно связаны между собой. Как известно, имя массива без индекса – это указатель на его первый элемент [4]. Рассмотрим, например, следующий массив:

```
double A[3][3];
```

Следующие два выражения абсолютно идентичны:

```
A
&A[0][0]
```

Доступ к элементам массива может осуществляться следующим образом:

```
A[i][j];
*(*(A+i)+j);
```

Записи являются идентичными.

Функции

Функция – это процедура(подпрограмма), которая несёт законченную смысловую нагрузку.

Определение функции состоит из заголовка функции и тела функции, которое заключается в ( ) и несёт смысловую нагрузку.

Общий вид функции выглядит следующим образом:

тип_функции имя_функции(список_аргументов)
{ тело_функции };

Тип_функции – это тип значения, которое возвращает функция. Если функция не возвращает никакого значения, то ей тип всегда void. Возврат значения из функции происходит с использованием оператора return, который имеет следующий вид

return выражение

Операторов return может быть сколько угодно, но функция возвращает всегда только одно значение.

По умолчанию если тип функции не определён явно, то функция имеет тип int.

Список_аргументов может быть либо пустим, либо, например

float fun(int, float);

Тело_функции – это всегда блок или составной оператор, т.е. последовательность описаний и операторов, заключённых в { } [6].

Для того, чтобы функция выполнила некоторые действия, она должна быть вызвана. Вызов функции имеет вид

имя_функции(список_фактических_параметров);

Например:

Kramer(A,B);

Решение квадратного уравнения в C++

В этой статье напишем 2 **программы на языке C++ решение квадратного уравнения $ax^2+bx+c=0$**. Причем одна программа будет легче, в том плане, что если дискриминант будет меньше нуля, то программа выведет сообщение о том, что действительных корней нет. А вторая программа будет правильнее, в том что, будет учитываться и комплексный вариант решения, т.е. находить корни, когда дискриминант меньше нуля.

Рассмотрим для начала 1-й вариант.

Первое что нужно сделать, это определиться с входными параметрами. Входными параметрами будут коэффициенты a, b и c. А на выходе соответственно мы должны получить значения x_1 и x_2 или вывести сообщение о том, что действительных корней нет. Еще у нас будет вспомогательная переменная d, которая будет отвечать за дискриминант.

Для понимания того, как писать программу, предоставляю вам блок-схему написания программы:

Сначала осуществляем ввод коэффициентов. Затем высчитываем дискриминант. Дальше идет разветвление в зависимости от величины дискриминанта. Если D>0, то у него два вещественных корня, и мы их выводим, иначе если D=0, то у него оба корня вещественны и равны и выводим одно значение, ну а если D<0, то действительных корней в уравнении нет.

Ниже приведу 3 примера, как может выглядеть консольное окно в зависимости от дискриминанта:

Рекомендуемый вид экрана программы, когда уравнение имеет два действительных корня:

Заключение

В результате изучения новых способов решения квадратных уравнений мы получили возможность решать уравнения не только по формуле, но и более интересными способами.

Человеку, изучающему алгебру, часто полезнее решить одну и ту же задачу тремя различными способами, чем решить три-четыре различные задачи. Решая одну задачу различными методами, можно путем сравнений выяснить, какой из них короче и эффективнее. Так вырабатывается опыт.

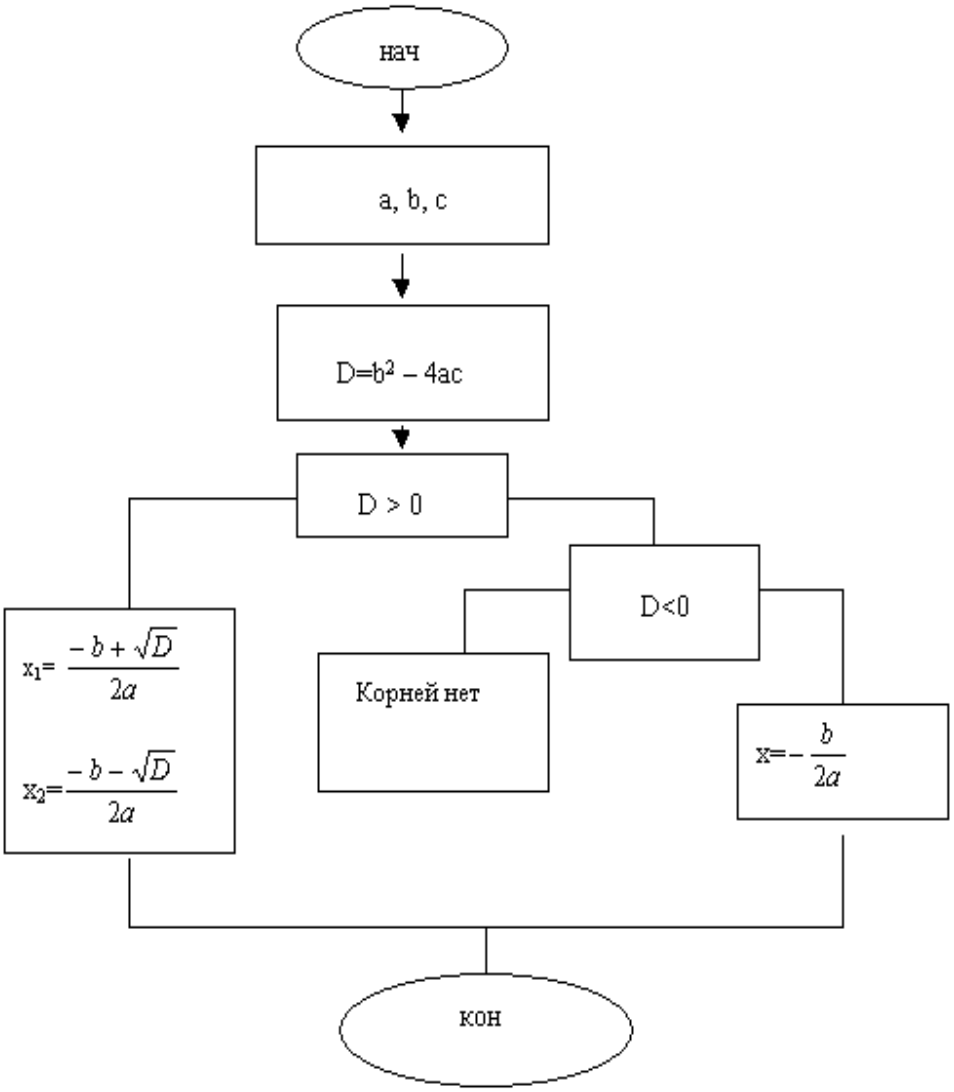
В ходе выполнения курсовой работы я закрепила знания работы с программой C++ . Благодаря уникальному сочетанию удобства разработки пользовательских интерфейсов, компонентной архитектуры, однотипности доступа к разнообразным базам данных было проще программировать.

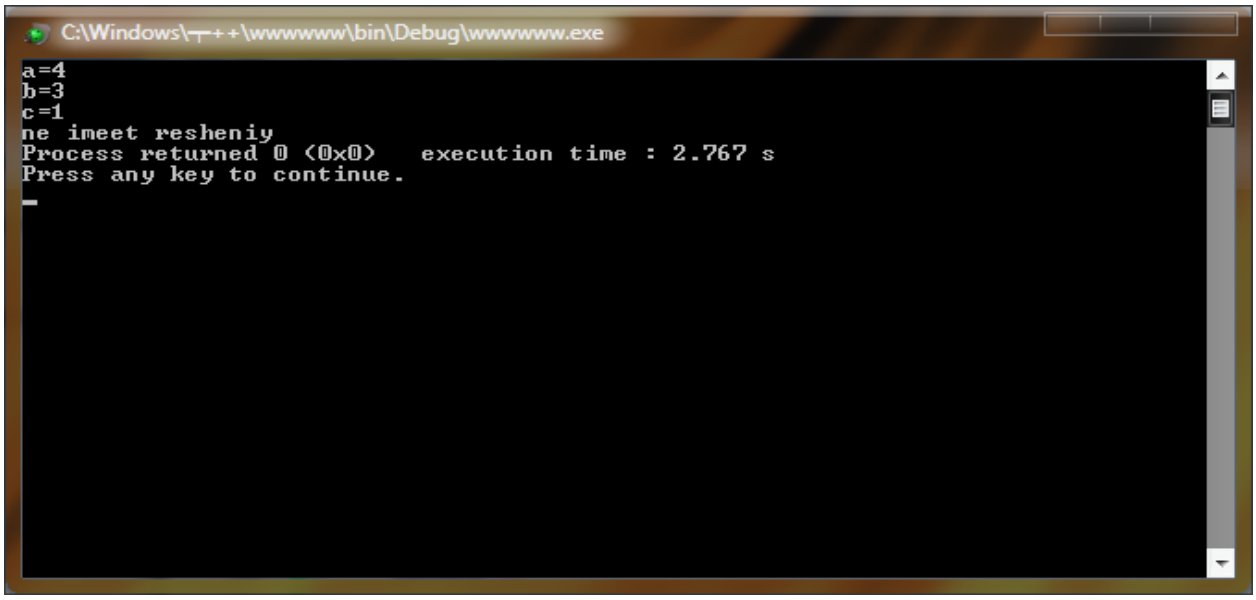
Прикладной работа

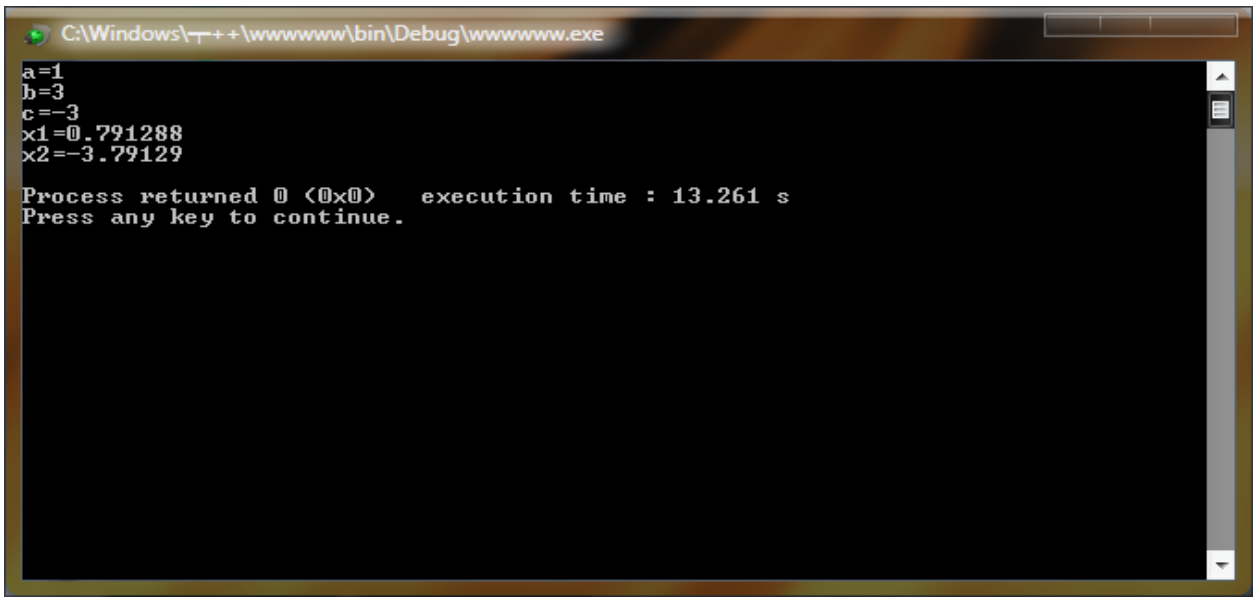
Решение квадратного уравнения: Программный код

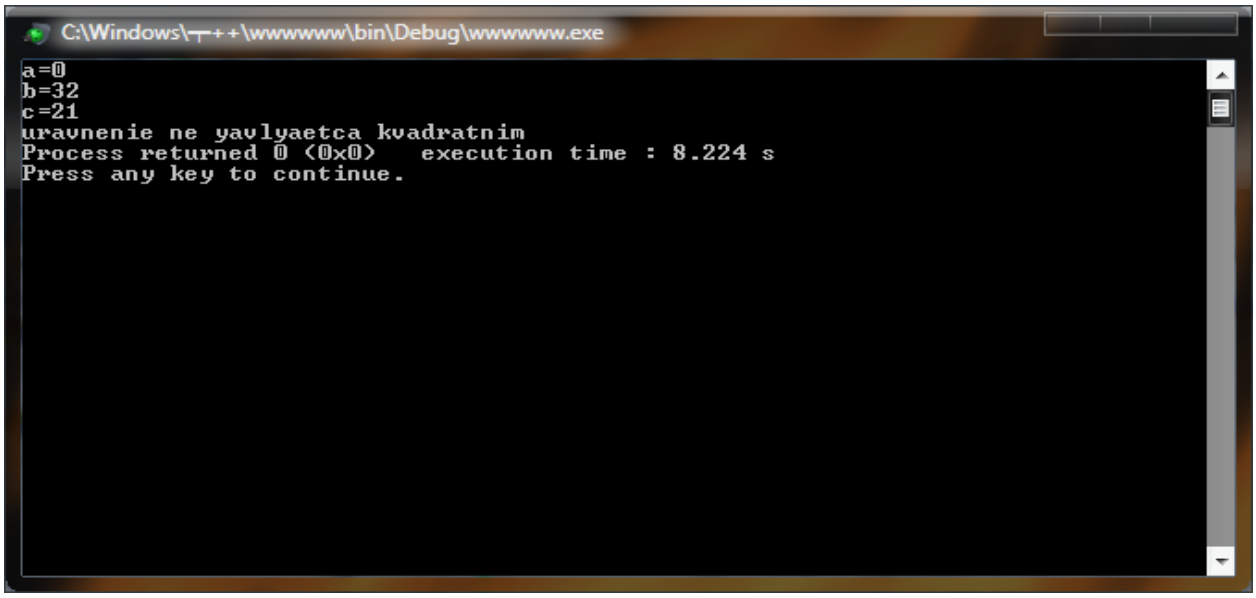
```
#include <iostream>
#include <cmath>
using namespace std;

int main() {
float a, b, c, d;  float x1, x2;
cout << "a=";  cin >> a;
cout << "b="; cin >> b;
cout << "c="; cin >> c;
if (a == 0) {
cout << "uravnenie ne yavlyaetca kvadratnim";
}
else {
d = b*b - 4 * a*c;
if (d < 0) {
cout << "ne imeet resheniy";
}
else {
x1 = (-b + sqrt(d)) / (2 * a);
x2 = (-b - sqrt(d)) / (2 * a);
cout << "x1=" << x1 << endl;
cout << "x2=" << x2 << endl;  }
}
return 0; }
```

Блок схема


нач
a, b, c
D=b² – 4ac
D > 0
D<0
$x_1 = \frac{-b+\sqrt{D}}{2a}$
$x_2 = \frac{-b-\sqrt{D}}{2a}$
Корней нет
$x = -\frac{b}{2a}$
кон


C:\Windows\┬++\wwwwww\bin\Debug\wwwwww.exe
a=4
b=3
c=1
ne imeet resheniy
Process returned 0 (0x0)   execution time : 2.767 s
Press any key to continue.


C:\Windows\┬++\wwwwww\bin\Debug\wwwwww.exe
a=1
b=3
c=-3
x1=0.791288
x2=-3.79129
Process returned 0 (0x0)   execution time : 13.261 s
Press any key to continue.


C:\Windows\┬++\wwwwww\bin\Debug\wwwwww.exe
a=0
b=32
c=21
uravnenie ne yavlyaetca kvadratnim
Process returned 0 (0x0)   execution time : 8.224 s
Press any key to continue.

Список литературы:

1.Глушаков С.В. и др. Язык программирования С++. —Харьков: Фолио, 2002. — 500 с.

2. Google.com

3.Allbest.ru