МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ
РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение
высшего образования
«КУРГАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Кафедра автоматизации производственных процессов

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

для самостоятельной работы по дисциплине
«Программирование и основы алгоритмизации»
для студентов специальностей 15.03.04 «Автоматизация технологических процессов и производств» и 27.03.04 «Управление в технических системах»

Курган 2016

Кафедра автоматизации производственных процессов

Дисциплина: «Программирование и основы алгоритмизации»

Составил: ст. преподаватель И.В. Скобелев

Утверждены на заседании кафедры «30» июня 2016 г.

Рекомендованы методическим советом университета
«17» декабря 2015 г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ 4
Алгоритмы основные понятия и определения 4
Алгоритмические языки 9
Операции и выражения 14
Реализация простейшей программы на VC++ 16
Графическое описание алгоритмов. 18
Операторы управления. Разветвляющиеся алгоритмы 20
Циклические алгоритмы. Операторы цикла 23
Реализация разветвляющихся алгоритмов, в лабораторной работе №3 26
Решение задач использующих изучаемые элементы графического интерфейса, массивы и структуры данных в лабораторной работе №4. 29
Работа с файлами в С (и в С++) 31
Список литературы 35

ВВЕДЕНИЕ

Задача курса программирования состоит в том, чтобы научить студента самостоятельно обучаться написанию программ в различных системах программирования, используя для этого информацию из различных источников и прежде всего Интернет. В данном курсе обучение программированию рассматривается на примере языка программирования VC++ и среды разработки программ Microsoft Visual Studio 6.0. Среда выдрана с учетом системных требований, так как позволяет проводить обучение практически на любом компьютере. При обучении основное внимание уделяется именно умению студента с самого начала создавать программы. Поэтому, в курсе главным являются лабораторные работы, которые подробно рассматриваются на лекциях.

Обучение предполагает, что студент проделывает лабораторную работу прежде чем перейти к следующему теоретическому материалу, без практического использования операторов невозможно понять их назначение. Вместе с тем студент получает методику для самостоятельного овладения любой другой системой программирования и любого другого языка программирования. Примеры, используемые в лабораторных работах, легко могут быть воспроизведены и в других языках программирования, и в других интегрированных средах разработки программ.

В результате обучения студент должен уметь написать простую программу в среде Microsoft Visual Studio 6.0. Полученные знания и умения студент демонстрирует в своей курсовой работе. Работа оформляется в виде программного продукта и в виде пояснительной записки, написанной в соответствии с ГОСТ. Работа демонстрируется и защищается студентом в конце курса.

Алгоритмы, - основные понятия и определения.

Алгоритмы

Любой человек ежедневно встречается с множеством задач от самых простых и хорошо известных до очень сложных. Для многих задач существуют определенные правила (инструкции, предписания), объясняющие исполнителю, как решать данную задачу. Эти правила человек может изучить заранее или сформулировать сам в процессе решения задачи. Чем точнее и понятнее будут описаны правила решения задач, тем быстрее человек овладеет ими и будет эффективнее их применять. **Источниками возникновения алгоритмов служат: наблюдение и эксперимент, научная теория, прошлый опыт и др.**

Решение многих задач человек может передавать техническим устройствам - ПК, автоматам, роботам и т.д. Применение технических устройств предъявляет очень строгие требования к точности описания правил и последователь-

ности выполнения действий. Поэтому разрабатываются специальные языки для четкого и строгого описания различных правил.

Алгоритмизация это раздел информатики, изучающий методы и приемы построений алгоритма, а также их свойства, т. е. алгоритмика задачи, построения модели и алгоритмизация.

Перед решением любой задачи с помощью персонального компьютера (ПК) выполняются следующие этапы: постановка этой задачи, построение сценария и алгоритмизация.

Алгоритмизация задачи - процесс разработки (проектирования) алгоритма решения задачи с помощью ПК на основе ее условия и требований к конечному результату.

На этапе постановки задачи описываются исходные данные и предпосылки, формируются правила начала и окончания решения задачи (достижения цели), т. е. разрабатывается информационная или эквивалентная ей математическая модель. Методом проб и ошибок ведется поиск метода решения задачи (метода вычислений, метода перебора вариантов, метода распознавания образов). На основании этого метода разрабатывается исходный алгоритм, реализация которого принципиально возможна с помощью ПК. При разработке исходного алгоритма и даже при выборе модели пользователь, т. е. человек, решающий конкретную задачу, должен иметь представление о математическом обеспечении ПК.

Алгоритм - понятное и точное предписание исполнителю совершить последовательность действий, направленных на достижение указанной цели или на решение поставленной задачи [1].

Алгоритм применительно к ПК - точное предписание, т.е. набор операций и правил их чередования, при помощи которого, начиная с некоторых исходных данных, можно решить задачу фиксированного типа. Команда алгоритма - предписание о выполнении отдельного законченного действия исполнителя.

Термин алгоритм происходит от имени узбекского ученого IX в. Аль-Хорезми, который в своем труде "Арифметический трактат", переведенном в XII в. с арабского на латынь, изложил правила арифметических действий над числами в позиционной десятичной системе счисления. Эти правила и называли алгоритмами. Таким образом, правила сложения, вычитания, деления, умножения чисел, правила преобразования алгебраических выражений, правила построения геометрических фигур, грамматические правила правописания слов и предложений - все это алгоритмы. Многие правила, инструкции, записанные в различных документах и представляющие собой подробнейшие указания, годные во всевозможных ситуациях, также можно отнести к алгоритмам.

Виды алгоритмов как логико-математических средств отражают также компоненты человеческой деятельности, а сами алгоритмы в зависимости от цели, начальных условий задачи, путей ее решения и определения действий исполнителя подразделяются на [3]:

- механические алгоритмы, или детерминированные, жесткие (например, алгоритм работы машины, двигателя и т. п.);

- гибкие алгоритмы, например стохастические, т. е. вероятностные и эв ристические.

Механический алгоритм задает определенные действия, обозначая их в единственной и достоверной последовательности, обеспечивая тем самым однозначный требуемый или искомый результат, если выполняются те условия процесса, задачи, для которых разработан алгоритм.

Вероятностный (стохастический) алгоритм дает программу решения задачи несколькими путями или способами, приводящими к вероятному достижению результата.

Эвристический алгоритм (от греческого слова "эврика" - "Я нашел") - это такой алгоритм, в котором достижение конечного результата программы действий однозначно не предопределено, так же как не обозначена вся последовательность действий, не выявлены все действия исполнителя. К эвристическим алгоритмам относят, например, инструкции и предписания.

В этих алгоритмах используются универсальные логические процедуры и способы принятия решений, основанные на аналогиях, ассоциациях и прошлом опыте решения схожих задач.

Эвристика (в переводе с греческого - отыскивают, открывают) - это совокупность специальных методов и приемов, позволяющих открыть новое, неизвестное, найти решение нетривиальной задачи.

Эвристика изучает продуктивное творческое мышление и на этой основе выявляет способы построения оптимальных направлений поиска решений задач, точные методы решения которых неизвестны.

Часто, для получения новых алгоритмов, используются уже существующие алгоритмы. Это осуществляется комбинированием уже известных алгоритмов или с помощью эквивалентных преобразований алгоритмов.

Алгоритмы называются эквивалентными, если результаты, получаемые с помощью этих алгоритмов для одних и тех же исходных данных, одинаковы.

Типичный пример эквивалентного преобразования алгоритмов - перевод с одного алгоритмического языка на другой.

В общем случае алгоритмизация вычислительного процесса включает следующие действия:

1) последовательную декомпозицию задачи, выделение автономных этапов вычислительного процесса и разбивку каждого этапа на отдельные шаги;
2) формальную запись содержания каждого этапа и/или шага;
3) определение общего порядка выполнения этапов и/или шагов;
4) проверку правильности алгоритма.

Последовательная декомпозиция предполагает разделение сложной задачи на совокупность более простых подзадач.

Часто начинающие программисты не уделяют этапу алгоритмизации достаточного внимания и даже пытаются его игнорировать. В результате процесс программирования сильно усложняется.

Значительно проще решать задачу постепенно, в два этапа (при этом сложность выполнения каждого отдельного этапа получается в несколько раз меньше сложности исходной задачи).

На первом этапе надо наметить общую стратегию решения задачи и составить соответствующий алгоритм. Причем для сложной задачи алгоритмизация выполняется постепенно. Сначала составляется укрупненная схема решения, а затем схемы работы отдельных блоков. Кроме того, при алгоритмизации одного и того же процесса можно использовать несколько способов записи (начиная с менее формализованных форм).

На втором этапе остается лишь выполнить кодирование (программирование), заменив формульно-словесные инструкции алгоритма операторами конкретного языка. Эта работа уже не связана с большим умственным напряжением. При несложных задачах для ее выполнения достаточно знать общие правила оформления программ, правила описания данных, основные операторы (ввода/вывода, обработки, управления).

Если бы мы знали алгоритмы решения всех задач, то их исполнение можно было бы поручить машине. Но оказалось, что не все задачи, которые нам хотелось бы решить, имеют алгоритмы решения. Задачи, в принципе не имеющие общего решения, называют алгоритмически неразрешимыми. К примеру, мы знаем, как решить любое квадратное алгебраическое уравнение, пользуясь одним и тем же алгоритмом. Похожие формулы существуют и для кубических уравнений и для уравнений четвертой степени. Но уже для уравнений степени выше четвертой таких формул нет и в принципе быть не может, хотя в частном случае отдельные уравнения можно решить. Есть и другие алгоритмически неразрешимые задачи, например задача о трисекции угла, о квадратуре круга и др.

Свойства алгоритмов

Алгоритмы обладают целым рядом свойств [1]: понятностью, дискретностью, точностью, результативностью, массовостью.

Свойства алгоритма - набор свойств, отличающих алгоритм от любых предписаний и обеспечивающих его автоматическое исполнение.

Понятность для исполнителя - содержание предписания о выполнении только таких действий, которые входят в систему команд исполнителя, т. е. алгоритм должен быть задан с помощью таких указаний, которые исполнитель (персональный компьютер, промышленный компьютер, контроллер, однокристальная микроЭВМ и др.) может воспринимать и выполнять по ним требуемые действия (операции).

Дискретность (прерывность, раздельность) - выполнение команд алгоритма последовательно, с точной фиксацией моментов окончания выполнения одной команды и начала выполнения следующей, т. е. алгоритм должен содержать последовательность указаний (команд), каждое из которых приводит к выполнению в исполнителе одного шага (действия).

Определенность - каждое правило алгоритма должно быть четким, однозначным. Благодаря этому свойству выполнение алгоритма носит механический характер и не требует никаких дополнительных указаний или сведений о решаемой задаче.

Результативность - либо завершение решения задачи после выполнения алгоритма, либо вывод о невозможности продолжения решения по какой-либо из причин, т. е. алгоритм должен обеспечивать возможность получения результата после конечного числа шагов.

Массовость - означает, что алгоритм решения задачи разрабатывается в общем виде, т.е. он должен быть применим для некоторого класса задач, различающихся лишь исходными данными. При этом исходные данные могут выбираться из некоторой области, которая называется областью применимости алгоритма.

Основные характеристики алгоритмов

Для решения одной и той же задачи как правило можно использовать различные алгоритмы. В связи с этим, возникает необходимость сравнивать их между собой, и для этого нужны определенные критерии качества алгоритмов.

Временные характеристики алгоритма определяют длительность решения или временную сложность [4].

Длительность решения часто выражается в единицах времени, но удобнее ее выражать через количество операций, так как количество операций не зависит от быстродействия конкретной машины.

Временной сложностью алгоритма называется зависимость времени счета, затрачиваемого на получение результатов от объема исходных данных.

Временная сложность позволяет определить наибольший размер задачи, которую можно решить с помощью данного алгоритма на ПК. Каждый алгоритм можно характеризовать функцией *J[n),* выражающей скорость роста объема вычислений при увеличении размерности задачи - *n.* Если эта зависимость имеет линейный или полиномиальный характер, то алгоритм считается "хорошим", если экспоненциальный - "плохим".

Для сложных задач эта характеристика имеет большое значение, т.к. ее изменение значительно сильнее влияет на время решения, чем изменение быстродействия ПК. Например, при зависимости $Дn) = 2^n$ увеличение производительности в 10 раз увеличивает размерность задачи, решаемой за то же время, всего на 15% [4].

Объемные характеристики алгоритма определяют его информационную сложность. Информационная сложность связана со сложностью описания, накопления и хранения исходных, промежуточных и результирующих данных при решении определенной задачи.

Объем текста алгоритма (программы) определяется количеством операторов, использованных для записи алгоритма.

Объем внутренней и внешней памяти необходимой для хранения данных и программ при использовании данного алгоритма определяется на основании расчетов или опытным путем. При недостатке памяти носителей информации используется сегментация программ.

Сложность структуры алгоритма определяется количеством маршрутов, по которым может реализовываться процесс вычислений и сложностью каждого маршрута.

Очевидно, что при выборе алгоритмов нужно учитывать не только их характеристики качества, но и способ реализации алгоритма. Например, многие итерационные алгоритмы удобны для ПК, но слишком трудоемки для человека. Тип используемой ПК также может влиять на выбор алгоритма (иногда имеет место и обратный вариант, когда сначала определяется алгоритм и лишь затем способ реализации).

Способы описания алгоритмов

Для строгого задания различных структур данных и алгоритмов их обработки требуется иметь такую систему формальных обозначений и правил, чтобы смысл всякого используемого предписания трактовался точно и однозначно. Соответствующие системы правил называют языками описаний.

К средствам описания алгоритмов относятся следующие основные способы их представления: словесный; графический; псевдокоды; программный. На практике используются также и другой способ описания: табличный (таблицы переключений (таблицы истинности); таблицы автоматов; циклограммы работы; таблицы решений).

Алгоритмические языки.

На практике в качестве исполнителей алгоритмов используются компьютеры или иные вычислительные устройства (однокристальные микроПК, промышленные компьютеры, технологические контроллеры и др.). Поэтому алгоритм, предназначенный для исполнения на компьютере, должен быть записан на понятном ему языке. И здесь на первый план выдвигается необходимость точной записи команд, не оставляющей места для произвольного толкования их исполнителем.

Следовательно, язык для записи алгоритмов должен быть формализован. Такой язык принято называть языком программирования, а запись алгоритма на этом языке - программой для компьютера.

К алгоритмическим языкам относят машинный язык (система команд), языки программирования.

Математическое обеспечение - средства, которые могут быть предоставлены пользователю для решения его задачи с помощью ПК. Оно включает в себя алгоритмическое обеспечение - методы и алгоритмы, модели решения задач, лингвистическое обеспечение - языки программирования, программное обеспечение - систему автоматизации программирования и информационное обеспечение - структуры данных и базы данных.

В настоящее время в мире существует несколько сотен реально используемых языков программирования. Для каждого есть своя область применения.

Любой алгоритм, как мы знаем, есть последовательность предписаний, выполнив которые можно за конечное число шагов перейти от исходных данных к

результату. В зависимости от степени детализации предписаний обычно определяется уровень языка программирования - чем меньше детализация, тем выше уровень языка.

По этому критерию можно выделить следующие уровни языков программирования: машинные; машинно-ориентированные (языки ассемблера); машинно-независимые (языки высокого уровня).

Машинные и машинно-ориентированные языки - это языки низкого уровня, требующие указания мелких деталей процесса обработки данных. Языки же высокого уровня имитируют естественные языки, используя некоторые слова разговорного языка и общепринятые математические символы. Эти языки более удобны для человека.

Языки высокого уровня делятся на:

- процедурные (алгоритмические) (Basic, Pascal, C и др.), которые предназначены для однозначного описания алгоритмов; для решения задачи процедурные языки требуют в той или иной форме явно выписать процедуру ее решения;
- логические (Prolog, Lisp и др.), которые ориентированы не на разработку алгоритма решения задачи, а на систематическое и формализованное описание задачи с тем, чтобы решение следовало из составленного описания;
- объектно-ориентированные (Object Pascal, C++, Java и др.), в основе которых лежит понятие объекта, сочетающего в себе данные и действия над ними. Программа на объектно-ориентированном языке, решая некоторую задачу, по сути, описывает часть мира, относящуюся к этой задаче. Описание действительности в форме системы взаимодействующих объектов естественнее, чем в форме взаимодействующих процедур.

Достоинства и недостатки машинных языков

Каждый компьютер имеет свой машинный язык, т. е. свою совокупность машинных команд, которая отличается количеством адресов в команде, назначением информации, задаваемой в адресах, набором операций, которые может выполнить машина, и др. При программировании на машинном языке программист может держать под своим контролем каждую команду и каждую ячейку памяти, использовать все возможности имеющихся машинных операций.

Но процесс написания программы на машинном языке очень трудоемкий и утомительный. Программа получается громоздкой, труднообозримой, ее трудно отлаживать, изменять и развивать.

Поэтому в случае, когда нужно иметь эффективную программу, в максимальной степени учитывающую специфику конкретного компьютера, вместо машинных языков используют близкие к ним машинно-ориентированные языки (ассемблер).

Основные преимущества алгоритмических языков программирования перед машинными и машинно-ориентированными языками: алфавит алгоритмического языка значительно шире алфавита машинного языка, что существенно повышает наглядность текста программы; набор операций, допустимых для использования, не зависит от набора машинных операций, а выбирается из соображений удобства формулирования алгоритмов решения задач определенного класса; формат предложений достаточно гибок и удобен для использования, что позволяет с

помощью одного предложения задать достаточно содержательный этап обработки данных; требуемые операции задаются с помощью общепринятых математических обозначений; данным в алгоритмических языках присваиваются индивидуальные имена, выбираемые программистом; в языке может быть предусмотрен значительно более широкий набор типов данных по сравнению с набором машинных типов данных.

Алгоритмические языки программирования в значительной мере являются машинно-независимыми. Они облегчают работу программиста и повышают надежность создаваемых программ.

Компоненты алгоритмического языка

Алгоритмический язык (как и любой другой язык) образуют три составляющие: алфавит, синтаксис и семантика.

Алфавит - это фиксированный для данного языка набор основных символов, т. е. "букв алфавита", из которых должен состоять любой текст на этом языке, - никакие другие символы в тексте не допускаются.

Синтаксис - это правила построения фраз, позволяющие определить, правильно или неправильно написана та или иная фраза. Точнее говоря, синтаксис языка представляет собой набор правил, устанавливающих, какие комбинации символов являются осмысленными предложениями на этом языке.

Семантика определяет смысловое значение предложений языка. Являясь системой правил истолкования отдельных языковых конструкций, семантика устанавливает, какие последовательности действий описываются теми или

иными фразами языка и в конечном итоге какой алгоритм определен данным текстом на алгоритмическом языке.

Понятия, используемые в алгоритмических языках

Каждое понятие алгоритмического языка подразумевает некоторую синтаксическую единицу (конструкцию) и определяемые ею свойства программных объектов или процесса обработки данных. Понятие языка определяется во взаимодействии синтаксических и семантических правил. Синтаксические правила показывают, как образуется данное понятие из других понятий и букв алфавита, а семантические правила определяют свойства данного понятия.

Основными понятиями в алгоритмических языках обычно являются следующие:

- Имена (идентификаторы) - употребляются для обозначения объектов программы (переменных, массивов, функций и др.).
- Операции бывают следующих типов:
- арифметические операции + ,-,-, / и др.;
- логические операции и, **или, не;**
- операции отношения <,>,<=,>=, = ,<>;
- операция сцепки (иначе присоединения, конкатенации) символьных значений друг с другом с образованием одной длинной строки; изображается знаком "+".

Данные - величины, обрабатываемые программой. Имеется четыре основных вида данных: константы, переменные, массивы и структуры данных

(списки, стеки и очереди, деревья и леса). Константы - это данные, которые зафиксированы в тексте программы и не изменяются в процессе ее выполнения.

Переменные обозначаются именами и могут изменять свои значения в ходе выполнения программы. Переменные бывают целые, вещественные, логические, символьные и литерные.

Массивы - последовательности однотипных элементов, число которых фиксировано и которым присвоено одно имя. Положение элемента в массиве однозначно определяется его индексами (одним - в случае одномерного массива или несколькими - если массив многомерный). Иногда массивы называют таблицами.

Линейный список представляет собой способ организации последователь

ности однотипных элементов, при котором каждый элемент, кроме первого, имеет одного предшественника (предыдущий элемент) и каждый элемент, кроме последнего, имеет одного преемника (следующий элемент). Доступ к каждому элементу списка можно получить, последовательно продвигаясь по списку от элемента к элементу. Другие типы списков, а также стеки, очереди, деревья и леса подробно рассмотрены в [5, 17].

Выражения предназначаются для выполнения необходимых вычислений, состоят из констант, переменных, указателей функций (например, sin(jc)), объединенных знаками операций. Выражения записываются в виде линейных последовательностей символов (без подстрочных и надстрочных символов, "многоэтажных" дробей и т. д.), что позволяет вводить их в компьютер, последовательно нажимая на соответствующие клавиши клавиатуры. Различают выражения арифметические, логические и строковые.

Арифметические выражения служат для определения одного числового значения. Например, (1 + sin(jc))/2. Значение этого выражения при jc = O равно 0.5, а при jc = я/2 - единице.

Логические выражения описывают некоторые условия, которые могут удовлетворяться или не удовлетворяться. Таким образом, логическое выражение может принимать только два значения - "истина" или "ложь" (да или нет).

Значения строковых (литерных) выражений - тексты. В них могут входить литерные константы, литерные переменные и литерные функции, разделенные знаком операции сцепки.

Операторы (команды). Оператор - это наиболее крупное и содержательное понятие языка: каждый оператор представляет собой законченную фразу языка и определяет некоторый, вполне законченный этап обработки данных. В состав операторов входят: ключевые слова; данные; выражения и т. д.

Операторы подразделяются на исполняемые и неисполняемые. Неисполняемые операторы предназначены для описания данных и структуры программы, а исполняемые - для выполнения различных действий (например, оператор присваивания, операторы ввода и вывода, условный оператор, операторы цикла, оператор процедуры и др.).

При решении различных задач с помощью компьютера бывает необходимо вычислить логарифм или модуль числа, синус угла и т. д. Вычисления часто употребляемых функций осуществляются посредством подпрограмм, называемых

стандартными функциями, которые заранее запрограммированы и встроены в транслятор языка. В качестве аргументов функций можно использовать константы, переменные и выражения. Каждый язык программирования имеет свой набор стандартных функций.

Арифметические выражения записываются по следующим правилам: 1. нельзя опускать знак умножения между множителями и ставить рядом два знака операций; 2. индексы элементов массивов записываются в квадратных (C++, Pascal) или круглых (Basic) скобках; 3. для обозначения переменных используются буквы латинского алфавита; 4. операции выполняются в порядке старшинства: сначала вычисление функций, затем возведение в степень, потом умножение и деление и в последнюю очередь сложение и вычитание. Операции одного старшинства выполняются слева направо.В записи логических выражений, помимо арифметических операций сложения, вычитания, умножения, деления и возведения в степень, используются операции отношения: < (меньше), <= (меньше или равно), > (больше), >= (больше или равно), = (равно), <> (не равно), а также логические операции - и, или, не.

Структуру программы на языке C++ рассмотрим на примере простой программы, печатающей строку текста. Листинг, характеризующий структуру построения простейшей программы на языке C++, приведен на рис. 4.1.

```
// Моя первая программа
# include < iostream .h >
main ( )
{
cout« " Это моя первая программа на языке C++";
```

Первая строка данной программы начинается с символа //, показывающего, что следующий за ним текст является комментарием, который игнорируется компилятором. Комментарии вставляются для документирования программы и облегчения ее чтения. Они помогают другим людям читать и понимать вашу программу. Комментарий, начинающийся с символа //, называется *однострочным,* потому что он должен заканчиваться в конце текущей строки. При использовании *многострочных комментариев* целесообразно применять символы /* и */. Все, что помещено между ними, игнорируется компилятором. Строка # include <iostream .h> является *директивой препроцессора.* Препроцессор - это специальная программа, которая обрабатывает строки программы, начинающиеся со знака #. Данная строка дает указание препроцессору перед компиляцией программы включить в нее информацию, содержащуюся в файле iostream.h. Следом идет обязательная функция main 0, а круглые скобки прямо указывают на то, что main - имя функции. Открывающая фигурная скобка отмечает начало последовательности операторов, образующих тело функции.

Строка cout « " "; - оператор вывода, с помощью которого выводится

на экран дисплея фраза, заключенная в кавычки. Функция может возвращать значение в программу с помощью оператора возврата (return). Этот оператор также означает выход из функции. Если же указанный оператор отсутствует, то функция автоматически возвращает значение типа void (пустой).

Закрывающая фигурная скобка отмечает конец последовательности операторов, образующих тело функции. На этой скобке выполнение функции и программы завершается . Программа на C++ состоит из одной или более функций, причем ровно одна из них обязательно должна называться main(). Функция - это блок программы, который выполняет одно или несколько действий. Описание функции состоит из заголовка и тела (см. рис.4.2). Круглые скобки являются частью имени функции, и ставить их надо обязательно, так как именно они указывают компилятору, что имеется в виду функция, а не просто английское слово main. Фактически каждая функция включает в свое имя круглые скобки, но в большинстве случаев в них содержится некая информация, передаваемая функции. Если же информация не передается, то в фигурных скобках можно указать ключевое слово void (пустой). Перед именем функции указывается ключевое слово, соответствующее типу возвращаемого функцией значения. Если значение не возвращается, то также можно указать ключевое слово void.

Операции и выражения.

Выражение в языке C++ (C) - это последовательность операндов, операций и символов-разделителей [2]. Операнды - это переменные, константы либо другие выражения. Разделителями в C++ являются символы [ ] ( ){},;:... * = #, каждый из которых выполняет свою функцию. Выражение может состоять из одной или более операций и определять выполнение целого ряда элементарных шагов по преобразованию информации. Компилятор соблюдает строгий порядок интерпретации выражений, называемый правилами предшествования. Этот порядок может быть изменен, если отдельные части выражения заключить в круглые скобки. Элементарная операция по преобразованию информации задается знаком операции.

По числу операндов, участвующих в операции, различают следующие типы:

- унарные (имеющие один операнд);
- бинарные (имеющие два операнда);
- тернарные (имеющие три операнда).

По типу выполняемой операции различают:

- арифметические операции;
- логические операции и операции отношения;
- операцию условия;
- операцию присваивания;
- операцию sizeof;
- операцию преобразования типов.

Арифметические операции

Язык C++ (C) включает [3]:

- арифметические операции сложения (задается знаком +), вычитания и унарный минус (-), умножения (*), деления (/), операцию определения остатка (%);
- операции инкремента (++) и декремента (--).

Операции сложения, вычитания, умножения и деления являются стандартными и выполняются так же, как и в большинстве других алгоритмических языках.

Операция определения остатка (%) (или деления по модулю) служит для определения остатка от деления числа целого типа, стоящего слева от знака, на целое число, расположенное справа от знака. Ее поясняет следующий пример: int x=7,y = 2, z;

z = x % y; // z=l - остаток от целого деления 7/3 z = x/y; // z =2 - результат деления целых чисел

Операции инкремента и декремента соответственно увеличивают и уменьшают операнд на 1. Операции ++ и ~ могут применяться только к переменным. Используются две формы их записи: префиксная (знак операции располагается слева от операнда) и постфиксная (знак операции справа от операнда). В префиксной форме сначала выполняется увеличение операнда на 1, и увеличенное значение используется в выражении. В постфиксной форме сначала берется значение операнда, и только после этого его значение увеличивается на 1. Например:

```
int x=0, y = 1, z=0;
z = x++;      // в результате  z = 0,x = 1
 z = ++x;     //  z=2, x =2
 z = ++y;     //  z=2, y= 2
```

Ниже приведен пример программы, иллюстрирующей работу префиксной и постфиксной форм инкремента.

```
# include < iostream .h> main{}
{
int x;
x=5;
cout« x <<"\n" ;
cout« x++ <<"\n" ; // Постфиксная форма инкремента
cout« x <<"\n" ;
x=5;
cout« x <<"\n" ;
cout« ++ x <<"\n" ; // Префиксная форма инкремента
cout« x <<"\n" ;
return 0;
```

Результаты работы программы: 5 5 6 5 6 6

Форма записи операций ++ и -- сказывается в составных выражениях. Например, три оператора присваивания вида number = number + 1; sum = sum + 1; x = x + 1;

могут быть записаны более кратко number += 1; sum + = 1; x += 1*

а с использованием операций инкремента в префиксной форме: ++number; ++sum ;

или в постфиксной форме:

```
number++;
sum++;
```

x

Логические операции и операции отношения

Логические операции и операции отношения используются при формировании логических выражений, имеющих только два значения: 1, если логическое выражение ИСТИННО; 0, если логическое выражение ЛОЖНО. Логические выражения наиболее часто используют вместе с операторами управления потоком вычислений - операторами циклов и ветвлений [2]. С++ поддерживает следующие логические операции:

&& - логическое И; дает результат 1 (ИСТИНА), если все операнды имеют значение 1 (ИСТИНА); в противном случае дает значение О (ЛОЖЬ);

|| - логическое ИЛИ; дает результат 1 (ИСТИНА), если хотя бы один из операндов имеет значение 1 (ИСТИНА); в противном случае дает значение О (ЛОЖЬ);

! - логическое НЕ; дает результат 1 (ИСТИНА), если операнд справа от знака имеет значение О (ЛОЖЬ); в противном случае дает значение 1 (ИС-ТИНА).

Возможные результаты выполнения логических операций представлены в

Таблица

a	b	a&&b	all b	!a
0	0	0	0	1
0	1	0	1	1
1	0	0	1	0
1	1	1	1	0

Реализация простейшей программы на VC++

с использованием графического интерфейса и библиотеки MFC (Windows Application на основе диалогового окна).

Рассмотрим пример написания простейшей – линейной программы на языке С++ . Данный пример отрабатывается в лабораторной работе № 2.

Создать пустой проект Console Application и добавляем в него файл с нашей программой.

```
#include "iostream.h"
void main()
{   float km,zen,ras,d1;
   cout<<"km =";
        cin >>km;
        cout<<"zen =";
        cin >>zen;
        cout<<"ras =";
        cin >>ras;
        d1=km*zen*ras/100;
        cout<<"Result =";
        cout<<d1<<"rub"<<"\n";

}
```

Это типичный линейный алгоритм, легко реализуемый консольным приложением. Так писали программы задолго до появления Windows.

Теперь напишем приложение отвечающее требованиям работы в Windows.

Создайте приложение с помощью MFC AppWizard (exe), с диалоговым окном в качестве главного. Добавляем в него три элемента Edit box и Static Text.для ввода данных, затем добавляем еще элемент типа Button и еще одно поле Edit box для вывода результата. Делаем надписи.

Чтобы изменить надпись элемента надо щелкнуть элемент правой кнопкой мыши и выбрать свойства в открывшемся окне изменить поле Caption. Для того, чтобы кириллица правильно отображалась в диалоговом окне надо в свойствах этого окна установить язык Russian.

Теперь связываем с каждым окошком переменные.

Чтобы связать переменную с элементом диалогового окна необходимо вызвать ClassWizard из меню View. Перейти на вкладку Member Variable , выделить требуемый идентификатор элемента и нажать кнопку Add Member Variable. В открывшемся окне набрать имя переменной и тип переменной.

Не забывайте, что тип у них float, потому, ЧТО ЭТО ЧИСЛА, КОТОРЫЕ МОГУТ БЫТЬ НЕ ЦЕЛЫМИ.

Потом привязываем к кнопке метод.

Чтобы связать метод с элементом диалогового окна, необходимо вызвать ClassWizard из меню View. Выбрать из списка имя соответствующего элемента а из списка Message соответствующее действие (например нажатие кнопки BN_CLICKED). И дважды щелкнете по нему. На предложение создать обработчик ответ OK.

Напишем код

```
void CL1v3Dlg::OnButton1()
{
        UpdateData(1);
   m_d1=m_km*m_ras*m_zen/100;

        UpdateData(0);
}
```

Для вычисления по нескольким поездкам модернизируем формулу

```
void CL1v3Dlg::OnButton1()
{
        UpdateData(1);
   m_d1=m_d1+m_km*m_ras*m_zen/100;

        UpdateData(0);
}
```

Начальные значения вводим кнопкой

```
void CL1v3Dlg::OnButton2()
{
        m_km=240;
```

```
  m_ras=8.5;
  m_zen=16.5;
UpdateData(0);
}
```

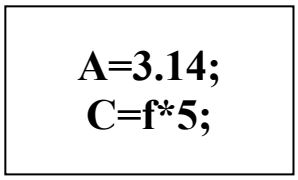
Для освоения материала лекции необходимо проработать лабораторные работы №1 и №2. Без этого дальнейший материал станет непонятен.

Графическое описание алгоритмов.

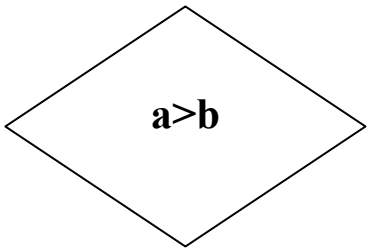
Правила составления блок-схем определяются ГОСТ 19.701-90 (ИСО 5807-85)

Наиболее часто используемые символы:

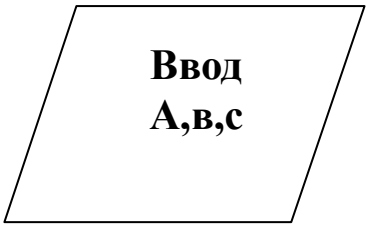
Процес – используется для обозначения последовательности вычислительных действий.



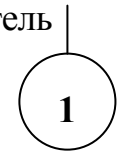
Решение – для проверки условий.



Ввод –вывод - обозначает ввод вывод в общем виде



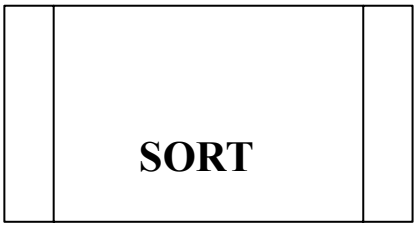
Соединитель



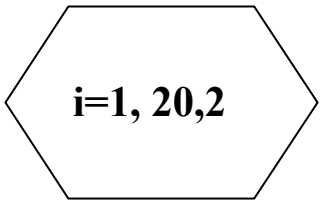
Пуск, останов

Печать

Предопределенный процесс или подпрограмма



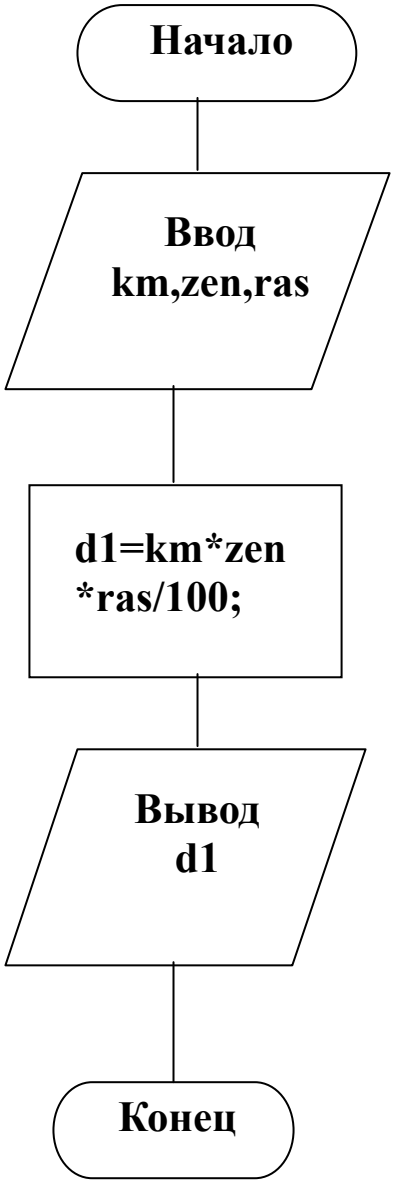
Модификация или начало цикла.



Использование графического метода описания алгоритмов рассмотрим на примере лабораторной работы №2.

Это типичный линейный алгоритм.

Блок-схема линейного алгоритма (решение задачи к лаб №2)



Операторы управления. Разветвляющиеся алгоритмы.

Оператор if.

Операторы управления вычислительным процессом позволяют выполнять ветвление, циклическое повторение одного или нескольких операторов, переда¬чу управления в нужное место кода программы. Под вычислительным процес¬сом понимают процесс выполнения операторов программы. Операторы про¬граммы могут быть простыми или составными. Простой оператор - это оператор, не содержащий другие операторы. Разделителем простых операторов служит точка с запятой. Специальным случаем простого оператора является пустой оператор, состоящий из единственного символа ';'. Составной опера¬тор, или блок, - это любая совокупность простых операторов, заключенная в фигурные скобки {}. Составной оператор идентичен простому оператору и мо¬жет находиться в любом месте программы, где синтаксис языка допускает на-личие оператора. Все операторы языка могут быть условно разделены на сле¬дующие категории:

- условные операторы, к которым относятся оператор условия if и оператор выбора switch;
- операторы цикла (for, while, do while);
- операторы перехода (break, continue, return, goto);
- другие операторы (оператор "выражение", пустой оператор).

Оператор if

Формат оператора:

if (выражение) оператор-1; [else оператор-2;]

Выполнение оператора if начинается с вычисления выражения.

Далее выполнение осуществляется по следующей схеме:

- если выражение истинно (т.е. отлично от 0), то выполняется оператор-1.
- если выражение ложно (т.е. равно 0),то выполняется оператор-2.
- если выражение ложно и отсутствует оператор-2 (в квадратные скобки заключена необязательная конструкция), то выполняется следующий за if оператор.

После выполнения оператора if значение передается на следующий оператор программы, если последовательность выполнения операторов программы не будет принудительно нарушена использованием операторов перехода.

Пример:
```
if (i < j)   i++;
else { j = i-3;   i++; }
```

Этот пример иллюстрирует также и тот факт, что на месте оператор-1, так же как и на месте оператор-2 могут находиться сложные конструкции.

Допускается использование вложенных операторов if. Оператор if может быть включен в конструкцию if или в конструкцию else другого оператора if. Чтобы сделать программу более читабельной, рекомендуется группировать операторы и конструкции во вложенных операторах if, используя фигурные скобки. Если же фигурные скобки опущены, то компилятор связывает каждое ключевое слово else с наиболее близким if, для которого нет else. Примеры:
```
int main ( )
 {        int t=2, b=7, r=3;
 if (t>b)        {        if (b < r)  r=b;        }        else r=t;
  return (0);      }
```

В результате выполнения этой программы r станет равным 2.

Если же в программе опустить фигурные скобки, стоящие после оператора if, то программа будет иметь следующий вид:
```
int main  ( )      {        int t=2,b=7,r=3;
  if ( t>b )        if ( b < c ) t=b;        else      r=t;
  return (0);      }
```

В этом случае r получит значение равное 3, так как ключевое слово else относится ко второму оператору if, который не выполняется, поскольку не выполняется условие, проверяемое в первом операторе if.

Следующий фрагмент иллюстрирует вложенные операторы if:
```
char ZNAC;    int x,y,z;
```

```
if (ZNAC == '-') x = y - z;
else  if (ZNAC == '+') x = y + z;
     else if (ZNAC == '*') x = y * z;
          else if (ZNAC == '/') x = y / z;
               else ...
```

Из рассмотрения этого примера можно сделать вывод, что конструкции использующие вложенные операторы if, являются довольно громоздкими и не всегда достаточно надежными. Другим способом организации выбора из множества различных вариантов является использование специального оператора выбора switch.

Оператор switch

Оператор switch предназначен для организации выбора из множества различных вариантов. Формат оператора следующий:

```
switch ( выражение )    {   [объявление]          :
     [ case  константное-выражение1]: [ список-операторов1]
     [ case  константное-выражение2]: [ список-операторов2]
       :
     [ default: [ список операторов ]]    }
```

Выражение, следующее за ключевым словом switch в круглых скобках, может быть любым выражением, допустимыми в языке СИ, значение которого должно быть целым. Отметим, что можно использовать явное приведение к целому типу, однако необходимо помнить о тех ограничениях и рекомендациях, о которых говорилось выше.

Значение этого выражения является ключевым для выбора из нескольких вариантов. Тело оператора smitch состоит из нескольких операторов, помеченных ключевым словом case с последующим константным-выражением. Следует отметить, что использование целого константного выражения является существенным недостатком, присущим рассмотренному оператору.

Так как константное выражение вычисляется во время трансляции, оно не может содержать переменные или вызовы функций. Обычно в качестве константного выражения используются целые или символьные константы.

Все константные выражения в операторе switch должны быть уникальны. Кроме операторов, помеченных ключевым словом case, может быть, но обязательно один, фрагмент помеченный ключевым словом default.

Список операторов может быть пустым, либо содержать один или более операторов. Причем в операторе switch не требуется заключать последовательность операторов в фигурные скобки.

Отметим также, что в операторе switch можно использовать свои локальные переменные, объявления которых находятся перед первым ключевым словом case, однако в объявлениях не должна использоваться инициализация.

Схема выполнения оператора switch следующая:

- вычисляется выражение в круглых скобках;
- вычисленные значения последовательно сравниваются с константными выражениями, следующими за ключевыми словами case;

- если одно из константных выражений совпадает со значением выражения, то управление передается на оператор, помеченный соответствующим ключевым словом case;

- если ни одно из константных выражений не равно выражению, то управление передается на оператор, помеченный ключевым словом default, а в случае его отсутствия управление передается на следующий после switch оператор.

Отметим интересную особенность использования оператора switch: конструкция со словом default может быть не последней в теле оператора switch. Ключевые слова case и default в теле оператора switch существенны только при начальной проверке, когда определяется начальная точка выполнения тела оператора switch. Все операторы, между начальным оператором и концом тела, выполняются вне зависимости от ключевых слов, если только какой-то из операторов не передаст управления из тела оператора switch. Таким образом, программист должен сам позаботится о выходе из case, если это необходимо. Чаще всего для этого используется оператор break.

Для того, чтобы выполнить одни и те же действия для различных значений выражения, можно пометить один и тот же оператор несколькими ключевыми словами case.

Выполнение оператора switch начинается с оператора, помеченного case 2. Таким образом, переменная i получает значение, равное 6, далее выполняется оператор, помеченный ключевым словом case 0, а затем case 4, переменная i примет значение 3, а затем значение -2. Оператор, помеченный ключевым словом default, не изменяет значения переменной.

Рассмотрим ранее приведенный пример, в котором иллюстрировалось использование вложенных операторов if, переписанной теперь с использованием оператора switch.

```
char ZNAC;     int x,y,z;
switch (ZNAC)     {
    case '+':  x = y + z;    break;
    case '-':  x = y - z;    break;
    case '*':  x = y * z;    break;
    case '/':  x = u / z;    break;
    default : ;        }
```

Использование оператора break позволяет в необходимый момент прервать последовательность выполняемых операторов в теле оператора switch, путем передачи управления оператору, следующему за switch.

Отметим, что в теле оператора switch можно использовать вложенные операторы switch, при этом в ключевых словах case можно использовать одинаковые константные выражения.

Циклические алгоритмы. Операторы цикла.

Оператор for

Оператор for - это наиболее общий способ организации цикла. Он имеет следующий формат:

for (выражение 1 ; выражение 2 ; выражение 3) тело

Выражение 1 обычно используется для установления начального значения переменных, управляющих циклом. Выражение 2 - это выражение, определяющее условие, при котором тело цикла будет выполняться. Выражение 3 определяет изменение переменных, управляющих циклом после каждого выполнения тела цикла.

Схема выполнения оператора for:

1. Вычисляется выражение 1.

2. Вычисляется выражение 2.

3. Если значения выражения 2 отлично от нуля (истина), выполняется тело цикла, вычисляется выражение 3 и осуществляется переход к пункту 2, если выражение 2 равно нулю (ложь), то управление передается на оператор, следующий за оператором for.

Существенно то, что проверка условия всегда выполняется в начале цикла. Это значит, что тело цикла может ни разу не выполниться, если условие выполнения сразу будет ложным.

Пример:
```
int main()
{ int i,b;
  for (i=1; i<10; i++)
    b=i*i;        return 0;      }
```

В этом примере вычисляются квадраты чисел от 1 до 9.

Некоторые варианты использования оператора for повышают его гибкость за счет возможности использования нескольких переменных, управляющих циклом.

Пример:
```
# include "iostream.h"
void main()
{ int top,  bot; char string[10]="Hello126 ", string1[10] ;
        for ( top=0, bot=9; top <= bot ; top++, bot--)
        {  string1[bot]=string[top];
                string1[top]=string[bot];        }
           cout<<"String =";
           for (int i=0;i<10; i++)
           {cout<<string1[i]; }
    cout<<'\n';}          }
```

В этом примере, реализующем запись строки символов в обратном порядке, для управления циклом используются две переменные top и bot. Отметим, что на месте выражение 1 и выражение 3 здесь используются несколько выражений, записанных через запятую, и выполняемых последовательно.

Другим вариантом использования оператора for является бесконечный цикл. Для организации такого цикла можно использовать пустое условное выражение, а для выхода из цикла обычно используют дополнительное условие и оператор break.

Пример:
```
for (;;)      { ...        ...  break;       ...      }
```

Так как согласно синтаксису языка Си оператор может быть пустым, тело оператора for также может быть пустым. Такая форма оператора может быть использована для организации поиска.

Пример: int t[5]= {3; 5; 12; 6; 15;}

for (int i=0; t[i]<10 ; i++) ;

В данном примере переменная цикла i принимает значение номера первого элемента массива t, значение которого больше 10.

Оператор while

Оператор цикла while называется циклом с предусловием и имеет следующий формат:

while (выражение) тело ;

В качестве выражения допускается использовать любое выражение языка Си, а в качестве тела любой оператор, в том числе пустой или составной. Схема выполнения оператора while следующая:

1. Вычисляется выражение.

2. Если выражение ложно, то выполнение оператора while заканчивается и выполняется следующий по порядку оператор. Если выражение истинно, то выполняется тело оператора while.

3. Процесс повторяется с пункта 1.

Оператор цикла вида

for (выражение-1; выражение-2; выражение-3) тело ;

может быть заменен оператором while следующим образом:

```
выражение-1;
while (выражение-2)        {  тело
  выражение-3;        }
```

Так же как и при выполнении оператора for, в операторе while вначале происходит проверка условия. Поэтому оператор while удобно использовать в ситуациях, когда тело оператора не всегда нужно выполнять.

Внутри операторов for и while можно использовать локальные переменные, которые должны быть объявлены с определением соответствующих типов.

Оператор do while

Оператор цикла do while называется оператором цикла с постусловием и используется в тех случаях, когда необходимо выполнить тело цикла хотя бы один раз. Формат оператора имеет следующий вид:

do тело while (выражение);

Схема выполнения оператора do while :

1. Выполняется тело цикла (которое может быть составным оператором).

2. Вычисляется выражение.

3. Если выражение ложно, то выполнение оператора do while заканчивается и выполняется следующий по порядку оператор. Если выражение истинно, то выполнение оператора продолжается с пункта 1.

Чтобы прервать выполнение цикла до того, как условие станет ложным, можно использовать оператор break.

Операторы while и do while могут быть вложенными.

Оператор continue

Оператор continue, как и оператор break, используется только внутри операторов цикла, но в отличие от него выполнение программы продолжается не с оператора, следующего за прерванным оператором, а с начала прерванного оператора. Пример:

```
int main()
{   int a,b;
    for (a=1,b=0; a<100; b+=a,a++)
    {  if (b%2) continue;
       ...   /* обработка четных сумм  */          }
    return 0;      }
```

Когда сумма чисел от 1 до а становится нечетной, оператор continue передает управление на очередную итерацию цикла for, не выполняя операторы обработки четных сумм.

Оператор continue, как и оператор break, прерывает самый внутренний из объемлющих его циклов.

Реализация разветвляющихся алгоритмов, в лабораторной работе №3.

Решаем задачу.

В гараже имеется 4 различных автомобиля (Ваз, Газель, Газ-66, Мерседес), каждый автомобиль имеет свой расход топлива на 100 км пути, а также свою стоимость топлива. Программа должна вычислять стоимость поездки для заданной машины

Строим приложение на основе диалогового окна как и во второй работе.

Добавляем необходимое количество полей для отображения Цены бензина, расхода, Грузоподъемности, расстояния, и результата, все как в предыдущей работе в ClassWizard делаем переменные связанные с окнами.

У нас получаются переменные.

```
float     m_zena;   цена литра
float     m_ras;    расход на 100 км
float     m_g;    грузоподъемность
float     m_km;    пройденные километры
float     m_Gr;    вес привозимого груза
float     m_rub;      стоимость перевозки.
```

Заводим кроме того служебную перемену и структуру для хранения данных об автомобиле.

```
class CL3v1Dlg : public CDialog
{// Construction
public:
int ki;   - индекс.
        struct a1     структура для хранения данных об автомобиле.
        { float ras,zena,g;
        } afto[4];    - у нас 4 автомаобиля.
```

Присваиваем начальные значения.

```
BOOL CL3v1Dlg::OnInitDialog()
{
```

```
        CDialog::OnInitDialog();
ki=0;
        // BAS
afto[0].ras =8.5;
afto[0].zena =18;
afto[0].g =400;
// Gasel
afto[1].ras =20;
afto[1].zena =10;
afto[1].g =1500;
// Gas66
afto[2].ras =30;
afto[2].zena =14;
afto[2].g =3000;
// mers
afto[3].ras =5;
afto[3].zena =19;
afto[3].g =300;

        // IDM_ABOU
```

Вызываем ClassWizard и делаем переменные для Check box.

Получаем

```
        BOOL    m_c1;
        BOOL    m_c2;
        BOOL    m_c3;
        BOOL    m_c4;
```

Делаем метод для кнопки вычислить

```
void CL3v1Dlg::OnButton1()
{       UpdateData(1);
        m_rub=(m_ras*m_zena*m_km/100)*ceil(m_Gr/m_g);
        UpdateData(0);}
```

Здесь мы используем функцию **ceil** это стандартная функция для нахождения наименьшего целого большего дроби.

Для ее включения надо добавить в начало файла

```
#include "math.h"
```

Делаем методы для каждого флажка

```
 void CL3v1Dlg::OnCheck1()
{ m_c1=1;
  m_c2=0;
  m_c3=0;
  m_c4=0;
ki=0;
m_ras=afto[ki].ras;
m_g=afto[ki].g;
m_zena=afto[ki].zena;
```

```
UpdateData(0);}
void CL3v1Dlg::OnCheck2()
{
m_c1=0;
  m_c2=1;
  m_c3=0;
  m_c4=0;
ki=1;
m_ras=afto[ki].ras;
m_g=afto[ki].g;
m_zena=afto[ki].zena;
UpdateData(0);}

void CL3v1Dlg::OnCheck3()
{m_c1=0;
  m_c2=0;
  m_c3=1;
  m_c4=0;
        ki=2;
m_ras=afto[ki].ras;
m_g=afto[ki].g;
m_zena=afto[ki].zena;
UpdateData(0);}

void CL3v1Dlg::OnCheck4()
{m_c1=0;
  m_c2=0;
  m_c3=0;
  m_c4=1;
ki=3;
m_ras=afto[ki].ras;
m_g=afto[ki].g;
m_zena=afto[ki].zena;
UpdateData(0);}
```

Дальше делаем кнопку Запомнить и метод к ней

```
void CL3v1Dlg::OnButton2()
{UpdateData(1);
        afto[ki].ras =m_ras;
afto[ki].zena =m_zena;
afto[ki].g =m_g;}
```

Если использовать Radio Button то алгоритм получится еще проще.

Получили типичный разветвляющийся алгоритм.

Решение задач использующих изучаемые элементы графического интерфейса, массивы и структуры данных в лабораторной работе №4.

Усложняем задачу, теперь у нас в гараже некоторое заранее неизвестное количество машин, каждая имеет свой расход и стоимость бензина. В добавок машины могут приобретаться новые и списываться старые, нам надо все это предусмотреть в программе.

Для решения задачи я буду использовать массив структур

int ki; - переменная счетчик.

```
struct a1 {   CString Marka;
        float r,z,g;
} Afto[10];
```

сделаю окошки с переменными как в лабораторной работе №3 и кроме того добавлю поле списка (**CListBox** имя - **m_List**)

```
CListBox    m_List;
        CString      m_marka;
        float   m_r;
        float   m_z;
        float   m_g;
        float   m_km;
        float   m_Gr;
        float   m_res;
```

Для добавления в список запишу создам кнопку и запишу :

```
void CL4v1Dlg::OnButton1()
{UpdateData(1);
        Afto[ki].Marka=m_marka;
        Afto[ki].r=m_r;
        Afto[ki].z=m_z;
        Afto[ki].g=m_g;
        ki=ki+1;
        m_List.AddString(m_marka);
UpdateData(0);}
```

Для удаления:

```
void CL4v1Dlg::OnButton2()
{m_List.DeleteString(m_List.GetCurSel());
        ki=ki-1;}
```

Для вычисления

```
void CL4v1Dlg::OnButton3()
{UpdateData(1);
m_res=ceil(m_Gr/m_g)*(m_r*m_z*m_km/100);
        UpdateData(0);}
```

Для отображения при выделении строки в списке

Делаем метод для события LBN_Selchange

```
void CL4v1Dlg::OnSelchangeList1()
{UpdateData(1);
```

```
        m_r=Afto[m_List.GetCurSel()].r;
        m_z=Afto[m_List.GetCurSel()].z;
        m_g=Afto[m_List.GetCurSel()].g;
        m_marka=Afto[m_List.GetCurSel()].Marka;
UpdateData(0);}
```

Теперь рассмотрим задачу оптимизации.

```
void CL4v1Dlg::OnButton4()
{int l,kxod;
   float min,R;
        l=0;
        min=99999.0f;
        UpdateData(1);
        for (int i=0;i<ki; i++)     {
      kxod=ceil(m_Gr/Afto[i].g);
              R=kxod*Afto[i].r*Afto[i].z*m_km/100;
      if (min>R)
              { min=R;
              l=i;}   }
  m_r=Afto[l].r;
        m_z=Afto[l].z;
        m_g=Afto[l].g;
   m_res=min;
        m_marka=Afto[l].Marka;
UpdateData(0);}
```

В результате у нас в полях будет выведен оптимальный по стоимости перевозки вариант.

Для случая с CComboBox

Можно обойтись без поля марка, а добавить еще одну переменную к окну **ComboBox** например

```
        CString      m_marka1;
        CComboBox        m_l1;
```

Теперь две переменные связаны с окно **ComboBox.**

Дольше все немного модифицируем

void CCombo2Dlg::OnButton1() - кнопка добавить

```
{       UpdateData(1);
Afto[ki].Marka=m_marka1;
        Afto[ki].r=m_r;
        Afto[ki].z=m_z;
        Afto[ki].g=m_g;
        ki=ki+1;
        m_l1.AddString(m_marka1);
        UpdateData(0);}
```

Кнопка удалить

void CCombo2Dlg::OnButton2()

```
{m_l1.DeleteString(m_l1.GetCurSel());
            ki=ki-1;
        m_marka1="";
UpdateData(0);}
```

И метод реакции на выбор из списка

```
void CCombo2Dlg::OnSelchangeCombo1()
{UpdateData(1);
        m_r=Afto[m_l1.GetCurSel()].r;
        m_z=Afto[m_l1.GetCurSel()].z;
        m_g=Afto[m_l1.GetCurSel()].g;
        m_marka1=Afto[m_l1.GetCurSel()].Marka;
UpdateData(0);    }
```

Работа с файлами в С (и в С++).

Язык С не содержит спец. встроенных средств для работы с файлами. Однако для решения практических задач очень важно уметь использовать внешние файлы. Структура файла также в языке ни как не определена. В общем случае файл или поток данных – это последовательность данных, как их интерпретировать решает программист.

Существует множество функций а также классов для работы с файлами. Условно можн разделить функции оперирующие с файлами как :

1. Потоками.
2. С файлами на нижнем уровне, байтами..
3. С консолями и портами.

Работа с файлами как с потоками

Ввод-вывод потока позволяет

1. Открывать и закрывать потоки.
2. Читать и записывать символ.
3. Читать и записывать данные.
4. Читать и записывать строки.
5. Читать и записывать форматированные данные любого вида.
6. Анализировать ошибки ввода- выцвода а также достижение конца потока.

Сначала файл надо объявить:

FILE *< имя>

Потом открыть

<.Имя >=fopen("имя на диске","вид доступа");

Например:

```
FILE *file;
file = fopen(“ d1.txt" , "w" );
```

Режимы доступа к файлу:

r – чтение существующего файла.

w- запись в файл.

a- добавление в конец файла, если файла нет он создается.

r+ -чтение и запись в файл. Файл должен существовать.

w+- чтение и запись в файл если файл существует то его содержимое уничтожается.

a+ - чтение и добавление в файл.

Могут быть добавлены так же спецификаторы потока

t – текстовый.

b- битовый.

Текстовый аналогичен файлу создаваемому в блокноте.

В двоичном нет разграничений строк. Все символы для двоичного потока равнозначны. По умолчанию принимается текстовый файл.

Системе все равно как файл был создан. Как при открытии указана так файл и интерпретируется.

После работы файл должен быть закрыт функцией fclose(a1);

Для форматного вывода может быть использована функция.

fprintf(Имя файла,"список форматов",список вводимых значений);

некоторые форматы

%s – символьная строка или символьный массив char[20]

%d – десятичное целое. int

%c – отдельный символ.

%f – величина типа float

По смотреть другие возможные форматы. В учебнике стр 46.-55.

```
char Str1[40];
int k;
float r1;
fprintf(a1,"%s %d %f %c",Str1,k,r1,'\n');
```

Форматный ввод может быть осуществлен функцией.

fscanf (Имя файла,"список форматов",список вводимых значений);

```
char Str1[40];
int k;
float r1;
fscanf (a1,"%s %d %f ",Str1,&k,&r1);
```

Конец файла может быть обнаружен с помощью функции:

feof(имя файла)

функция возвращает значение отличное от нуля если достигнут конец файла.

Использованию данных функций посвящена лабораторная работа № 5.

Практическая работа с файлами в лабораторной работе №5.

Большим недостатком нашей программы, которую мы писали во всех лабораторных работах, является то обстоятельства, что данные в ней не сохраняются при закрытии программы, и каждый раз их приходится снова набирать. Чтобы устранить этот недостаток, используем простейшие файловые функции для сохранения результатов работы программы на диске.

Если в лабораторной работе объявить файл.

FILE *uk;

То сбросить в него данные можно следующим образом

```
void CLab4Dlg::OnButton5()
{
uk=fopen("f1.txt","w");
for (int i=0;i<ki; i++)
{
fprintf(uk,"%s %f %f %f %c",Afto[i].Marka,Afto[i].r,Afto[i].z,Afto[i].g,'\n');
}
fclose(uk);
}
```

А получить данные в структуру можно

```
void CLab4Dlg::OnButton6()
{ char am[50];

        uk=fopen("f1.txt","r");

 while (!feof(uk))
 { fscanf(uk,"%s %f %f %f",am,&Afto[ki].r,&Afto[ki].z,&Afto[ki].g);
        if (feof(uk)) break;
    Afto[ki].Marka=am;
  m_List.AddString(am);
        ki=ki+1;
        };

  fclose(uk);
UpdateData(0);
}
```

Если надо организовать стандартный диалог для выбора файла то используется следующая конструкция.

```
void CLab4Dlg::OnButton5()
{ CString Name1;
        CFileDialog
fileDialog(TRUE,NULL,"*.txt",NULL,"(*.*)|*.*|(*.txt)|*.txt|");
int result = fileDialog.DoModal();          вызов метода и получение результата
 if (result==IDOK)                          проверка результата.
 {      AfxMessageBox(fileDialog.GetPathName()); на всякий случай вывод результата

 Name1=fileDialog.GetPathName();
uk=fopen(Name1,"w");
for (int i=0;i<ki; i++)
{
fprintf(uk,"%s %f %f %f %c",Afto[i].Marka,Afto[i].r,Afto[i].z,Afto[i].g,'\n');
```

```
}
fclose(uk);
}
}
```

Тоже самое для кнопки чтения

```
void CLab4Dlg::OnButton6()
{CString Name1;
char am[50];
CFileDialog   fileDialog(TRUE,NULL,"*.txt",NULL,"(*.*)|*.*|(*.txt)|*.txt|");

int result = fileDialog.DoModal();
 if (result==IDOK)
 {        AfxMessageBox(fileDialog.GetPathName()); //

 Name1=fileDialog.GetPathName();
uk=fopen(Name1,"r");

//        uk=fopen("f1.txt","r");

 while (!feof(uk))
 {
fscanf(uk,"%s %f %f %f",am,&Afto[ki].r,&Afto[ki].z,&Afto[ki].g);
        if (feof(uk)) break;
    Afto[ki].Marka=am;
   m_List.AddString(am);
//        m_marka=Afto[ki].Marka;
        ki=ki+1;
        };

   fclose(uk);
UpdateData(0);
 }
}
```

И наконец для кнопки Cancel можно придумать следующую структуру.

```
void CLab4Dlg::OnCancel()
{
        // TODO: Add extra cleanup here
        int a =AfxMessageBox(" Желаете сохранить структуру в файле.",1);
        if (a==1) {
        CLab4Dlg::OnButton5();
        }
        CDialog::OnCancel();
}
```

Список рекомендуемой литературы

1 Давыдов В.Г Программирование и основы алгоритмизации: Учеб. пособие. - М.: Высш.шк , 2003.
2 Холзнер С. Microsoft Visual C++6 с самого начала – СПб.: Питер 2005.
3 Либерти Джесс. Освой самостоятельно C++ за 21 день.- М.: Издательский Дом "Вильяме", 2001. - 814 с.
4 Павловская Т.А. C/C++: Программирование на языке высокого уровня. Учебник. - СПб.: Питер,2001. - 460 с.
5 http://www.cetiforum.ru
6 http://www.microsoft.ru
7 http:// www.firststeps.ru
8 http://www.realcoding.net/
9 Франка П. С ++ учебный курс**:** Питер – 2005.
10 Павловская Т.А., Щупак Ю.А. **C++.** Объектно-ориентированное программирование: Практикум: Питер – 2005

Скобелев Игорь Вадимович

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

для самостоятельной работы по дисциплине
«Программирование и основы алгоритмизации»
для студентов специальностей 15.03.04 «Автоматизация технологических процессов и производств» и 27.03.04 «Управление в технических системах»

Авторская редакция

Подписано к печати	Формат 60x84 1/16	Бумага 65 г/м^2
Печать трафаретная	Усл.печ.л. 2,25	Уч.-изд.л. 2,25
Заказ	Тираж 25	Не для продажи

Редакционно-издательский центр КГУ.
640669, г. Курган, ул. Советская, 63/4.
Курганский государственный университет.