Тажмамат уулу Кубанычбек

Что такое программирование

Кто хочет стать программистом, может начать с этой книги.

Препринт
2010 г.

Введение

Дорогие читатели. Вы держите в руках книгу, в которой отражен многолетний опыт подготовки специалистов. Целями данной книги является:

1) заинтересовать читателя программированием;
2) показать по возможности самую короткую дорогу к освоению профессионального программирования.

Мы постарались кратко и доступно изложить основы теории, дав не только сами знания, но и показав место каждой крупицы знаний в общей картине мира, и закрепить эти знания практическими примерами, близкими к задачам, которые приходится решать программистам.

Для того, чтобы не повторяться, в некоторых вопросах мы отсылаем к соответствующим частям Сообщения о языке, вложенного в используемую нами систему программирования. Новичкам сразу весь текст этого Сообщения читать очень трудно, но отдельные части этого Сообщения читаются довольно легко, таким образом, маленькими шагами вы сможете научиться самостоятельно работать с документацией.

Мы надеемся, что пройдя курс занятий с этой книгой, остальные необходимые знания читатель сможет научиться сам добывать из Интернета, документации к программам и других источников.

Мы не претендуем на то, что все написанное в книге является придуманным нами, как выражался Исаак Ньютон, мы стоим на плечах гигантов, может быть, поэтому нам видно чуть больше.

Почему написана эта книга

При подготовке специалистов автору нужна была одна достаточно простая, и в то же время доводящая до профессионального уровня книга, начав с которой ученик может стать приносящим прибыль сотрудником и дальше самостоятельно разбираться в море уже написанных книг и пособий по программированию.

Книга пишется одновременно с проведением годичных курсов по программированию, поэтому материал книги проверяется на усвояемость все новыми и новыми учениками.

Дополнительно к этой книге мы предлагаем практический курс. На практическом курсе ученики выполняют настоящую работу – совместно делают аналоги существующих популярных компьютерных программ. У каждого ученика задание получается индивидуальным. Когда сделанные учениками программы можно будет практически применять, мы эти программы сделаем доступными бесплатно по Интернету вместе с исходными кодами и указанием, кто какой кусок программы написал. Для тех, кто учится на наших курсах, это будет лучшей рекламой и лучшим документом, подтверждающим квалификацию и работоспособность – настоящее **портфолио**. В течение двенадцати месяцев практического обучения можно стать хорошим программистом.

Для тех, кто хочет учиться на курсах, существуют скидки. Кто выиграет специальный тренажер – компьютерную игру «Сапер», уровень «Профессионал», без учета времени, скидка составит 50%. Кто выиграет эту игру, уровень «Профессионал», за менее, чем 180 секунд, может учиться бесплатно.

Что такое программирование

Как сказал один умный человек, программирование – это умение разделить сложную работу на множество простых работ, чтобы ее мог выполнить любой дурак. Роль дурака выполняет компьютер. У компьютера нет разума. И никогда не будет. Это предмет веры.

Зачем нужно программировать

Это связано с заменой труда человека на труд машины. Человеку надо давать зарплату, человек может ошибаться, лениться, протестовать, заболеть, умереть. Машине нужна электроэнергия, иногда надо заменять некоторые детали, иногда заменять всю машину. Многие ошибки, которые машина может допустить, человек знает заранее и поэтому их можно избежать или уменьшить их влияние. Когда труд машины становится дешевле или надежнее труда человека, или гибель машины может заменить гибель человека, становится выгодней заменить человека машиной, и здесь всегда нужен другой человек – Программист.

Со временем условия и приоритеты меняются. Раньше, когда компьютеры и их время были очень дорогими, люди стремились экономить время машин, делать программы быстрей, исключая ненужные действия. Сейчас компьютеры очень дешевы, труд человека стал цениться больше, поэтому стараются работу делать так, чтобы сэкономить время человека – программиста, иногда в ущерб скорости исполнения программы на компьютере.

Иногда некоторые предприниматели запускают в рекламе мысль о том, что программирование уже не нужно. Так поступали в свое время продавцы программ Windows(tm) 3.1, Clipper (tm), Matlab (tm) и вполне взрослые люди начинали везде повторять эту мысль. Но полностью необходимость в программировании ни одна система не изжила, они просто заменяли некоторые операции, выполняемые человеком-программистом на работу компьютера. Для того, чтобы программировать, нужен разум, и никакое изделие или программа разума не заменят. Чем больше техника входит в нашу жизнь, тем больше программистов нужно для общества, поэтому получаемые Вами, дорогой читатель, при работе с этой книгой знания, умения, навыки не пропадут, пока существует и развивается Человечество. Начав с роли программиста, со временем Вы можете стать руководителем или предпринимателем, в крайнем случае – постановщиком задач для молодых программистов.

Для предпринимателей. Программирование – это такой вид деятельности, где большую долю издержек производства составляет только зарплаты сотрудников, намного меньшую долю – оборудование (обычные компьютеры + сеть) и литература. Это позволяет создавать продукцию на экспорт для крупного международного рынка. В слабых и бедных государствах программирование позволяет с одной стороны, использовать дешевую рабочую силу, с другой стороны, эффективно уклоняться от вредного воздействия коррупции, так как материального экспортируемого товара нет. Одни из самых крупных состояний нажиты на продаже продукции программирования.

Как на этом делать деньги

Что такое профессиональное программирование? Это значит: путем программирования человек находит для себя свой хлеб насущный. Поэтому здесь также затрагиваются вопросы, касающиеся экономики программирования. Вот несколько вариантов добывания денег через программирование:

1) занять место ушедшего программиста в какой-либо фирме и продолжить делать его программу или иным образом устроиться на работу программистом за зарплату;
2) оказывать услуги по переводу части работы каких – либо фирм (крупных единичных клиентов) с плеч людей на плечи компьютеров;
3) сделать свою программу и продавать ее (массовые клиенты);
4) сделать свою программу и давать ее всем бесплатно, разместив в программе рекламу (деньги брать с рекламы);
5) сделать свою программу, но никому ее не давать, оказывать на ее основе услуги для единичных клиентов лично;
6) сделать свою программу, но никому ее не давать, оказывать на ее основе услуги для массовых клиентов через Интернет;
7) делать программы, управляющие работой каких-либо устройств.

Может быть, со временем Вы найдете свои способы добывания денег через программирование. Программистов всегда не хватает, поэтому для граждан слабых и бедных государств это хороший шанс найти работу или бизнес в сильных и богатых государствах.

Какую еще пользу даст умение программировать

Если человек научится повелевать безумными машинами, то ему легче будет научиться повелевать разумными людьми. Из программистов получаются хорошие руководители – менеджеры, офицеры. Конечно, умение программировать не заменяет собой умение руководить людьми, но это позволяет быстрее научиться руководить людьми. А руководителей общество ценит.

Для изучения искусства управления людьми можете ознакомиться с книгой Мухина «НАУКА УПРАВЛЯТЬ ЛЮДЬМИ: Изложение для каждого», свободно доступной в Интернете, но только в той части, где он описывает управление предприятием. По мнению авторитетных людей, управление предприятием (экономическая наука) и управление обществом в целом (экономическая система) – это разные темы, смешивать их, подобно А. Смиту и Риккардо, ошибочно. Экономическая наука (управление предприятием, создание материальных благ) едина для всех политических систем. А экономическая система (распределение благ в обществе) – это уже политика и идеология. Такие ошибки приводят к кризисам капитализма и пролетарским революциям.

Какие роли играет человек в работе с машиной

1. Оператор – это человек, который только нажимает на кнопки готовой программы. Таких людей очень много, поэтому программа делается так, чтобы этим людям было очень удобно использовать программу. Работа оператора самая легкая и самая низкооплачиваемая. Иногда операторы выполняют свою работу не за деньги, а за другое вознаграждение, например, любители компьютерных игр.
2. Системный администратор – это человек, который устанавливает, настраивает или поднимает «упавшую» программу. Таких людей намного меньше, чем операторов, поэтому часто их работу не очень сильно упрощают, когда делают программу. Поэтому системному администратору надо знать много разных скучных деталей использования программы. Системными администраторами часто становятся люди, у которых не получается программировать. От этого у

них вырабатывается комплекс неполноценности, который они удовлетворяют своей властью над операторами.

3. Программист – это человек, который делает программу.
4. Постановщик задач – это человек, хорошо знающий дело, которое выполняют люди и которое должны выполнять машины. Он должен объяснить программисту это дело. Хорошими постановщиками задач становятся старые программисты, которым уже трудно переучиться на новые инструменты программирования, и которые не набрались смелости, чтобы стать предпринимателями.
5. Тестер – это человек, который использует недоделанную программу и старается за деньги или другое вознаграждение найти в ней ошибки. Бесплатно готовы быть тестерами самые нетерпеливые операторы/клиенты – покупатели программ.
6. Кодер – это человек, который переводит программу с одного языка на другой. Со временем работу кодеров заменяют программы, называемые компиляторами.
7. Клиент – это человек, принимающий решение о покупке программы или услуги на ее основе.
8. Предприниматель – это человек, который делает работу программиста возможным, находит деньги для его зарплаты и ищет, как выгодней продать труд программиста. Предприниматель с одной стороны изучает, что надо сейчас или что может понадобиться в будущем клиенту, за что клиент готов добровольно выложить деньги, с другой стороны изучает, как лучше и дешевле заставить программиста работать. Многие предприниматели и в том и в другом ошибаются и становятся банкротами. Работа предпринимателя самая трудная, но он берет себе большую часть денег, если получается организовать работу.

Иногда один человек играет одновременно несколько ролей. И тогда у него внутри может идти борьба мотивов. С другой стороны, если один человек выполняет несколько функций, он может сэкономить много времени, затрачиваемое на то, чтобы один человек объяснил другому, что ему надо.

Как работает компьютер

Вы можете себе представить, как работает автомобиль с механической трансмиссией и двигателем внутреннего сгорания. Вы можете представить, как работает осветительная лампочка накаливания. Некоторые из вас могут даже представить себе, как работает газоразрядная лампа. Для того, чтобы стать профессиональным программистом, надо хорошо себе представлять, как работает компьютер, что внутри него.

Задайте себе вопрос: с чем работает компьютер? Кто-то скажет, что компьютер работает со всем. Но это не очень правильный ответ. Компьютер не может копать. Компьютер не может забивать гвозди. Значит, компьютер работает с объектами одного определенного рода. Посмотрев на компьютер, вы можете сказать: с числами, текстом, рисунками, чертежами, звуками, видео и т.п. Что общего между всем этим? Ответ – информация.

Задайте себе еще один вопрос. Что еще кроме компьютера может работать с информацией? Ответ – человеческий Разум. Другого мы в этом мире не наблюдали. Конечно, мозг животных тоже воспринимает через органы чувств изображения, звуки, запахи и т.п. Однако у компьютера и человеческого разума есть то, чего нет у мозга животных – способность воспринимать и обрабатывать произвольную, в том числе абстрактную информацию.

Значит, задачей компьютера является хранение и обработка произвольной информации.

Как компьютер хранит и передает информацию

Как много информации люди могут сохранить? Очень много, но ограниченно много. Ограниченно, так, как объем информации, сохраняемый отдельным человеком, мал. Общее количество людей тоже ограниченно.

Как мало информации люди могут сохранить? Некоторые из вас сразу скажут – ноль. Самый малый возможный объем сохраняемой информации – нулевой.

Сформулируем вопрос иначе. Как мало информации люди могут сохранить за исключением нулевого объема. Попробуйте на этот вопрос ответить самостоятельно, не читая дальше.

Самый малый объем хранимой информации – это одна единица информации. Информация для профессора Плейшнера из кинофильма «17 мгновений весны» о том, что на явочную квартиру заходить нельзя – наличие горшка с цветком на окне. Как мал этот кусок информации, но стоил жизни для того, кто неверно ее обработал.

Другие примеры самого маленького кусочка информации.

Некоторые люди звонят по сотовому телефону, но не говорят, а сразу сбрасывают звонок. Человек на другом конце после прихода звонка понимает, что он значит, и, например, идет открывать дверь. Такой звонок почти ничего не стоит, но он передает важную информацию. Такое использование сотового телефона стало так популярным, что компании, оказывающие услуги сотовой связи, решили брать отдельную плату за каждый звонок дополнительно к оплате за время разговора.

Моя мама, если в доме никого нет, оставляет включенным свет. Это – информация для вора, что дома хозяева есть, и они не спят. Большинство воров воспринимают эту информацию и боятся войти в такой дом.

По законам некоторых стран ворам по решению суда отрубают кисть. Это - информация для воров, как носителей философии паразитического образа жизни, что воровать *нельзя*, другие меры на них не действуют.

Биты и байты

Итак, самый маленький кусочек информации – это *бит*. С английского языка так и переводится – «кусочек». Он принимает два значения, отличающихся друг от друга. Эти значения условно назовем через «0» и «1». Один великий математик доказал, что самыми эффективными будут компьютеры, в которых самый маленький кусочек информации может принимать три значения $-\sqrt{2}$, 0, $\sqrt{2}$, но в то время, когда появились первые цифровые компьютеры, проще было сделать систему, в которой самый маленький кусок информации мог принимать только два значения, а потом люди к этому привыкли.

У шпионов и разведчиков есть свой способ понимания какой либо сложной ситуации. Если ситуация слишком сложна, чтобы ее охватить одним взглядом, они стараются уменьшить или увеличить картинку до тех пор, пока она не станет очень простой. Попробуем этот способ применить и мы.

Итак, компьютер работает от электричества. Но внутри него куча проводов и деталей. Попробуем уменьшить картину. Какой самый простой электрический прибор мы видим перед собой на стене?

Выключатель. Через него течет электрический ток. Это выключатель может принимать два состояния. Выключен/Включен. Вот вам и *бит*.

Один выключатель может хранить, таким образом, информацию о двух состояниях. Как сказал Кара-Мурза в своей свободно доступной по Интернету книге «Манипуляция сознанием», слова играют очень большое значение для человека. Мы тоже стараемся, манипулируя сознанием ученика, сформировать из него специалиста. Поэтому очень важно включать в речь термины. Мы будем применять слово «бит». А вы представляйте себе при этом выключатель.

Итак, *один* бит имеет *два* возможных состояния.

ВЫКЛЮЧЕН	0
ВКЛЮЧЕН	1

Кстати, на самом выключателе часто написаны 0 и 1.

А сколько возможных состояний у *двух* битов (мысленно представляем два включателя рядом)? Давайте посчитаем.

ВЫКЛЮЧЕН	ВЫКЛЮЧЕН	00
ВЫКЛЮЧЕН	ВКЛЮЧЕН	01
ВКЛЮЧЕН	ВЫКЛЮЧЕН	10
ВКЛЮЧЕН	ВКЛЮЧЕН	11

Четыре состояния. А теперь у *трех* битов?

ВЫКЛЮЧЕН	ВЫКЛЮЧЕН	ВЫКЛЮЧЕН	000
ВЫКЛЮЧЕН	ВЫКЛЮЧЕН	ВКЛЮЧЕН	001
ВЫКЛЮЧЕН	ВКЛЮЧЕН	ВЫКЛЮЧЕН	010
ВЫКЛЮЧЕН	ВКЛЮЧЕН	ВКЛЮЧЕН	011
ВКЛЮЧЕН	ВЫКЛЮЧЕН	ВЫКЛЮЧЕН	100
ВКЛЮЧЕН	ВЫКЛЮЧЕН	ВКЛЮЧЕН	101
ВКЛЮЧЕН	ВКЛЮЧЕН	ВЫКЛЮЧЕН	110
ВКЛЮЧЕН	ВКЛЮЧЕН	ВКЛЮЧЕН	111

Восемь состояний. Как вы видите, единицами и ноликами даже удобней записывать, чем словами. В словах букв много, пока человек их прочтет и поймет, уходит больше времени, а в единицах и ноликах посмотрел, и сразу понятно.

Теперь давайте посмотрим внимательно на возможные состояния одного, двух и трех битов. Что в них общего и в чем разница?

А отличие и разница в том, что при переходе от одного бита к двум битам сначала пишутся все состояния для одного бита с добавлением нолика в начале, затем пишутся опять состояния для одного бита, но уже с единицей в начале. То же самое при переходе от двух битов к трем битам.

Значит, таким образом, мы легко можем посчитать количество состояний для любого числа битов.

А еще значит, что при увеличении числа битов на единицу, количество состояний увеличивается в два раза. Можем даже построить таблицу.

1	2
2	4
3	8
4	16
5	32
6	64
7	128
8	256
9	512
10	1024
11	2048
12	4096
13	8192
14	16384
15	32768
16	65536
17	131072
18	262144
19	524288
20	1048576
21	2097152
22	4194304
23	8388608
24	16777216
25	33554432
26	67108864
27	134217728
28	268435456
29	536870912
30	1073741824
31	2147483648
32	4294967296

Значит при 32 битах возможно более четырех миллиардов состояний. В конце восьмидесятых и начале девяностых был переход от 16 битных систем к 32 битным. Вам задание на самостоятельную работу, определить, сколько состояний будет для 64 битов. Сейчас идет процесс перехода от 32 битных систем к 64 битным системам.

Кто знаком с математикой, может вспомнить о *геометрической прогрессии*. Сила геометрической прогрессии в том, что, начав с малого можно быстро дойти до очень

большого количества. Так, немного закваски превратят в кефир любое количество молока. Другой пример – одна ячейка сетевого маркетинга разросшись покроет за малое время любую страну. Третий пример – один маленький кружок большевиков в царской России среди пролетариата быстро разросся до таких размеров, что со временем смог захватить власть в государстве. Четвертый пример – цепная реакция распада тяжелых ядер, при соответствующих условиях начавшись случайно с одного любого ядра, распространится очень быстро на весь материал, способный распадаться, и мы видим взрыв атомной бомбы. Главное – правильно подобрать принципы роста и подготовить или найти соответствующие условия.

Байт – это восемь битов. 32 бита – это 4 байта. Восемь битов имеет 256 возможных состояний. Создатели первых компьютеров думали, что это достаточно для обозначения все больших и малых букв любого алфавита (они не думали о китайском и японском языках), цифр и часто используемых знаков и поэтому, видимо, этому количеству битов дали особое название, хотя можно было бы обойтись и без него.

Килограмм это 1000 грамм. Мегатонна это 1000*1000 тонн. Эти приставки люди придумали, чтобы большие количества чего-либо можно было произносить также легко, как и обычные количества. Такая же ситуация и с битами. Но в таблице, которую мы только что составили, числа 1000 нет. Есть 1024. Это 2^{10}. Вот и решили для 1024 применять приставку «*кило*». 1024 бит это килобит. 1024 байт это килобайт или 8*1024 килобит. 2^{20}=1024*1024=1048576 байт это мегабайт. И так далее. Попробуйте сами подсчитать, сколько байтов и битов в гигабайте (2^{30}) и терабайте (2^{40}).

Разница между 1000 и 1024 иногда используется некоторыми фирмами для обмана покупателей. Так, объявляя размер винчестера (накопителя на жестких магнитных дисках) в гигабайтах, за гигабайт они имеют в виду 1000*1000*1000, а не принятые в компьютерном мире 2^{30}=1024*1024*1024, сами подсчитайте разницу.

Скачайте инструментарий разработчика бесплатно

В качестве основной среды программирования мы выбрали систему BlackBox с языком программирования Компонентный Паскаль, сделанной по мотивам произведений известного мастера создания языков программирования и компьютерных систем Никлауса Вирта. В качестве базы данных СУБД Firebird. В качестве дополнительного инструмента для языка C/C++ компиляторы от Borland(tm) и gcc. Инструкция по скачиванию и запуску приведена в приложениях в конце этой книги. Все эти системы бесплатны, условия использования даны в их лицензиях. Общеизвестно, что наиболее широко применяемые языки C/C++ не очень подходят для начального изучения программирования, так как требуют слишком большой внимательности от человека, можно сказать, внимательности, присущей скорее компьютеру (компилятору). И в то же время много программ и программных интерфейсов сделаны на языках C/C++, поэтому программист вынужден их изучать и использовать. Мы не ввязываемся в так называемые холивары (holy wars – «священные войны») о выборе языка программирования, так как они отвлекают от настоящих Священных войн.

В приложениях в конце книги также сказано, как создать свою подсистему и другие технические подробности.

Целые числа

Итак, 3 бита имеют 2^3=8 возможных состояний. Пронумеруем каждое из состояний, начиная с 0.

0	000
1	001
2	010
3	011

4	100
5	101
6	110
7	111

Вот мы и получили возможность кодировать через три бита целые числа от 0 до 7. Соответственно, для четырех битов мы можем задать беззнаковые целые числа от 0 до 15. Можем составить таблицу.

битов	начальное значение	конечное значение	общее количество
3	0	7	8
4	0	15	16
8	0	255	256
16	0	65535	65536
32	0	4294967295	4294967296

Задание на самостоятельную работу – определите, какое максимальное беззнаковое целое можно задать при помощи 64 битов.

Ну, а если нам понадобятся и положительные и отрицательные числа? Тогда, естественно, часть значений используем для задания отрицательных чисел.

0	000
1	001
2	010
3	011
-4	100
-3	101
-2	110
-1	111

Как вы видите, половина таблицы заполнена также, как и для беззнаковых целых, а вторая половина заполнена отрицательными значениями так, что они тоже возрастают. Вновь составим таблицу.

битов	мин. значение	макс. значение	общее количество
3	-4	3	8
4	-8	7	16
8	-128	127	256
16	-32768	32767	65536
32	-2147483648	2147483647	4294967296

Задание на самостоятельную работу – определите, какое максимальное и минимальное знаковые целые можно задать при помощи 64 битов.

Ну вот, мы и добрались до первой программы.

```
MODULE ProgbookIntegers;
	IMPORT  Log := StdLog,Dialog;
	CONST
		l*=5;
		m=6;
	VAR
		i,j*,k-:INTEGER;

PROCEDURE Test* ();
```

```
BEGIN
	Log.String('Исходные значения');Log.Ln;

	Log.Int(i);
	Log.Int(j);
	Log.Int(k);

	Log.Ln;

	i:=5;
	k:=j+1;

	Dialog.UpdateInt(k);

	Log.String('Измененные значения');Log.Ln;

	Log.Int(i);
	Log.Int(j);
	Log.Int(k);

	Log.Ln;
	Log.Ln;

	Log.String('Минимальное и максимальное значения');Log.Ln;

	Log.Int(MIN(INTEGER));
	Log.Int(MAX(INTEGER));

	Log.Ln;
	Log.Ln;

	Log.String('Константы');Log.Ln;

	Log.Int(l);
	Log.Int(m);

	Log.Ln;
	Log.Ln;

END Test;


END ProgbookIntegers.
```

ProgbookIntegers.Test

После ключевого слова CONST следуют константы. После ключевого слова VAR следуют переменные. Все это – ячейки в памяти, которым для нашего удобства даются понятные нам имена l, m, i, j, k. Как Вы, наверное, уже догадались, константы не изменяют своего значения во время выполнения программы, а переменные могут изменять значения.

Может возникнуть вопрос – а зачем придумывать еще какие-то имена для констант, когда можно просто вписать само это значение там, где надо? Тут есть одна хитрость. Рано или поздно вместо этих жестко вписанных значений Вам придется жестко вписывать другие значения, а найти их всех и исправить в большом тексте программы займет довольно много времени, и кроме того, можно что-то упустить. А если значения вводятся только там, где придумываются имена констант, то таких мест сравнительно мало, и их легко исправить, как говорится, если враг обнаружен, то он уничтожен.

Естественно, имена должны между собой отличаться, чтобы не было путаницы. Но невозможно сделать так, чтобы ВСЕ имена в мире были разными, так как программистов очень много и согласовывать между ними ВСЕ имена констант и переменных, ясное дело, нереальная задача. Даже для одного программиста давать разные имена ВСЕМ переменным и константам – уже затруднительно. Поэтому введено понятие области видимости. Подробней мы объясним это позже, но сейчас надо уяснить одно – в пределах одной области видимости имена должны быть отличающимися между собой. Примеры мы Вам даем такие, в которых есть ОДНА область видимости.

«*» и «-»

Вы заметили, что рядом с именами некоторых переменных стоят значки «*» и «-». Это означает, что эти переменные извне можно читать и изменять для «*» и только читать для «-». Чуть далее, когда научитесь извне выполнять эти действия, будет наглядный пример. Естественно, константы, помеченные «*» извне и внутри можно только читать.

INTEGER

Как вы заметили, в примере есть ключевое слово INTEGER. Это в данной реализации 32 битное знаковое целое. Минимальное и максимальное значения, допускаемые в этом типе, можно узнать через MIN(INTEGER) и MAX(INTEGER). Кроме типа INTEGER есть еще и другие целые, с другим числом битов, знаковые и беззнаковые. Задание на самостоятельное выполнение – после завершения чтения параграфа о целых числах дополните следующую таблицу.

тип	MIN	MAX
BYTE		
SHORTINT		
INTEGER	-2147483648	2147483647
LONGINT		

Декларативная и императивная части.

Наша программа состоит из двух частей: декларативной и императивной. Слово «декларативная» напоминает декларацию – документ, заполняемый при переходе через границу, где вы указываете, что везете с собой. В программе в декларативной части вы объявляете, с чем будете работать. Слово «императивная» напоминает слово «император». Этот человек отдает всем приказы, а ему никто не смеет приказать. В императивной части вы отдаете приказы компьютеру. Вот мы вырезали императивную часть нашей программы.

```
BEGIN
	Log.String('Исходные значения');Log.Ln;

	Log.Int(i);
	Log.Int(j);
	Log.Int(k);

	Log.Ln;

	i:=5;
	k:=j+1;

	Dialog.UpdateInt(k);

	Log.String('Измененные значения');Log.Ln;

	Log.Int(i);
	Log.Int(j);
	Log.Int(k);

	Log.Ln;
	Log.Ln;
```

```
	Log.String('Минимальное и максимальное значения');Log.Ln;

	Log.Int(MIN(INTEGER));
	Log.Int(MAX(INTEGER));

	Log.Ln;
	Log.Ln;

	Log.String('Константы');Log.Ln;

	Log.Int(l);
	Log.Int(m);

	Log.Ln;
	Log.Ln;

END
```

Все остальное можно отнести к декларативной части.

Знак «:=» используется для того, чтобы дать приказ машине поместить в указанную переменную указанное значение. Имя переменной слева, значение справа.

Команда Log.String() дает приказ распечатать в окно рабочего журнала строку, указанную внутри скобок.

Команда Log.Int() дает приказ распечатать в окно рабочего журнала целое значение, указанное внутри скобок. Это может быть значение переменной, константы или выражение.

Команда Log.Ln дает приказ машине закончить строку в окне рабочего журнала и начать новую (возврат каретки или Enter).

Команда Dialog.UpdateInt() заносит на форму новое значение отображаемой на ней переменной.

Создание подсистемы.

Для удобства работы с программы в BlackBox хранятся в виде подсистем. Подсистема – это такая специальная папка в папке BlackBox. В нашем случае это Progbook. Внутри нее есть другие папки с определенными именами, нас сейчас интересуют следующие из них:

Mod	Здесь хранятся тексты программ
Rsrc	Здесь хранятся файлы для форм, меню и др.
Docu	Здесь хранится документация по программам

Имена файлов в этих папках могут совпадать, но содержание разное, в зависимости от того, в какой из этих папок лежит файл.

Создание формы

Через форму пользователь вводит в программу данные, видит результат, запускает некоторые элементы программы. Формы в нашей системе делаются довольно легко.

Выполните следующие инструкции.

1. Меню «Диалоги»/ «Новая форма».
2. В поле «Связать с» наберите «ProgbookIntegers»
3. Появится форма с полями и кнопками. Сохраните ее в Progbook/Rsrc/Integers.odc .

Запустить эту форму можно одним из двух способов.

Способ 1. Меню «Диалоги»/ «Открыть как инструментальный диалог».

Способ 2. Открыть файл Progbook/Rsrc/Menus.odc . После строки

MENU "Progbook"

добавить туда строку

"MyIntegers" "^1" "StdCmds.OpenAuxDialog('Progbook/Rsrc/Integers','Integers')" ""

Потом сохранить и закрыть файл. Меню «Инфо»/ «Обновить меню». Появится пункт меню «Progbook»/ «MyIntegers». При запуске этого пункта меню будет появляться Ваша форма.

Значения тех переменных, которые были помечены «*», можно читать и менять в форме. Значения тех переменных, которые были помечены «-», можно читать с формы.

Попробуйте вводить разные значения в поля и понажимайте на кнопку на форме и смотрите на результат в окне рабочего журнала.

Попробуйте найти зависимость между исходным кодом и тем, что выходит в журнал после нажатия кнопки.

Не следует забывать о максимальных и минимальных значениях целых чисел. Вы должны следить, чтобы значения в ваших переменных не вышли за эти рамки, иначе программа поведет себя непредсказуемо.

Буквы и строки

Символы

Восьми битам соответствует 256 различных значений. 16 битам 65536 различных значений. Кроме целых чисел каждому из этих значений можно задать буквы, рисунки цифр, знаки препинания и другие используемые значки. Вот пример такой таблицы для кодовой страницы WIN1251 (восьмибитовая кодировка).

….(вставить кодовую таблицу)

Могут быть и другие варианты задания символа для каждого из вариантов сочетаний битов (другие кодовые страницы). Кроме того, могут быть и 16 битовые кодовые страницы. В них могут быть задано 65536 различных символов. Этого количества достаточно даже для кодирования китайских иероглифов. Как известно, в китайском письме одному понятию соответствует один отдельный символ. Сколько слов, примерно столько и букв.

16 битовые символы задаются ключевым словом CHAR, 8 битовые – SHORTCHAR. Вот пример с буквами.

```
MODULE ProgbookChars;
	IMPORT  Log := StdLog,Dialog;

	CONST
		g*='9';
		h='+';

	VAR
		d,e*,f-:CHAR;

PROCEDURE Test* ();
BEGIN
	Log.String('Исходные значения');Log.Ln;

	Log.Char(d);
	Log.Char(e);
	Log.Char(f);

	Log.Ln;

	d:=41X;
	e:='F';
	f:=CHR(ORD(e)+1);
```

```
	Dialog.UpdateChar(f);

	Log.String('Измененные значения');Log.Ln;

	Log.Char(d);
	Log.Char(e);
	Log.Char(f);

	Log.Ln;
	Log.Ln;

	Log.String('Минимальное и максимальное значения');Log.Ln;

	Log.Char(MIN(CHAR));
	Log.Char(MAX(CHAR));

	Log.Ln;
	Log.Ln;

	Log.String('Константы');Log.Ln;

	Log.Char(g);
	Log.Char(h);

	Log.Ln;
	Log.Ln;

END Test;


END ProgbookChars.
```

ProgbookChars.Test

Попробуйте запустить этот пример через меню «Progbook»/ «Chars», попробуйте вводить разные значения и посмотрите, что будет в рабочем журнале.

Строки

Одними буквами, естественно, все не ограничивается. Если на каждую букву делать свою переменную, то будет очень неудобно работать. Поэтому вводится массив букв. О массивах подробней будет сказано дальше. Сейчас просто запомните, что массив – это несколько переменных под одним именем, отличающиеся номером, или, как еще говорят, индексом. Доступ к элементам массива происходит через указание индекса в квадратных скобках. Элемент массива-строки является символом (буквой). Длина массива в нашем случае указывается в тексте программы. Однако не вся эта длина используется в ходе работы программы. Для обозначения того, какая часть массива используется, автоматически или вручную в конце строки ставится нулевой бит.

Таким образом, при необходимости, строку можно оборвать на нужном элементе, присвоив следующему после него элементу значение 0 (0X в шестнадцатеричной системе исчисления). Такие строки в программировании называются нуль-терминированными. Сразу на память приходит Терминатор из художественного кинофильма, которого присылали, чтобы оборвать жизнь его жертв.

А вот и пример со строками.

```
MODULE ProgbookStrings;
	IMPORT  Log := StdLog,Dialog;
	CONST
		r*='Здравствуй ';
		s='мир !';



	VAR
		o,p*,q-:ARRAY 255 OF CHAR;

PROCEDURE Test* ();
BEGIN
```

```
	Log.String('Исходные значения');Log.Ln;

	Log.String('['+o+']');
	Log.String('['+p+']');
	Log.String('['+q+']');

	Log.Ln;

	o:="Hello, world!";
	q:=o+'   '+p;
	q[1]:='E';
	q[20]:=0X;

	Dialog.UpdateString(q);

	Log.String('Измененные значения');Log.Ln;


	Log.String('['+o+']');
	Log.String('['+p+']');
	Log.String('['+q+']');

	Log.Ln;
	Log.Ln;

	Log.String('Константы');Log.Ln;

	Log.String(r);
	Log.String(s);

	Log.Ln;
	Log.Ln;

END Test;


END ProgbookStrings.
```

ProgbookStrings.Test

Как вы, наверное, заметили, одну строку можно добавить к другой, используя оператор «+». А еще вы, наверное, смогли заметить, что то, что располагается между одинарными или двойными кавычками, тоже рассматривается в качестве строк.

В модуле Strings вы можете найти много полезных инструментов для работы со строками. Если того, чего надо, там нет, после изучения темы «Массивы» вы можете сами написать что-то свое.

Вещественные числа

Не всегда целые числа достаточны для наших задач. Очень часто приходится использовать числа с дробями. Они называются вещественными числами. Подробней об этом и другом, что связанно с математикой, можно узнать из простым языком написанной книги Рихарда Куранта и Робинса «Что такое математика». Эту книгу Курант, великий математик, написал, чтобы поднять научно-технический потенциал США перед началом второй мировой войны. Как видите, он в этом преуспел. Сейчас перевод на русский язык этой книги и других его книг можно достать свободно в Интернете.

Вещественные числа широко используются для описания природных явлений. Наука, изучающая природные явления, называется физикой. Просто и доступно о физике можете узнать из лекций по физике лауреата Нобелевской премии Фейнмана, перевод которых на русский язык также доступен в Интернете.

Вспомним экспоненциальную форму записи числа.
$3,14=0.314*10^1$.

0.314 – это мантисса, десятичная дробь, которая может принимать значения от 0 до 1. Степень при десятке - это показатель. Таким образом, вещественное число можно задать ДВУМЯ целыми числами, из одного получается мантисса после того, как эти цифры спереди дополняются нулями до нужной длины и в начале дописывается «0.», а из второго, показатель (умножить на десять в степени этого показателя).

Такие вещественные числа называются числами с плавающей десятичной точной. Есть и другие способы представления вещественных чисел, но этот – наиболее дешевый и поэтому наиболее часто применяется. Как китайский товар. Он дешевле, и таким образом выдавливает товары, произведенные в других странах, а к его недостаткам покупателям приходится привыкать.

К недостаткам чисел с плавающей точкой вам, как программистам тоже придется привыкнуть. Надо учесть, что длина мантиссы ограничена. При умножении и делении вещественных чисел это особой проблемы не создает. Проблема может возникнуть, когда мы складываем или вычитаем вещественные числа.

Мантиссы при этом изменяются так, чтобы показатели были равны. Ведь мы всегда можем представить $0.314*10^1$ как $0.0314*10^2$, или даже как $0.0031*10^3$. При этом мантисса одного из двух чисел, участвующих в операции, может выйти за пределы допустимой длины мантиссы и теряются. Это добавляет погрешность в вычисления.

Вредный эффект от такой погрешности стараются уменьшить различными способами: стараются изменить команды компьютеру так, чтобы сложений и вычитаний, особенно чисел с сильно отличающимися показателями, было как можно меньше.

Надо учесть, что вещественные переменные могут принять значения inf – бесконечность, nan (not a number) – «не является числом» (например, если ноль разделим на ноль). Бесконечность – это абстракция, придуманная математиками, чтобы легче было делать некоторые вычисления.

Вот и пример с вещественными.

```
MODULE ProgbookReals;
	IMPORT  Log := StdLog,Dialog;
	CONST
		z1*=2.718;
		z2=-77.7;

	VAR
		x,y*,z-:REAL;

PROCEDURE Test* ();
BEGIN
	Log.String('Исходные значения');Log.Ln;

	Log.Real(x);
	Log.Real(y);
	Log.Real(z);

	Log.Ln;

	x:=3.14;
	z:=y/2;

	Dialog.UpdateReal(z);

	Log.String('Измененные значения');Log.Ln;

	Log.Real(x);
	Log.Real(y);
	Log.Real(z);

	Log.Ln;
	Log.Ln;

	Log.String('Минимальное и максимальное значения');Log.Ln;

	Log.Real(MIN(REAL));
	Log.Real(MAX(REAL));

	Log.Ln;
```

```
		Log.Ln;

		Log.String('Константы');Log.Ln;

		Log.Real(z1);
		Log.Real(z2);

		Log.Ln;
		Log.Ln;

END Test;


END ProgbookReals.
```

ProgbookReals.Test

Логические значения

Существуют чем-то похожие на биты логические переменные и константы.

Сходство с битами в том, что логические переменные и константы могут принимать значения «Истина» и «Ложь».

Разница с битами в том, что логические переменные и константы могут принимать значение «Значение не определено».

Под одну логическую переменную может быть выделен один или несколько байтов, так как память в компьютере выделяется не битами, а байтами. Но когда возникает очень много логических переменных и констант, становится целесообразным применять множества (SET) , о которых отдельно еще будет сказано. Сейчас надо только запомнить, что во множестве (SET) используется каждый бит.

Вот и пример с логическими переменными и константами.

```
MODULE ProgbookBooleans;
		IMPORT  Log := StdLog, Dialog;
		CONST
			d*=TRUE;
			e=FALSE;

		VAR
			a,b*,c-:BOOLEAN;

PROCEDURE Test* ();
BEGIN
		Log.String('Исходные значения');Log.Ln;

		Log.Bool(a);
		Log.Bool(b);
		Log.Bool(c);

		Log.Ln;

		a:=TRUE;
		c:=TRUE;

		Dialog.UpdateBool(c);

		Log.String('Измененные значения');Log.Ln;

		Log.Bool(a);
		Log.Bool(b);
		Log.Bool(c);

		Log.Ln;
		Log.Ln;

		Log.String('Минимальное и максимальное значения');Log.Ln;

		Log.Bool(MIN(BOOLEAN));
		Log.Bool(MAX(BOOLEAN));

		Log.Ln;
		Log.Ln;

		Log.String('Константы');Log.Ln;
```

```
        Log.Bool(d);
        Log.Bool(e);

        Log.Ln;
        Log.Ln;

END Test;


END ProgbookBooleans.
```

ProgbookBooleans.Test

Примеры хранения информации об окружающем мире

При помощи битов можно задавать целые, вещественные, символы и строки, логические переменные и множества. А уже на основе целых, вещественных, символов, строк, логических переменных и множеств можно моделировать объекты окружающего мира.

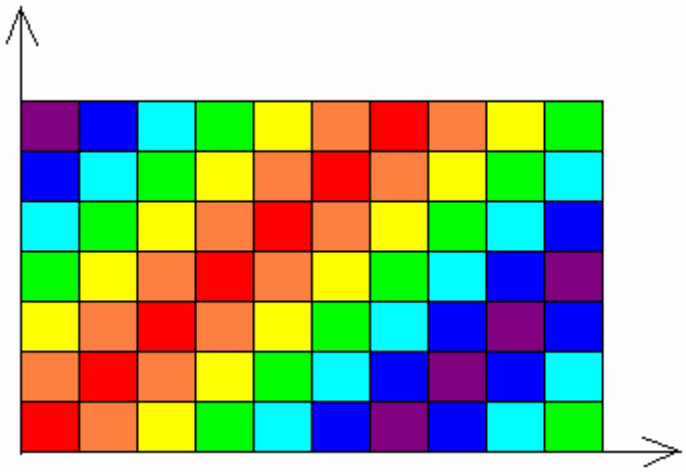
Текст

Текст – самый простой пример. Текст состоит из последовательности символов, каждый из которых задается одним или двумя байтами в соответствии с кодовой страницей. В некоторых случаях текст может содержать символы форматирования. Такой текст читается в двух режимах. В первом режиме все символы показываются. Во втором режиме символы форматирования не показываются, но они влияют на то, как показываются другие символы, например, **жирным**, *курсивом*, подчеркнутым. Примеры текстов с форматированием – языки html, xml, система редактирования LaTeX.
(вставить примеры текстов с форматированием и простые тексты)

Подробней о них можете узнать в Интернете. Скажем только, что система редактирования LaTeX появилась в те времена, когда компьютеры были маломощными не позволяли одновременно отображать и изменять текст, содержащий разные шрифты, тогда люди придумали на экране набирать одно (текст с символами форматирования), а на принтере выводить совсем другое (текст с разными шрифтами, готовый результат). Ввод такого текста был похож на детскую игру «Морской бой», где игроки задают координаты квадрата, по которому стреляет пушка, и только по результату догадываются, уничтожили ли они все корабли противника.

Растровые изображения

Вот простой пример растрового рисунка.



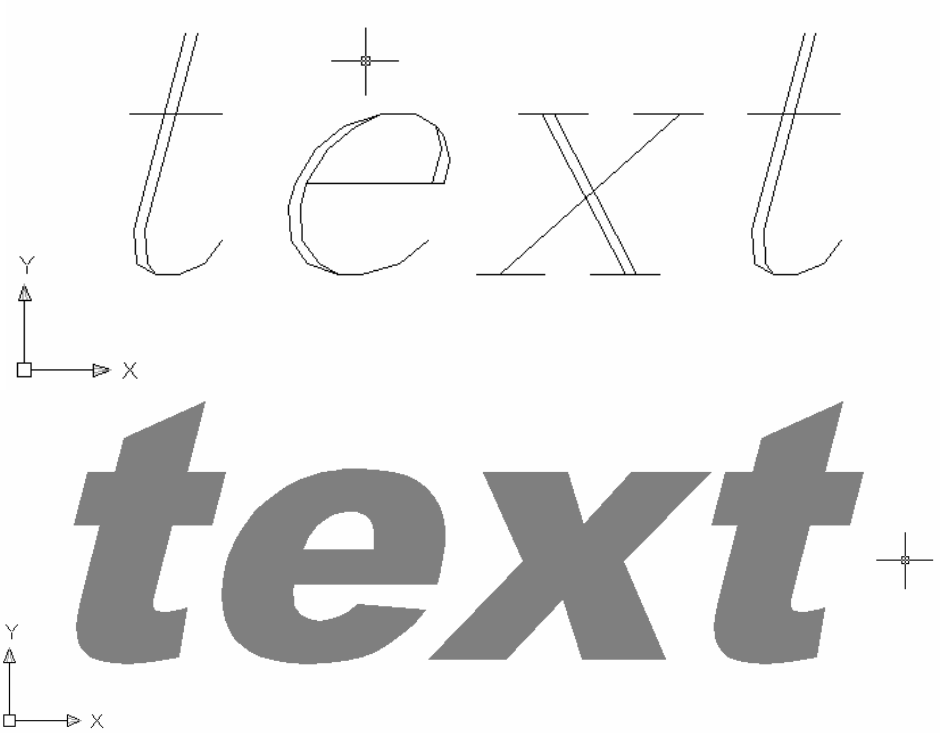
Сначала задаются количество точек по горизонтали и по вертикали. У каждой точки, получается, есть свой целочисленный номер по горизонтали и по вертикали. Чтобы лучше понять, вспомним историю Рене де Карта. Это человек жил во Франции до эпохи капитализма. Руководителем государства был король. Основным средством развлечения народных масс были театры. Причем, каждый посетитель занимал то место, которое ему нравилось. Среди посетителей были также военнослужащие. В армии дисциплина была поставлена плохо. Все споры вооруженные военнослужащие решали, в основном, через дуэль. Участнику дуэли скрыть от трибунала факт участия было легко. В спорах за лучшие места погибали и получали ранения военнослужащие. Объем небоевых потерь катастрофически возрастал. Рене де Карт нашел решение этой проблемы. Он предложил пронумеровать каждый ряд, а в каждом ряду пронумеровать также места. В продаваемых билетах заранее отмечали ряды места, и каждый посетитель мог сесть только на свое кресло, определяемое номером ряда и номером места. Эту систему назвали декартовой системой координат.

Таким образом, после задания координат каждой точки, далее задается цвет каждой точки целим числом. Например, 1 – это красный, 0 – это белый, 5 – зеленый и т.п. Целым числом можно кодировать также оттенки цветов. Иногда задают также сочетания основных цветов, например, RBG – red, green, blue или красный, зеленый, синий, в зависимости от того, какого из этих трех цветов больше, получается различный цвет точки.

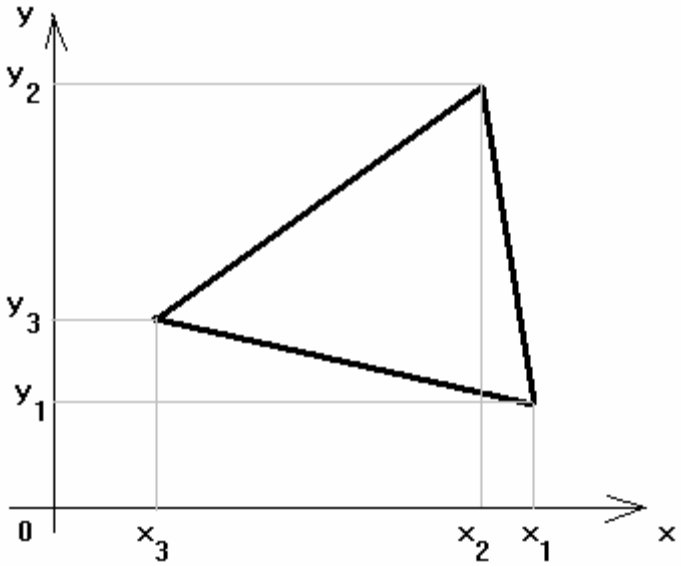
Когда точек достаточно много и картинка достаточно большая, глаз человека начинает воспринимать ее, как единое целое. Таким способом хранятся в виде последовательности битов, например, целые фотографии.

Векторные изображения, чертежи и модели

Векторный рисунок не хранит информацию о каждой точке. В нем содержится информация о контурах и заливке внутри этих контуров.

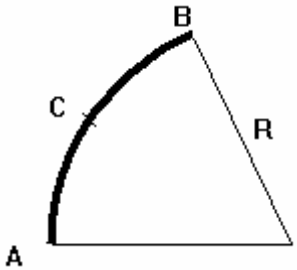


Вот, например, простейший треугольный контур.



В битах через целые и вещественные кодируется количество отрезков, координаты вершин отрезков и цвет внутри контура.

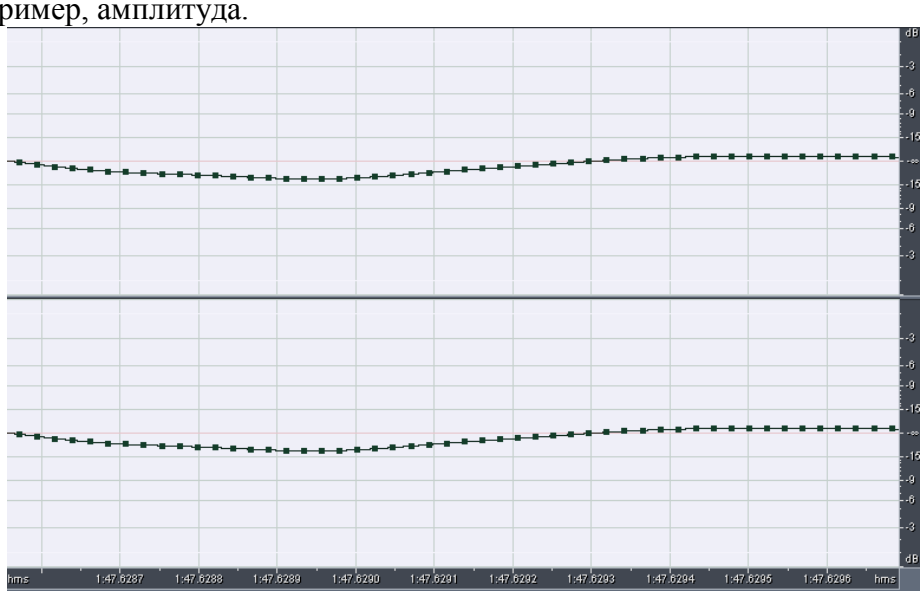
Вместо отрезков могут быть дуги окружностей, описанные координатами трех точек, из которых можно вычислить радиус.



Векторные рисунки являются основой для построения чертежей и моделей. В модели координаты вершин отрезков и других фигур могут быть трехмерными, т.е. задаваться через x,y,z.

Звук

Звук может быть задан несколькими путями. При оцифровке (хранения в виде битов) звука, поступающего из внешнего источника (микрофон, линейный вход), через определенные промежутки времени записываются важные характеристики звукового потока, например, амплитуда.



При воспроизведении такого звука в обратном порядке, конечно, часть информации теряется, но в целом, разница между цифровым и обычным звуком не воспринимается после определенного порога.

Просто иногда от длительного прослушивания цифрового звука может разболеться голова.

Видео

С видео очень просто. С определенной частотой сохраняются растровые кадры (как фотографии), между последующим и предыдущим кадрами разница небольшая, но если показывать подряд много кадров, человеку, который все это смотрит, кажется, что картинка движется. К движущимся кадрам прикладывается звуковая дорожка.

Как битовая информация записывается материально

Ну, а теперь немного о том, как эти биты хранятся на компьютере. С электричеством все ясно: 1 – есть напряжение, 0 – нет напряжения, точнее, для 1 одно напряжение, для 0 другое напряжение.

Перфокарты и перфоленты

Исторически первыми хранителями информации были перфокарты, позднее перфоленты. Сейчас перфокарты используются, как ключи для кодовых замков в подъездах, может быть у кого-то такой замок стоит.

Между двумя электрическими контактами пропускается бумага с дырочками. Если контакты попадают на дырочку, то электрическая цепь замыкается, и есть ток. А если не на дырочку, то цепь разомкнута, тока нет.

Оперативная память

Внутри компьютера есть оперативная память на конденсаторах. 1 – заряд есть, 0 – заряда нет. Такая память работает, пока устройство в целом подключено к электрическому питанию. При отключении память стирается.

Магнитные носители

Внутри компьютеров есть также так называемые накопители на жестких магнитных дисках, или винчестер. Раньше также применялись накопители на гибких магнитных дисках или дискеты, и накопители на магнитных лентах. В них информация (0 и 1) хранится через ориентацию магнитных доменов. Сильное магнитное поле переориентирует направление магнитных доменов (записывает информацию), слабое магнитное поле определяет направление доменов (считывает информацию). Такая память может храниться долго без электрического питания.

Оптические носители

На лазерных дисках (CD, DVD) зеркальные и матовые участки означают 1 и 0. На зеркальном участке луч отражается, на матовом поглощается.

Флэшки

Вот вам задание на самостоятельное выполнение: поищите в Интернете, узнайте, какое природное явление используется для хранения информации флешках.

Как битовая информация передается от компьютера к компьютеру

Как вы поняли, информацию (биты и байты) надо не только хранить, но и передавать. Для этого их надо наложить на какое-нибудь физическое поле и это поле передавать.

Акустические модемы

Исторически первой в мире цифровой связью на расстояние была азбука Морзе. Азбука Морзе это пример манипуляции (отключение-включение) с несущим сигналом.

Далее с развитием электроники одновременно с манипуляцией стали использоваться также модуляция/демодуляция сигнала на несущую частоту. Подробнее об этом можете узнать, например, в википедии (wikipedia.org).

Отметим, если в качестве несущих используются звуковые волны, то это обычный голосовой модем, используемый для соединения через телефонную станцию.

Высокочастотные модемы

Для сетевых плат и беспроводной связи используют в качестве несущих электромагнитные колебания.

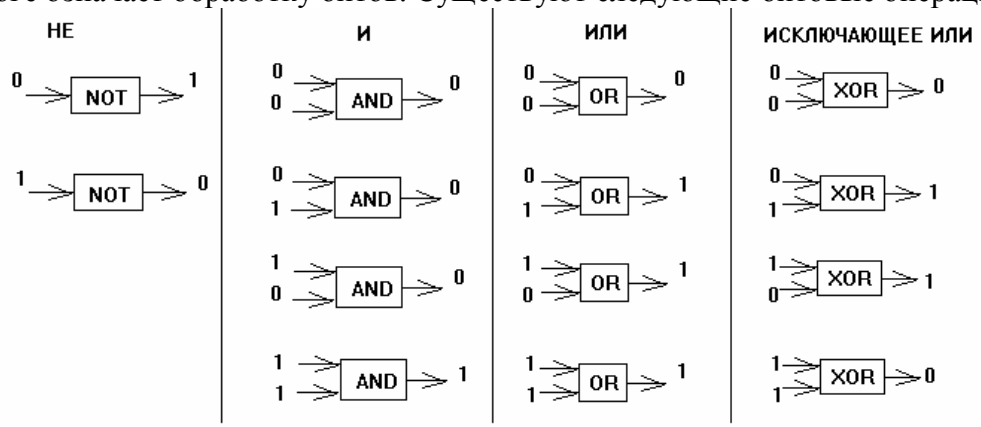
Оптоволоконные линии

В волоконно-оптических линиях связи в качестве несущего используется луч света, с которым проводят операции модуляции и манипуляции.

Как компьютер обрабатывает информацию

Битовые операции

Если компьютер хранит информацию в виде битов, то и обработка информации в конечном итоге означает обработку битов. Существуют следующие битовые операции.



Чтобы легче было разобраться, представьте, что 0 –это ЛОЖЬ, а 1 – это ИСТИНА. НЕ ИСТИНА будет ЛОЖЬ. НЕ ЛОЖЬ будет истиной. Операции И и ИЛИ также легко представить через ИСТИНУ и ЛОЖЬ. ИСТИНА И ИНСТИНА будет ИСТИНОЙ, все остальные манипуляции ИСТИНЫ с ЛОЖЬЮ или сама ЛОЖЬ всегда будут ЛОЖЬЮ.

Единственная из этих битовых операций «ИСКЛЮЧАЮЩЕЕ ИЛИ» не имеет прямого аналога в обыденной человеческой логике. Эту операцию можно представить так: выходной результат будет ИСТИНА, если два входных значения отличаются друг от друга.

Задание на самостоятельную работу – пройтись по каждому из вариантов входных значений по каждой битовой операции и проверить результат лично.

Кроме этих операций есть еще операции битового смещения. Значения всех битов могут сместиться влево

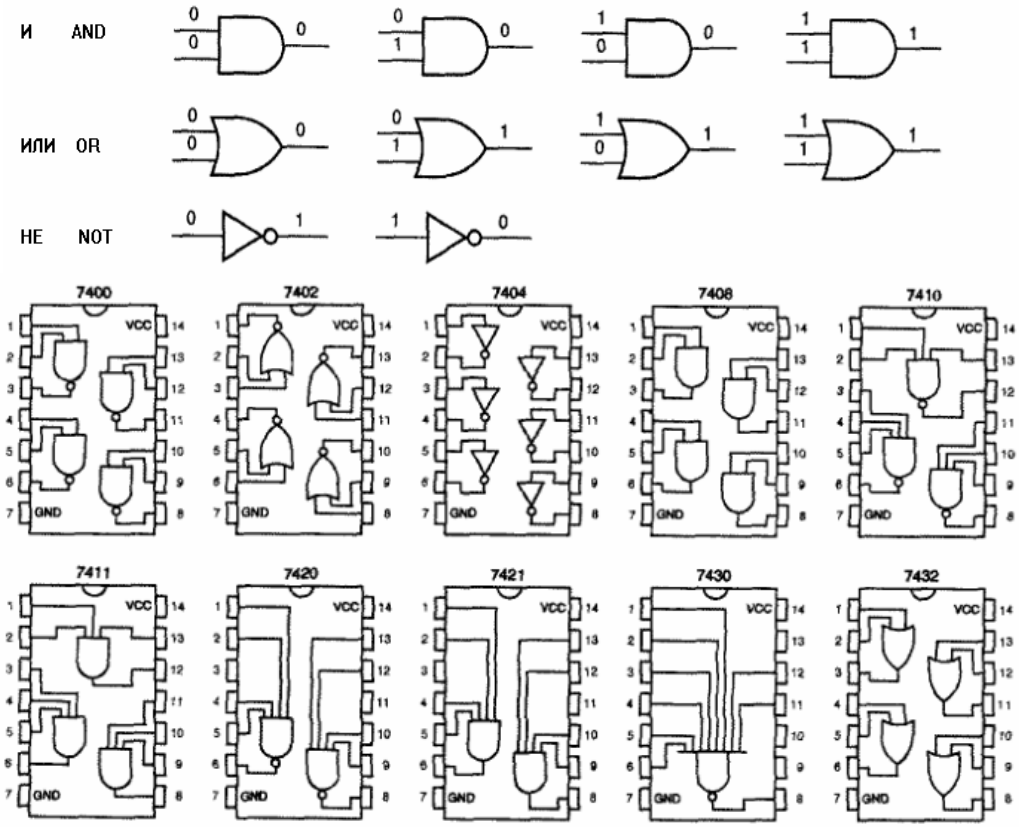
0001000100
0010001000

или вправо.

0001000100
0000100010.

Логические элементы

Существуют электронные схемы, реализующие операции NOT, AND, OR, XOR. При проектировании цифровых микросхем это – кирпичики, из которых собирается сложная микросхема.



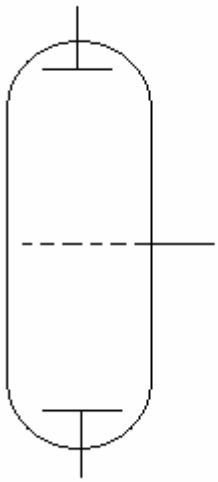
Варианты реализации NOT, OR, AND схем через электронные приборы можете найти в Интернете.

Усилители

Главная возможность для построения NOT, OR, AND электронных схем заключается в том, что в приборах, основанных на некоторых природных явлениях, можно усиливать слабый сигнал, то есть, слабый сигнал управляет сильным потоком и в результате получается усиление сигнала. Можете представить это так. Командир – один человек, но он может контролировать действия многих людей, находящихся в его подчинении, и много людей – подчиненных, могут сделать гораздо больше работы, чем командир один.

Вакуумные лампы

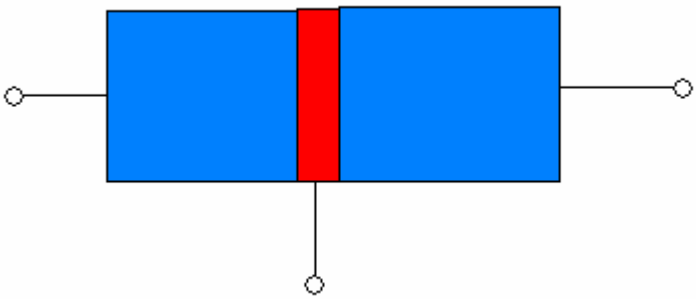
Исторически первыми усиливающими приборами были вакуумные лампы. Внутри лампы откачан воздух и движутся электроны в электромагнитном поле.



Сильный поток – от анода к катоду. Между ними расположена сетка. К сетке подается слабый изменяющийся потенциал. Этот слабый изменяющийся потенциал регулирует мощный поток электронов и мощный поток электронов изменяется так, как изменяется слабый потенциал в сетке. Это и есть усиление слабого сигнала. Первые электронные компьютеры были сделаны на основе ламп и были по размерам очень большими, могли занимать целый этаж в здании.

Транзисторы

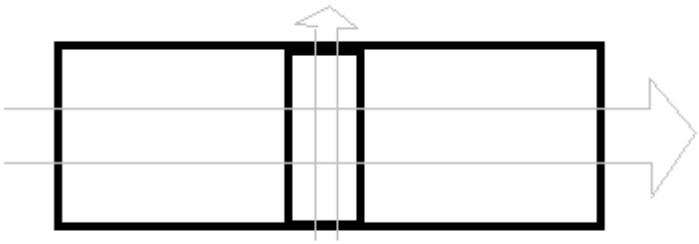
Транзисторы сделаны из полупроводниковых материалов. В полупроводниках можно создавать зоны, в которых ток проводится положительно заряженными частицами (зоны p - проводимости) и зоны, в которых ток проводится отрицательно заряженными частицами (зоны n - проводимости). Как это делается, можете узнать в Интернете или специальной литературе.



Рассмотрим простейшую модель биполярного транзистора. Два толстых слоя по края называются коллектором и эмиттером. Тонкий слой между ними называется базой. Тип проводимости коллектора и эмиттера одинаковый. Тип проводимости базы другой. Между коллектором и эмиттером идет мощный поток заряженных частиц. На базу подается слабый переменный потенциал. Этот слабый переменный потенциал регулирует мощный поток так, что мощный поток изменяется в зависимости от изменений слабого потенциала. Современные компьютеры сделаны на основе полупроводников. Из полупроводников можно сделать схемы с очень маленьким размером элементов, поэтому мы своими глазами можем видеть, что с каждым размеры электронных приборов ощутимо уменьшаются. Кроме того, при массовом производстве себестоимость производства полупроводниковой техники очень резко падает, что позволило многим покупать дешевую электронную технику.

Оптоэлектронные схемы

В оптоэлектронных усилителях используется свойство изменения прозрачности в одном направлении в зависимости от светового потока в другом направлении.



Таким образом, сильный луч изменяется в зависимости от того, как изменяется слабый луч.

Однако, в настоящее время оптоэлектроника пока не получила широкого применения, так как скорость изменения степени прозрачности у найденных наукой материалов пока очень мала.

Сверхпроводниковые схемы

В специальной литературе вы можете ознакомиться с тем, как на основе явления природы – сверхпроводимости и эффекте Джозефсона можно построить компьютер. Скажем только, что сейчас пока такие компьютеры очень дорогие, поэтому широкого применения не нашли, но в будущем, возможно, будут использоваться широко.

Гидравлические схемы

Существуют и экзотические возможности усиления сигнала, например, маломощный поток воды в специально сделанном устройстве может управлять потоком воды большой мощности. В связи с высокой стоимостью и большими размерами такие усилители для вычислительной техники в принципе можно сделать, но будет невыгодно.

Действия в программе

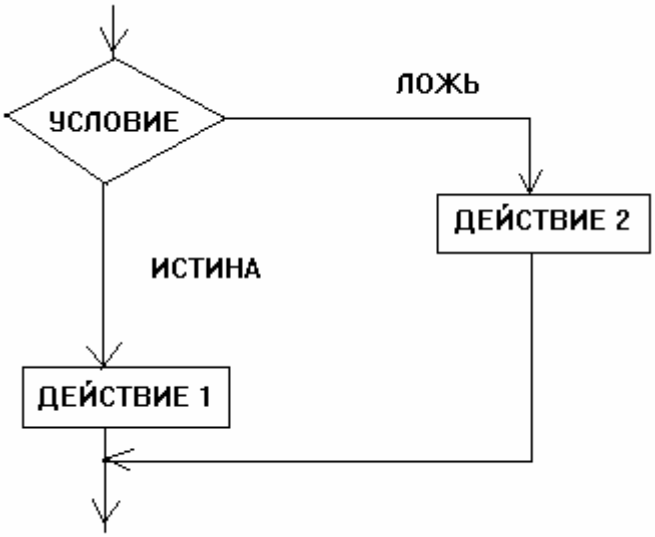
Наконец, наступило время поподробнее рассмотреть императивную часть программ.

Линейные действия

Вы уже встречались с последовательными линейными действиями, выполняемыми одно за другим, если нет аварийного завершения программы.
Познакомим вас еще с двумя важными видами действий.

Ветвление

Во-первых, это ветвление.

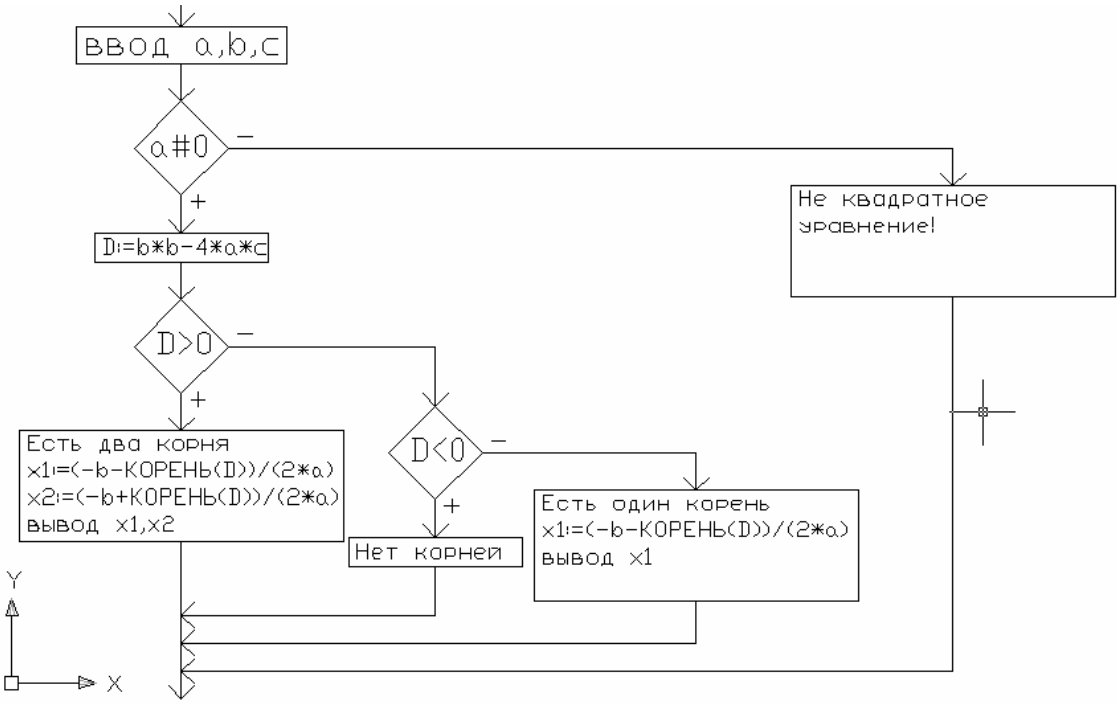


Условие – это логическое выражение, переменная или константа. Если условие истинно, то выполняется действие 1. Если условие ложно, то выполняется действие 2. Мы показали схему с указанием так называемого «царского пути», главной линии действия. Этот термин введен Пароджаевым в его свободно доступных по Интернету книгах. Главная линия алгоритма для повышения его наглядности идет вертикально сверху вниз.

Действие 1 может быть единичным или представлять собой несколько простых действий. То же верно для действия 2.

В сообщении о языке можете ознакомиться с описанием оператора IF.

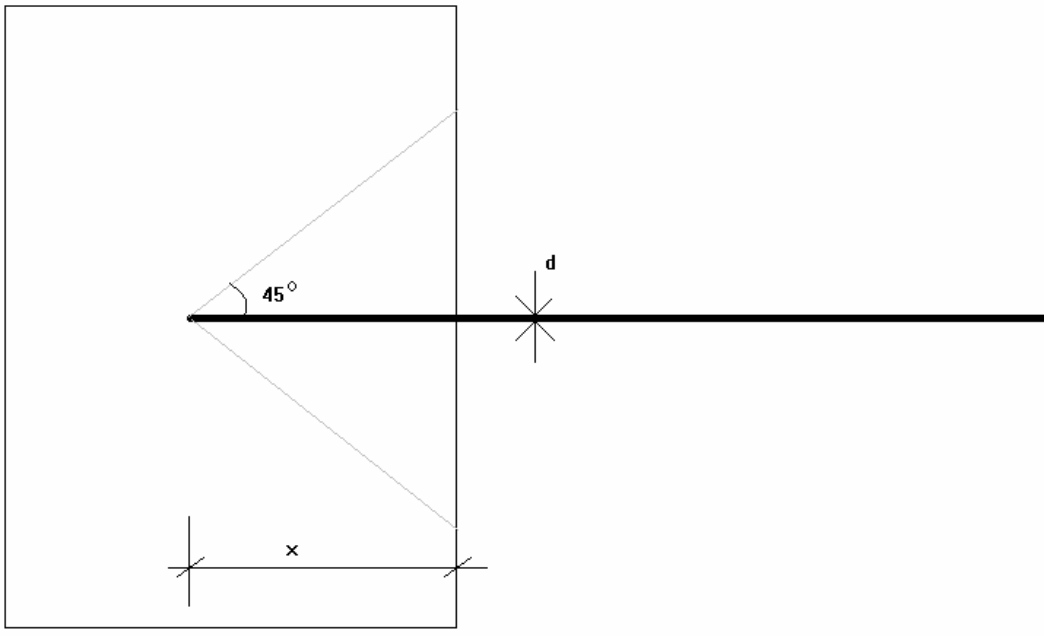
Рассмотрим хорошо известный пример с решением квадратного уравнения через дискриминант. Этот способ использовал еще Аль-Хорезми (проверить!), от имени которого произошло слово АЛГОРИТМ. Для уравнения $ax^2+bx+c=0$ при известных a,b,c надо найти значения x.




```
MODULE ProgbookIf;
        IMPORT  Log := StdLog,Math;
        VAR
                a*,b*,c*:REAL;

PROCEDURE Solve* ();
```

```
        VAR  D,x1,x2:REAL;
BEGIN
        IF  a#0  THEN
                D:=b*b-4*a*c;

                IF  D>0  THEN
                        Log.String('Есть два корня');Log.Ln;
                        x1:=(-b-Math.Sqrt(D))/(2*a);
                        x2:=(-b+Math.Sqrt(D))/(2*a);

                        Log.String('x1=');
                        Log.Real(x1);
                        Log.Ln;

                        Log.String('x2=');
                        Log.Real(x2);
                        Log.Ln
                ELSE
                        IF  D<0  THEN
                                Log.String('Нет корней');Log.Ln;
                        ELSE
                                Log.String('Один корень');Log.Ln;
                                x1:=(-b-Math.Sqrt(D))/(2*a);

                                Log.String('x1=');
                                Log.Real(x1);
                                Log.Ln
                        END
                END

        ELSE
                Log.String('Уравнение не квадратное');Log.Ln;

        END
END Solve;

END ProgbookIf.
```

Зачем Аль Хорезми понадобилось решать квадратное уравнение? Многие люди задаются вопросом, а зачем нам конкретно нужно уметь решать квадратное уравнение. Вот пример того, как техническая задача – определение длины анкеровки арматурного стержня в бетоне, сводится к квадратному уравнению, а корень этого квадратного уравнения есть эта длина. Длина анкеровки рассчитывается из условия то, что конический кусок бетона не должен оторваться от основного бетона при выбранной длине анкеровни.



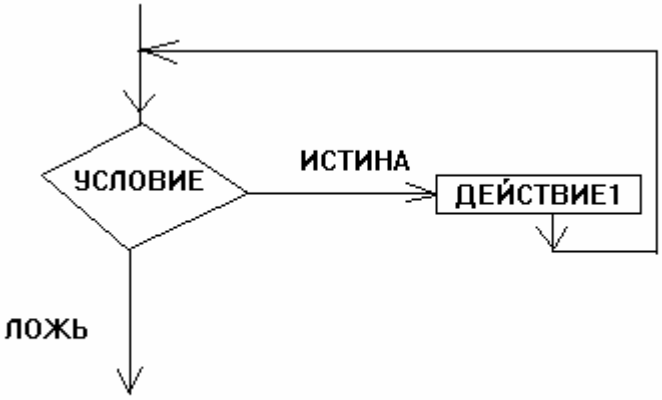
```
MODULE MetroShaibu;
	IMPORT  con := StdLog,  Math;

CONST
	pushinka=0.0001;
	Rs=3720; (*кг/см2*) (*уточнить класс арматуры*)
	Rbt=16.3; (*кг/см2*) (*уточнить класс бетона*)

VAR
	d*:INTEGER; (*в мм*)

PROCEDURE Qvur (a,b,c:REAL;VAR kornei:INTEGER;VAR x1,x2:REAL);
	VAR  D:REAL;
BEGIN
	D:=b*b-4*a*c;

	IF  D<0  THEN
		kornei:=0
	ELSIF (ABS(D)<pushinka) THEN
		kornei:=1;
		x1:=-b/(2*a);
		x2:=x1
	ELSE
		kornei:=2;

		x1:=(-b-Math.Sqrt(D))/(2*a);
		x2:=(-b+Math.Sqrt(D))/(2*a)
	END

END Qvur;


PROCEDURE Считай* ();
	VAR  As:REAL;
	kornei:INTEGER;
	x1,x2:REAL;
BEGIN
	As:=Math.Pi()*(d/10)*(d/10)/4;

	(*все в сантиметры*)
	Qvur (1,4*d/10,  (d/10)*(d/10)-4*As*Rs/(Math.Pi()*Rbt),kornei,x1,x2);
	IF  kornei>0  THEN
		IF  x1>0  THEN
			con.String('x1=');
			con.Tab;
			con.Real(x1);
			con.Ln
		END;

		IF  x2>0  THEN
			con.String('x2=');
			con.Tab;
			con.Real(x2);
			con.Ln
		END
	ELSE

		con.String('нет решения!');con.Ln;
	END;

	RETURN
END Считай;

END MetroShaibu.
```

Видимо, перед Аль Хорезми, возможно, встала в свое время задача из техники, которая решалась квадратным уравнением.

В сообщении о языке можете также ознакомится со сложными видами ветвлений с использованием ключевых слов ELSIF, CASE, но для начала простого ветвления IF-THEN-ELSE достаточно для работы.

Цикл

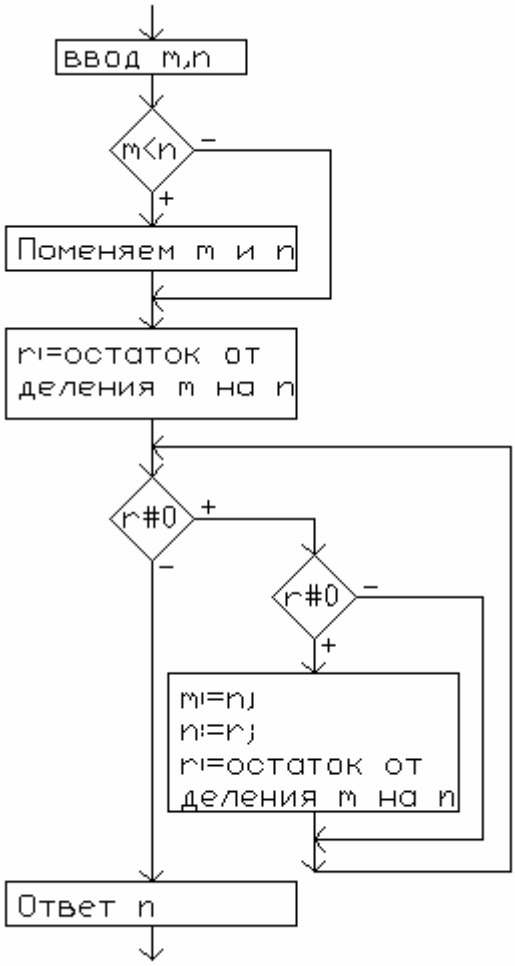
Другим важным видом действий является цикл. Цикл состоит из проверки условия и повторяемых действий.



Рассмотрим цикл с условием в начале (цикл WHILE). Вначале проверяется условие, если оно истинно, выполняется действие1, иначе цикл завершается. После выполнения действия1 условие проверяется заново и так до тех пор, пока условие не станет ложным. Действие1 может быть единичным действием или набором действий. Здесь мы схему цикла также дали в стиле Паронджаева с указанием «царского пути».

В сообщении о языке можете ознакомиться с описанием оператора WHILE.

Рассмотрим пример – алгоритм Евклида для поиска наибольшего общего делителя.




```
MODULE ProgbookWhile;
	IMPORT  Log := StdLog, Dialog;
	VAR
		m*,n*:INTEGER;

PROCEDURE Euklid* ();
	VAR  p,r:INTEGER;
BEGIN
	IF  m<n  THEN
		p:=m;
		m:=n;
		n:=p;
		Log.String('Поменяли значения ');
		Log.Ln;
		Log.String('m=');
		Log.Int(m);
		Log.Ln;

		Log.String('n=');
		Log.Int(n);
		Log.Ln;

		Dialog.UpdateInt(m);
		Dialog.UpdateInt(n);
	END;

	r:=m-(m DIV n)*n; (*формула для поиска остатка*)

	Log.String('r=');
	Log.Int(r);
	Log.Ln;

	WHILE  r#0  DO
		IF  r#0  THEN
				m:=n;
```

```
                                n:=r;

                                Log.String('Обновили значения ');
                                Log.Ln;
                                Log.String('m=');
                                Log.Int(m);
                                Log.Ln;

                                Log.String('n=');
                                Log.Int(n);
                                Log.Ln;

                                r:=m-(m DIV n)*n; (*формула для поиска остатка*)

                                Log.String('r=');
                                Log.Int(r);
                                Log.Ln;

                        END
                END;

                Log.String('Ответ ');
                Log.String('n=');
                Log.Int(n);
                Log.Ln;


                RETURN
END Euklid;


END ProgbookWhile.
```

Задание на самостоятельную работу – попробуйте представить себя в роли компьютера и выполнить алгоритм Евклида с разными исходными значениями.

В сообщении о языке также в будущем можете ознакомиться с другими видами циклов, но для начала цикла WHILE достаточно для работы.

Ограничения, о которых надо всегда помнить

Есть хорошая английская пословица «forewarned is forearmed» - «кто предупрежден, тот вооружен».

Ограничения компьютера

Ограничения отображения мира в типах данных

Всегда надо помнить, что целые и вещественные числа в компьютере отличаются от целых и вещественных чисел в арифметике и алгебре.

Как мы уже говорили, у целых и вещественных чисел в компьютере есть максимальное и минимальное значения. Алгоритмы надо строить так, чтобы значения переменных не выходили за эти границы.

Кроме того, как это уже говорилось ранее, надо помнить об ошибках округления при сложении и вычитании вещественных чисел.

Кроме того, при преобразовании одного целого к другому, например, INTEGER в SHORTINT может произойти выход значения за пределы допустимых. Эти целые описываются разным количеством байтов.

То же справедливо в отношении вещественных чисел.

Кроме того, ошибка в общем случае может произойти при переходе от беззнакового целого к знаковому, описываемые одинаковым числом байтов.

В общем эти правила просты, но их несоблюдение может привести к большим проблемам.

Ограничения объема хранимой и обрабатываемой информации

Ограничения объема оперативной памяти и места на жестком диске связаны с конкретными компьютерами, на которых исполняется программа.

Рано или поздно программист сталкивается с нехваткой памяти. Здесь можно провести следующую аналогию – большой кусок хлеба сразу трудно проглотить, но если откусывать по кусочкам и жевать, то рано или поздно этот кусок хлеба можно съесть. Например, мультифронтальный метод решения систем линейных алгебраических уравнений позволят на том же компьютере решить системы уравнений такого большого размера, на которых обычный метод Гаусса надолго зависает.

Есть еще и такой термин – **нормализация**. То есть ту часть информации, которая часто повторяется, можно заменить короткой меткой и составить таблицу таких меток, таким образом можно сжать информацию.

В зависимости от природы данных существуют разные методы сжатия информации. Например, есть данные, которые не требуют точного воспроизведения, если после разжатия результат будет немного изменяться от того, что было в начале, то это не страшно по условию задачи. Это позволяет для таких данных построить еще более эффективные методы сжатия. Такое сжатие, например, используется для фотографий, музыки, видео.

Ограничения скорости обработки информации

Ограничения скорости обработки информации тоже можно отнести к конкретной машине. Как бы быстро ни работал компьютер, всегда найдется задача, которая с первого раза будет обрабатываться медленно, очень медленно.

В некоторых случаях можно скорость повысить, исключив многократно выполнение того, что уже сделано.

Если задать приоритет скорости, то объем расходуемой памяти может увеличиться, и наоборот, если задать приоритет на объем памяти, скорость может немного уменьшиться. Что именно важно – определяется Делом, решаемым при помощи компьютера.

Если совсем не получается ускорить программу, никакими ухищрениями, то можно использовать параллельные вычисления, о которых в примерах будет сказано.

Ограничения человека

Программа – товар. Поэтому надо думать об удобстве ее использования. Делайте программу так, чтобы человеку, использующему ее и человеку, дорабатывающему ее, все было просто и понятно.

Сколько может человек запомнить и как преодолеть этот барьер

Главное ограничение, о котором надо помнить всегда – умному человеку трудно оперировать одновременно более, чем 7 независимыми мыслями. Способ преодоления этого ограничения – использовать зависимости между мыслями, сознание структуры. Пример структуры – разделение страны на области, районы. Маленький город управляется целиком, большой город делится также на районы. Практическое применение этого приема – разбивка структур данных на записи, разбивка программы на модули и процедуры, об этом будет сказано дальше.

Менее умным людям трудно оперировать одновременно более, чем тремя независимыми мыслями, но таким людям лучше заняться другим видом деятельности, так как им будет труднее конкурировать с другими.

Еще раз о невозможности построения искусственного интеллекта или о том, какие задачи можно поручать компьютеру

В результате развала Советского Союза народными массами овладела идея о том, что марксистко-ленинская коммунистическая идеология неверна в корне. А вместе с тем тень недоверия легла на атеизм, лежащий в основе этой идеологии. Заслугой Карла Маркса является то, что он единственный смог построить целостную картину мира на основе атеизма в форме диалектического и исторического материализма. Все остальные картины мира, построенные на атеизме, являются ущербными и недостаточными для построения идеологии. Альтернативные существующие идеологии прямо или косвенно подразумевают наличие Творца.

Капитализм косвенно признает Творца, но призывает отделить это признание от жизни, а те, с кем сейчас капиталисты усиленно воюют, прямо признают Творца и призывают жить по Его законам, а не придумывать отсебятину, так как человек-законодатель всегда ограничен своим временем, пониманием задачи и, возможно, корыстью, и поэтому все придуманные человеком законы общества со временем принесут страдания другим людям. Например, страдания угнетенных.

Коммунисты и сочувствующие им лица думали, что человеку, как высокоорганизованной форме материи посильно создать другого человека с нуля (рождение человека естественным путем здесь не имеется в виду) при достижении соответствующей степени организации искусственной системы. А что отличает человека от других наблюдаемых объектов материального мира? Разум. Значит, полагали коммунисты и сочувствующие им лица, возможно построение системы искусственного интеллекта, разумных роботов.

Многие силы были брошены на решение этой задачи и напрасно. В бесплодных попытках решения задачи построения искусственного интеллекта были получены другие косвенные результаты – язык программирования LISP (tm), библиотека кнопок и других элементов управления wxWidgets и другое. А искусственный интеллект никак не получался. Хороший пример – программы- переводчики, например, с английского на русский. Как известно, переводят не слова, а смысл текста. Вот смысл компьютер и не в состоянии уловить. Поэтому все программы – переводчики давали и будут давать смешной результат перевода. Всегда.

Человек не может сотворить другого человека. Человек не может сотворить Разум. Поэтому запрограммировать можно только те задачи, которые имеют формализованное описание. То есть, те задачи, которые возможно разбить на простые действия, ветвления и циклы. И все. Ничего другого не стоит даже пытаться реализовать. Бизнес-план, основанный на решении задачи построения искусственного интеллекта всегда будет убыточным.

Небольшое отступление и напутствие

Это уже не совсем касается программирования, но для общего кругозора полезно. Карл Маркс совершил несколько ошибок.

1. Мышление – это свойство не просто высокоорганизованной материи. Это свойство здорового и зрелого человека (из всех материальных объектов, наблюдаемых нами). И как бы высоко мы ни организовывали материю (компьютеры, компьютерные сети), мыслить она никогда не будет.
2. Мышление – это не просто отражение материального мира. Отражение скорее характерно для неживой природы. Человек – существо живое. Для мышления необходимо еще два фактора – передача информации от материального мира мозгу через здоровые органы чувств и предварительная информация в сознании человека, в зависимости от которой будет разный результат.

3. Исходя из указанного в п.2, человек не может своим мышлением познать неуничтожимость материи в будущем, так как будущее – за пределами наших органов чувств. Следовательно утверждение «материя неуничтожима» является не доказанной теорией, а гипотезой, следовательно, на основе этого НЕЛЬЗЯ строить картину мира. Следовательно, ошибка в теории познания привела к ошибочному диалектическому материализму.
4. Четвертая ошибка Карла Маркса в том, что на основе ошибочного диалектического материализма он построил ошибочный исторический материализм. Он рассматривал только один фактор – соотношение производительных сил и производственных отношений. В итоге проблему капитализма он видел только в частной собственности на средства производства. На самом деле проблема капитализма – в секуляризме, отказе от доминировавшего ранее религиозного воззрения, что привело к тому, что отношения между людьми определяются не религиозной моралью, а выгодой, а правила морали следуют только из выгоды, что привело к беспощадной эксплуатации пролетариата буржуазией. Попытки царей привить западный капитализм в дореволюционную Россию начиная с освобождения крепостных с 1861 году также привели к тому, что отношения между классами в России, до этого в какой-то мере опиравшиеся на религиозную нравственность, стали основываться только на выгоде и привели к беспощадной эксплуатации пролетариата и крестьянства, что создало условия для революции.

Ленин далее развил эту ошибочную теорию и предложил на время оставить только одного собственника – социалистическое государство. Сталин и Берия, отлично владея экономической наукой, действительно построили одно крупнейшее предприятие в мире – СССР. Однако со временем ошибки Маркса привели как минимум, к трем причинам, разрушившим его.

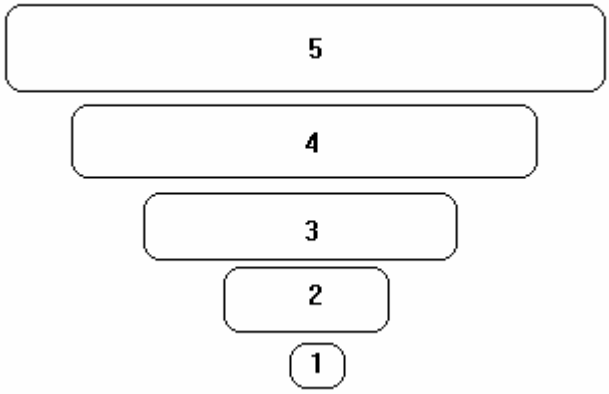
1. Здесь есть еще и ошибка Ленина на основе ошибки Дарвина. Человек остается таким же, каким и был. Человеческая жадность, лень, основанные на инстинкте самосохранения, неистребимы, сколько бы лет при социализме ни жил человек. Запрет на владение частной собственностью и позднесоветская уравниловка после смерти Сталина привели к тому, что большинство людей перестали работать хорошо. Это привело ко все большему числу убыточных предприятий, дотируемых за счет продажи за рубеж природных ресурсов. А когда цены на природные ресурсы в мире упали, это государство-предприятие перестало окупать себя и развалилось.
2. Советские математики под руководством академика Глушкова доказали, что в мирное время задача управления единым предприятием-государством превышает возможности компьютеров тех лет. Она превышает возможности и современных компьютеров. Система стала трудноуправляемой и близкой к неуправляемой. Подробней об этом можно узнать в Интернете.
3. Задача управления единым государством-предприятием успешно решалась, когда руководители (Сталин, Берия) хорошо знали экономическую науку. Но и они, начиная со времен Ленина, были вынуждены использовать аппарат контроля – коммунистическую партию. Верхушка этого аппарата со временем обюрократилась, так как были только права и не было ответственности за работу предприятий. Со

временем желание удовлетворить собственные потребности и желание роскоши пересилило в них коммунистическую идеологию, и им стало выгодней развалить свое государство, что они и сделали. И сейчас они наживаются, в основном, за счет продажи природных ресурсов. И в том числе поэтому население России сокращается, так как население в данном случае – только потребитель природных ресурсов, чем его больше, тем на большее число придется делить прибыль от продажи природных ресурсов.

Поэтому, уважаемый читатель, изучайте программирование, изучайте эффективные способы построения предприятий, создайте высокотехнологичную альтернативу доходам от продажи сырьевых ресурсов, думайте, и только таким образом вы сможете сохранить свой народ. И не забывайте думать о том, как обустроить жизнь справедливо, опираясь на правильную картину мира.

Как делать большие проекты

Как писал Генрих Саулович Альтшуллер в свободно доступной по Интернету книге «Как стать гением», если у Вас есть великие идеи, то их реализацию надо выстроить в следующем порядке.



Отсортируйте свои идеи по сложности реализации. Сначала беритесь за те, реализация которых требует минимума расходов времени и труда. Потом постепенно переходите к более сложным проектам. И так до самого сложного. Если вы сразу возьметесь за очень сложный проект, то прежде чем от его использования вы начнете получать выгоду, у Вас могут кончиться деньги.

Есть еще одна проблема второго проекта, о которой говорится в книге Брукса «Мистический человеко-месяц или как разрабатываются программные продукты», которую можно найти в Интернете. Пройдя первый этап, человек имеет вероятность возгордиться, переоценить свои силы и на втором этапе взять слишком сложное и зачастую непосильное задание.

Для того, чтобы составить хороший план, можете воспользоваться системой GTD (Get iT Done). Подробное описание этой системы также есть в Интернете. Скажем коротко и ясно: носите постоянно с собой ручку и листок бумаги. Идея может прийти в любой, даже самый неподходящий момент. Сразу записывайте самую суть кратко. Иначе забудете, потом трудно будет вспомнить. Когда будет время, оформляйте свои идеи и прячьте до реализации. Но не затягивайте с оформлением, так как у Вас может собраться неразгребаемая куча бумажек.

Вы хотите стать профессиональным программистом. Профессионал в каком либо деле – это человек, зарабатывающий свой хлеб насущный этим делом. Поэтому прежде,

чем затеять какой либо проект, для Вас всегда должно быть предельно ясно, как из этого проекта вы будете получать выгоду (материальную, духовную или и то и другое).

Как правильно писать документацию

Любая качественная программа требует документирования. Написание сразу качественной документации – довольно сложная задача для человека. Однако, если мы воспользуемся методом великого русского писателя Льва Николаевича Толстого, то написание документации станет довольно простым занятием.

1. Пишем мысли в черновом варианте.
2. Сортируем записанные мысли.
3. Работаем над стилем.

После написания документации надо ее обязательно редактировать. По выражению одного моего знакомого журналиста и редактора, «редактор – это тупой читатель». Просмотрите написанное Вами с такой точки зрения, и сразу станет ясно, что надо исправлять.

Ограничения задачи

Чаще всего ограничения задачи используют при постановке самой задачи. Здесь надо хорошо знать саму область, для которой вы собираетесь писать программу.

Структура программы

На начальном периоде использования компьютеров в моде были так называемые «простыни» - большие распечатки программ длиной в несколько метров. Действительно, с распечатки читать удобней, чем с экрана, на ней больше информации, компьютер при этом не шумит, можно не спеша находить и исправлять ошибки. С другой стороны, программ стало писаться намного больше, распечатывать весь код не хватит леса на планете. И тогда стали менять системы программирования – делить большие куски программного кода на много маленьких, каждый из которых можно посмотреть с экрана. Особенность «простыни» (длинных кусков кода) в том, что пока вы дочитаете ее до конца, вы можете забыть, что было в начале. А маленький кусок кода целиком можно уместить в голове и обдумать. Как управлять народными массами? Надо структурировать. Как съесть большую лепешку? Надо отламывать и откусывать по размеру рта. Так же и управляем кодом. Делим на куски, каждый кусок соединен определенным образом с другими.

Процедуры

С процедурами вы уже сталкивались в примерах. Процедура это и есть самый маленький упакованный кусок императивной части кода со своей декларативной частью.

Одна процедура может вызывать другую. Если рядом с именем процедуры стоит знак “*”, то ее можно вызывать извне.

Процедура может быть рекурсивной, то есть вызывать саму себя.

(вставить пример с сортировкой)

У процедур могут быть свои константы и переменные. Если имя константы или переменной совпадает с именем константы или переменной модуля (о модулях- далее), то все действия внутри процедуры с этим именем сводятся к действиям с переменными и константами, объявленными внутри процедуры.

(вставить пример с локальной и модульной переменными)

При вызове процедура может принимать параметры и возвращать параметры.

Ознакомьтесь с пунктом «**Описания процедур**» в Сообщении о языке.

Процедуры нужны для того, чтобы код был не виде бескрайней простыни, а в виде мелко нарезанных кусков, достаточных, чтобы понять с одного чтения.

Модули

Модуль – это минимальный кусок исходного кода, написанный на языке программирования, который можно перевести в машинный код.

В модуле может быть несколько процедур и другие данные. Исходный код – это код, легко читаемый человеком, но напрямую не выполняемый компьютером. Он написан строго по определенным правилам, которые позволяют специальной программе разобрать его по полочкам.

Машинный (объектный) код – это код, воспринимаемый машиной к исполнению. Процесс перевода исходного кода в машинный называется компиляцией или трансляцией.

Наступило время раскрыть, что же означает «доступ извне». Это доступ из другого модуля. Программа делится на модули, как устройство на блоки, эти модули соединяются между собой. Хорошо разработанные модули можно использовать в разных программах, таким образом экономить время.

Ознакомьтесь с пунктом «**Модули**» в Сообщении о языке.

(вставить пример с несколькими модулями)

Составные структуры данных

Записи

Запись это составная переменная. Она состоит из полей. Поля записи могут быть переменными простого типа (целые, вещественные, символы, логические, множества) или составного типа (запись, массив, указатель).

Записи нужны для того, чтобы группировать связанные смыслом программы переменные в единое целое и затем работать с этим единым целым. В записи может присутствовать несколько полей разных типов. Ознакомьтесь с пунктом «**Типы записей**» в Сообщении о языке.

Вот пример с записями.

(вставить пример с записями)

(Привести пример с использованием записи в Dialog.Update)

Массивы

Массив это тоже составная переменная. Он состоит из элементов. Элементы записи могут быть переменными простого типа (целые, вещественные, символы, логические, множества) или составного типа (запись, массив, указатель).

Все элементы массива – одного типа. Они отличаются номером (индексом). Массивы могут быть одномерными (с одним индексом) и многомерными (с несколькими индексами)

Массивы нужны для того, чтобы много одинаковых кусочков данных можно было хранить и обрабатывать единым способом, доставая их через номер элемента. Если размер массива задан в тексте программы, то такой массив называется статическим.

Ознакомьтесь с пунктом «**Типы массивов [array types]**» в Сообщении о языке. Будьте осторожны с массивами. При помощи массива можно сразу занять очень большой кусок памяти, так что свободная память может закончиться.

Вот пример с массивами.

(вставить примеры с массивами)

Указатели на записи и массивы. Повышение возможности соответствия структуры данных задаче

Указатели на массивы позволяют задавать размер массива во время работы программы. Такие массивы называются динамическими массивами. Изменение размеров существующего динамического массива означает удаление существующего с потерей содержимого и создание нового.

Указатели на записи позволяют распределять память для элементов записи также во время работы программы. Указатель на запись можно определить заранее, до определения самой записи, и таким образом, элементом записи может быть указатель на саму эту запись или указатель на запись, в который вложена напрямую или через еще один указатель наша запись. Это позволяет делать структуры данных, размер которых может меняться во время выполнения программы с сохранением содержимого: списки, деревья, кольца. (ДАТЬ ПРИМЕР СО СПИСКОМ ПО ВЫБОРКЕ УНИКАЛЬНЫХ ЗНАЧЕНИЙ)

Сборщик мусора отслеживает работу указателей, автоматически освобождает занятую указателями память, когда эти указатели становятся нулевыми или исчезают, например, с завершением процедуры, внутри которой они были определены. Сборщик мусора тратит на это время и память компьютера. Поэтому когда динамические структуры – списки и др. становятся слишком большими, на их обслуживание уходит слишком много времени и ресурсов, программа может даже зависнуть. Поэтому везде, где можно, минимизируйте применение динамических структур типа списков, деревьев, колец, старайтесь заменять их динамическими массивами. Иногда два раза пройтись по данным (первый раз для определения размера динамического массива, второй раз для заполнения элементов массива) обходится дешевле, чем работать со списком. Для динамической структуры, как список, хорошее применение – выборка уникальных значений. Деревья применяются часто в компиляторах.

При разработке достаточно сложных программ очень важно создать структуру данных, адекватную решаемой задаче. Указатели на массивы и записи позволяют создать такую структуру. Массив может быть вложен в запись, а запись может быть вложена в массив. Таким образом можно создать разветвляющуюся и масштабируемую структуру. Масштабируемость в данном случае реализовывается через возможность задавать любой размер динамического массива. Разветвляемость реализуется через то, что в записи могут быть разнородные члены. При этом возникает одновременно и проблема и возможность сэкономить – для того, чтобы оперировать с такой структурой (сохранять на диск, считывать с диска и т.п.), надо быть очень внимательным. Если структура меняется, все изменения надо вносить в процедуры по чтению-записи. Эта внимательность, как очевидна, опять присуща скорее машине, чем человеку, поэтому ряд задач можно смело переложить на плечи машины.

Отсылаем Вас к рекламе услуг по программированию в конце книги, там есть очень недорогая услуга, позволяющая значительную часть скучной работы заменить работой компьютера. Обычно на обслуживание структур данных уходит значительная часть времени программистов, а уникальных алгоритмов приходится делать не так уж много. Использование самого компьютера для таких нудных работ сильно повышает производительность труда программистов. Реклама таких услуг покрывает расходы на подготовку этой бесплатной для некоммерческого применения книги.

В других языках есть указатели на переменные, но такие указатели занимают почти столько же памяти, сколько сами динамически адресуемые переменные, поэтому в Компонентном Паскале такую возможность не включили.

Кроме того, в некоторых других языках есть так называемая адресная арифметика, то есть со значением указателя на массив можно выполнять арифметические действия.

Это может привести к тому, что указатель выйдет за пределы динамического массива и данные из совсем другой области (другие переменные или даже код) в битовом виде будут пониматься программой, как элементы массива, могут быть изменены, как элементы массива, а потом, в своей соответствующей части программы они станут переменными другого типа или даже исполнимым кодом, но их содержимое будет испорчено и программа поведет себя труднопрогнозируемым способом, так как то, какие переменные или даже код будут расположены рядом с нашим массивом, определяется разработчиками компиляторов и не оговаривается стандартами. Но если программа уже жестко откомпилирована, то специально изучив ее можно воспользоваться такой ситуацией. Люди, специально пытающиеся взломать программы, подготавливают такой набор данных для атакуемых программ, использующих адресную арифметику, которые приводят в таким ошибочным ситуациям и записывают в область кода такой код, который нужен злоумышленнику.

Таким образом, если вы пишите свою программу на Компонентном Паскале, такая ситуация Вашей программе не грозит, так как адресной арифметики в нем нет. Если вы пишете программу на других языках, использующих адресную арифметику, просто откажитесь от таких возможностей языка.

Зачем нужно и не нужно объектно-ориентированное программирование

Объектно-ориентированное программирование придумали для того, чтобы экономить время программистов и иметь возможность использовать повторно старые написанные программы в тех случаях, когда обычные процедуры уже не помогают этого сделать. Используйте объектно-ориентированное программирование только в таких случаях. Некоторые экстремисты стараются использовать объектно-ориентированные подход везде, однако это не всегда дает хороший результат. Программы, написанные с объектно-ориентированным подходом

1. Не всегда так просты и очевидны, как программы с использованием просто процедур, поэтому дают в семь раз больше ошибок, чем обычные процедурные программы.
2. Иногда делают много лишних ненужных действий, размер и скорость работы программ увеличиваются.

Минусов, как видите, два. Плюс от объектно-ориентированного программирования один - в экономии времени. Однако сам факт использования объектно-ориентированного подхода экономии не дает, экономию дает умелое применение этого подхода в зависимости от задачи.

Поэтому принимайте решение об использовании объектно-ориентированного подхода только если экономия времени разработки программы будет существенна.

В Интернете очень много информации об объектно-ориентированном программировании. В сообщении о языке с эти связаны **расширение и наследование типов**, а также **методы**.

Вот пример с объектно-ориентированным подходом.

(вставить простой пример)

Что делать дальше.

Теперь надо полученные знания закрепить примерами. Вы изучили основные понятия и простейшие приемы, теперь надо отработать умения и навыки их применения для разных задач. Как это делать? Вспомним слова Никлауса Вирта, великого разработчика языков программирования. Он сказал, что «Программирование – это

конструирование». Представьте себе, то, что вы уже изучили – это элементы детской игры «Конструктор». Теперь надо собирать из элементов разные игрушки. Как это делать? Включайте фантазию.

Когда хорошо усвоите на практике тот минимум информации, который мы вам дали, можете ознакомиться уже подробно с Сообщением о языке.

Практика. Построение файла чертежей формата AutoCAD (tm) DXF

Есть такая система для построения чертежей. Она называется AutoCAD (tm). Одной из причин ее широкой популярности является то, что ее авторы открыли всему миру, как хранятся данные о чертеже. У этой системы есть открытый формат текстового файла с расширением DXF. Документация по структуре DXF файла дана в самой системе AutoCAD(tm). Здесь мы вкратце скажем о самом главном. Файл DXF состоит из дуплетов и кусков. Дуплет – это две строки. В первой строке стоит десятичное число, написанное цифрами, они называют его кодом. От значения этого числа зависит, какой тип присваивается значению во второй строке и что вторая строка означает.

Набор дуплетов, начинающийся кодом 0, за которым следует другой дуплет с кодом 0, назовем куском. О кусках в документации системы AutoCAD(tm) явно не говорится, но подразумевается. Данные в дуплетах, принадлежащих одному куску, связаны между собой. Это могут быть, например, для линии – координаты начала и конца, наименование слоя, в котором эта линия лежит, цвет линии, толщина и др. Вот пример с тремя линиями.

(вставить пример dxf файла)

Откройте его в системе AutoCAD(tm) и сверьте эти характеристики линий, которые вы там увидите (выбрать одну линию, правой кнопкой вызвать локальное меню, выбрать пункт «Properties», в окне свойств проверить соответствующие характеристики).

А вот и пример программы, в которой мы этот dxf файл с нужным нам рисунком генерируем.

(вставить пример генерации dxf файла)

Для вывода текстового файла мы подготовили небольшой модуль.

(вставить модуль вывода текстового файла)

Практика. Построение простейшего движка для web сайта

Сделаем для примера статический движок. Разобравшись с ним, а также базами данных, вы сами сможете сделать динамический движок.

Допустим, у вас есть несколько фотографий. Надо по ним сделать веб - страницы так, чтобы на каждой странице была одна фотография, на самом верху и в самом низу что-то нужное вам, а также линейка с ссылками для перехода с одной страницы на другую. Пусть надо еще сделать так, чтобы информацию на самом верху и в самом низу веб - страницы было легко менять. Под каждой фотографией пусть надо поместить свой текст.

Фотографии пусть лежат в папках. Для каждой папки надо создать альбом. Для переходов по альбомам тоже нужны ссылки. В папке с фотографиями будет файл с пояснениями по альбому. Рядом с каждой фотографией тоже свой текстовый файл с пояснением. Эту систему после незначительной переделки можно будет использовать не только для фотографий, но и для музыки, видео и т.д.

Что такое язык html, вы можете узнать в Интернете. Здесь мы покажем кое-что, о чем не было написано до этого.

Замечательный российский программист (уточнить имя) в свое время разработал систему генерации отчетов FastReport. Мы позаимствуем оттуда одну идею. А суть его идеи в том, что отчет делится на полосы. На этих полосах есть поля, куда помещаются

данные из программы, например, из базы данных. Макет полос отделен от программы, изменять его может пользователь. Есть полосы разных видов. Каждая полоса может повторяться ноль и более раз.

Мы тоже подготовим свои полосы – куски текстовых файлов. В этих полосах будут поля, заполняемые программой. Сколько раз каждая полоса будет повторена, пусть зависит от программы.

Практика. Технология COM. Передача компьютеру части работы по внешней обработке документов программ AutoCAD (tm), Word (tm), Excel (tm)

Для начала коротко о пакетном и диалоговом режимах работы программ.

Практика. Технология COM. Передача компьютеру части работы по работе с документами внутри программ AutoCAD (tm), Word (tm), Excel (tm)

Иногда внутри готовой программы надо сделать какой-то свой инструментик.

Практика. Работы с бинарным файлом изображений формата BMP

Работа с растрами через множества (SET)

Практика. База данных системы сетевого маркетинга. Хранение и обработка информации для многих компьютеров на одном компьютере

(о деревьях тоже, помянуть Джоя Целко)

Практика. Параллельные вычисления на кластерах. Решение одной большой задачи силами многих компьютеров

Наступает момент, когда ваша задача на одном компьютере, несмотря на все ухищрения, решается очень долго или вообще не решается. Значит, пришло время использовать параллельные вычисления. Что такое параллельные вычисления? Это когда одна задача делится на несколько задач и каждая из них выполняется на разных процессорах и они выполняются одновременно – как несколько автомобилей едут в одну сторону по параллельным дорожкам. Но как делить одну задачу на несколько? Тут надо подумать. Почти в любом алгоритме есть циклы. В некоторых случаях нет разницы, какой из проходов цикла выполнять первым. Вам надо найти такие циклы. Назовем их кандидатами на распараллеливание. Из этих кандидатов на распараллеливание найдем тот цикл, которые ближе всего к точкам запуска программы. Этот цикл лучше всего и распараллелить, то есть разные проходы исполнять на разных компьютерах. Все эти компьютеры вместе называются суперкомпьютером.

Как связывать между собой эти компьютеры? Из того, что у нас уже есть в наличии, мы можем использовать сетевые системы баз данных. В базах данных в простейшем сетевом случае есть главный компьютер – **сервер**, на котором хранится вся нужная информация, и второстепенные компьютеры – **клиенты**, с которых идет доступ к

данным сервера через сеть. Мы можем использовать эту же систему. На сервере можем хранить задания и список:

1. Какой клиент какое задание выполняет.
2. Какие задания уже выполнены.
3. Какие задания еще надо выполнить.

Свободные клиенты обращаются к серверу, смотрят, какие не выполненные задания еще есть, загружают данные по невыполненному заданию, помечают это задание своим, когда выполнят, пометят его, как выполненное. Вы можете написать такую систему сами, используя базы данных. Можете воспользоваться готовой системой BOINC (найдете в Интернете). У системы BOINC есть свои плюсы и минусы. Она предназначена для организации расчетов на очень большом числе компьютеров, подключенных к Интернету, в свободное от основной работы время, задания включаются через механизм хранителя экрана, если не прикасаться к компьютеру некоторое время. Каждая задача выполняется два раза. Так что эта система ориентирована на использование **чужих** компьютеров.

Если все используемые Вами компьютеры для параллельных вычислений **свои**, то система, основанная на BOINC, будет делать больше ненужной вам работы в силу своей специфики. Если вы воспользуетесь обычной базой данных для организации вычислений, как мы рекомендуем, то вам придется покупать или другим способом использовать для вашего суперкомпьютера меньшее число компьютеров. А вот и пример решения системы линейных алгебраических уравнений методом параллельных вычислений.

(вставить пример СЛАУ на кластере)

При организации параллельных вычислений как никогда важно равномерно распределить нагрузку на все используемые элементы суперкомпьютера, так как если какой-то элемент перегружен, остальные недогружены, это замедляет всю систему в целом. Как говорится, там где тонко, там и рвется.

Практика. Задачи метода конечных элементов. Описание природных явлений на компьютере

Оглянитесь вокруг. В чем разница между той техникой, что появилась за последние пятьдесят лет и с техникой, что была до этого? Правильно – важнейшая разница – в большом разнообразии приборов и устройств, появившихся в последнее время. Почему появилось так много техники? Как известно, любое устройство сначала формируется в виде мысли в разуме изобретателя. Потом в виде чертежей передается тому, кто это устройство может сделать. Потом это устройство надо проверить, а как оно поведет себя на самом деле? Такая проверка называется испытанием. Раньше все испытания проводили с самими новыми изделиями или их уменьшенными копиями. Если что-то работало не так, изделие переделывали и заново испытывали. Все это занимало очень много времени. Сейчас появилась возможность проводить большую часть таких испытаний не в натуре, а внутри компьютера до того, как это изделие сделано. Это намного удешевляет стоимость работ по подготовке изделия к производству и уменьшает необходимое для этого время. А как это получается? Люди накопили знания о природных процессах и сохранили их в виде алгоритмов. Сейчас мы постараемся разобраться с такими алгоритмами.

Есть так называемые уравнения математической физики, позволяющие языком математики описать сложные природные явления. Конечно, это описание не абсолютно точное, например, когда мы изучаем, как нагревается двигатель автомобиля, мы не учитываем движение каждого атома. Иначе время, затрачиваемое на решение такой задачи, было бы слишком большим, больше, чем мы можем себе это позволить. Поэтому мы применяем абстракцию – то есть, отбираем самые важные признаки природного явления и закрываем глаза на все остальное. Язык математики позволяет составлять уравнения. Как мы знаем, в уравнениях есть известные члены, неизвестные члены и связь

между ними через некоторые математические действия – сложение, вычитание, умножения, деление, равенство. Более сложные действия – степень, синусы, косинусы, логарифмы и так далее тоже в конечном итоге можно привести к простым действиям с нужной степенью точности.

Но есть еще связь другого рода. Связь, показывающая, как изменяется та или иная величина. Например, производные. Как изменяется положение автомобиля относительно города, откуда он выехал, с течением времени, если взять очень малый, настолько малый промежуток времени, в течение которого изменение расстояния автомобиля от города можно получить, умножив какое-то неизменяемое число на продолжительность этого промежутка времени, даже если нажимается педаль газа или тормоза. Это число, как всем известно, называется скоростью. Промежуток времени выбирается настолько маленьким, что скорость считается неизменной. Эта скорость для расстояния в математике, как известно, называется производной. В свою очередь, скорость тоже может меняться, и производной для нее является ускорение. Уравнения, в которых есть производные, называются дифференциальными уравнениями. Они позволяют сделать то, чего не позволяют обычные уравнения, не содержащие производных – очень точно описывать природные явления. Но и решать такие уравнения намного труднее. При помощи разума, бумаги, ручки и справочника можно решить только некоторые виды дифференциальных уравнений. Кроме обыкновенных дифференциальных уравнений, где изменение проходит только по одной величине, есть еще и дифференциальные уравнения в частных производных, где изменения могут быть по нескольким величинам, например, по разным направлениям и по времени. То есть, мы можем описать не только движение автомобиля, движущегося по ровной дороге в одну сторону, но и движение самолета, делающего крутые виражи, и движение воздуха, огибающего самолет, и то, как при этом трясется топливо в баках самолета. Только на бумаге все это очень и очень сложно решать, время решения может стать слишком большим; бумаги может уйти больше, чем есть на всей Земле, не говоря уже о терпении, забывчивости и сроке жизни того, кто решает, и о том, сколько денег уйдет, чтобы платить ему зарплату все это время.

Появление компьютера позволило из дифференциальных уравнений получить много обычных уравнений и получить приблизительный результат, отличающийся от точного решения уравнения на такую величину, которой можно пренебречь. Представьте, что у Вас вместо циркуля только есть линейка и надо нарисовать окружность. Линейкой мы можем нарисовать только многоугольник, близкий к окружности. Если сделать очень много черточек линейкой, то при определенном числе черточек наш глаз перестанет видеть разницу между тем, что нарисовано линейкой, и тем, что нарисовано циркулем. Общая длина черточек и длина дуги окружности все равно будут отличаться. Но разница будет очень маленькой. Это и есть пренебрежительно малая величина.

Есть много методов, при помощи которых сложные дифференциальные уравнения сводятся к множеству простых уравнений. А решение этого множества простых уравнений легко сводится существующими методами математики к выполнению большого числа простых арифметических действий над числами, которые в свою очередь сводятся к операциям над битами внутри микропроцессора.

Большинство из этих методов требует написания отдельной программы для каждой задачи. Представьте, для того, чтобы проверить, как греется новый вид утюга, надо писать и отлаживать отдельную программу. Поэтому со временем люди нашли такой метод, который требует для множества видов задач написать одну универсальную программу, а менять только данные, вводимые в такую программу. Это метод конечных элементов.

Забудьте на время о молекулах и атомах, представьте изучаемый объект в виде сплошного и непрерывного. В методе конечных элементов это сплошное и непрерывное делится на маленькие кусочки. Это могут быть: точки, отрезки, треугольники, четырехугольники, кубы, тетраэдры (как пакет для молока или кефира, имеющий четыре треугольных грани), октаэдры (две четырехугольные пирамидки, склеенные

основаниями). До определенного уровня мелкоты чем на большее количество мелких элементиков мы разобьем наше сплошное и непрерывное, тем точнее будет результат. До тех пор, пока этих частичек не станет так много, что при выполнении операций сложения или вычитания будут слишком большие потери точности при хранении промежуточных значений в вещественных переменных.

Работа каждого элементика описывается в виде заданных обычных уравнений. Для разных элементиков эти уравнения могут быть разными, так как, например, у отрезков могут быть разные длины и условия закрепления, у треугольников разные углы и стороны и т.д. Но для одного типа элементиков существует один способ построения этих уравнений. Такие уравнения называются локальными.

Так как эти элементики связаны между собой в узлах, то значение какой-либо величины в узле одного элемента равно этому значению в совпадающем узле примыкающего элемента. Это позволяет из множества локальных уравнений собрать одно большое глобальное уравнение, где описывается работа всей системы элементиков. Это уравнение решается, и это приближенное решение мы берем, и используем, как будто точное решение дифференциальных уравнений. Ведь все равно при составлении этих дифференциальных уравнений мы забыли о том, что наше сплошное и непрерывное на самом деле состоит из молекул и атомов, что конечно, вносит, надеемся, небольшую ошибку в решение, поэтому слишком точное решение все равно не нужно. В узлах решение считаем точным, а между ними берем приближенно разными методами.

Кто хочет детально ознакомиться с этой темой, может найти в Интернете книгу Зенкевича «Метод конечных элементов в технике». Необходимые знания по математике можно получить из книг Рихарда Куранта «Курс дифференциального и интегрального исчисления», «Методы математической физики», также доступных в Интернете.

(Надо вложить примеры с алгоритмами по МКЭ.)

Приложение 1. Инструкция по скачиванию и использованию бесплатного инструментария разработчика.

Приложение 2. Создание и компиляция собственной подсистемы.

Приложение 3. Использование стандартных элементов управлений, связанных с языковыми конструкциями.

Приложение 4. Особенности использования языков C, C++ и динамически загружаемых библиотек DLL

Реклама курсов по программированию в г. Ош

на кыргызском языке

Ассалому алейкум, урматту бүтүрүүчүлөр.

Албетте, сиздер эми эмне кылыш керек экенин ойлонуп жатасыңар.

Кыргызыстанда акча табыш кыйын экенин эчким үчүн сыр эмес, ошого көпчүлүк Россияга же башка чет жерге ырыскы тапканы барат.

Бирок ал жерде дагы бизди өтө эле күтпөйт. Аны дагы сиздер билесиңер.

Кээбир адамдар гана ооматтуу сооданы эптей алат.

Ошентип, көпчүлүк кара жана оор жумушту аткарып калат.

Албетте, эчким аны каалабайт.

Сиздердин ата-энеңер сиздерди элге керектүү кесипке ээ болсун деп каалайт. Сиздер дагы аны каалайсыңар.

Алар башка доордо өскөн.

Алар жаш кезде жогорку окуу жайда окуш сыйлуу жана урматту болчу эле.

Ошол маалда жогорку окуу жайлар жакшы окутчу эле, себеби бул нерсе Советтер Союзуна керек эле.

Азыр жогорку окуу жайлардын саны көбөйдү, окууга өтүш дагы оңоюраак болуп калды, бирок илгеркидей окутпай калды.

Көп ата-энелер билбейт жогорку окуу жайда убакыт жөн эле өтөөрүн.

Акча сунуп окууга кирет, акча сунуп баа алат, акча сунуп дипломду дагы алат.

Андан кийин түзүк жумушка кириш үчүн дагы акча сунуш керек.

Бул бизде гана эмес, Россия жана Казакстандын көп жерлеринде дагы ушундай.

Беш жыл!

Көпчүлүгү жогорку окуу жайга аскерден качып тапшырат.

Бирок азыр аскерди акчасын төлөп эки айда эле бүтүрсө болот экен, беш жыл жогорку окуу жайда отургандан көрө ал арзанираак дагы болот экен.

Беш жылда ата эненин канча акчасы кетеерин ойлонуп көргүлө.

Карагыла, жөн эле бир кагаз үчүн кеткен каржыны жана убакытты пайдалуу жумшаса болот.

Мисалы үчүн, Индияда чет элге жумушка кеткен адистер кайра кайтып башкаларына программист деген кесипке ээ болгонго жардам берет экен. Алар Индияда ишканаларын ачып, ошол жерликтерди кайра ишке алып чет мамлекеттер үчүн жумуш кылып берет экен. 2002 жылы жалпы программисттиктен түшкөн пайда Индияда 20 миллиард долларга жеткен экен. Бул окуя бир миллиард Индиянын калкынан 8 миллиону эле окуп-жазганга жараган учурда болуп жатат.

Ал эми сиздердин баарыңар окуп-жазганды билесиңер, бул билимди акча табуу билимге эле айлантыш керек.

Биздин Казакстан, Россия, Бириккен Араб Имараттарында, Индияда иштеп келген алдыңкы адистер Индиялык патриоттордун тажрыйбасын пайдаланалы деди.

Патриот деген сөзгө күлбөгүлө. Патриот деген кыйнсынган адам эмес, патриот деген баардык журттун кызыкчылыгын ойлонгон адам.

Мына, ошентип алар Обасофт деген ишкананы ачты. Бул ишкана чет элге компьютер үчүн программаларды өзү жасап өзү сатат, жана жаштарды кесипке үйрөтөт.

Ошол үйрөтүштү мыйзам чегинде алып барыш үчүн, биринчи кадамы окуу курстар түрүндө болот.

Баары болуп 4 курс бар.

Нөлдөн баштап программа түзүүнү үйрөтүү – 1 ай.
Компьютердик графика менен программа түзүү – 2 ай.
База данных жана бизнес-программаларды түзүү – 3 ай.
Профессионалдык деңгелде программа түзүү – 6 ай.

Баары болуп 12 ай.

Баштаңкы курстарды бүтүп жана контролдук же курстук иштерди аткарып жактагандан кийин сертификаттар берилет, баардык курстарды бүтүп жана дипломдук ишти жактагандан кийин диплом берилет, ал дипломдо окулган сабактардын жана аткарылган иштердин тизмеси жазылат.

Бир курсту бүтүп кийинки курска өткөндүн ортосунда эс алып алса дагы болот, же түз эле окууну уланта берсе болот.

Сабактар мына ушундай ирээтте өтөт.

- теориалык курста сырлар жана маанилер түшүндүрүлөт, азыркы жогорку окуу жайдай жөн эле жазылган сөздү тез-тез үн чыгарып окуп бербей эле;

- практикалык курста жумуштун усулдары менен машыктанат десе болот;

- өз алдынча аткарылган жумуштар берилет жана текшерилет. Бул контролдук, курстук жана дипломдук иштер.

Теориялык курсту Тажимамат уулу Кубанычбек алып барат. Бул киши өзүнүн ишканасында 40 тан көп уста адистерди даярдаган, алар интернет аркылуу

Кыргызыстанда, Казакстанда жана Россияда иштеген жана азыр дагы иштеп жатышат. Окулатурган нерсе Россиянын Илимдер Академиясынын мүчөсү Федор Васильевич Ткачев менен такталган. Бул академик Информатика 21 деген проектти алып барат, ГУУГЛдан көрүп алсаңар болот.

Практикалык курсту Оморов Бакытбек алып барат. Бул киши дүнүйөдө таанылган Патншейп деген программаны түзгөн Автокад деген система үчүн, жана Казакстанда металлоконструкция чыгара турган заводторго дагы программаларды түзгөн.

Окуу курсу Паскаль жана Си++ тилдери менен болот, профессионалдык программа түзүү курсунун бүткөндөн кийин башка программа түзүү тилдерин өз алдында үйрөнүү оңой маселе болуп калат.

Биричи кадам аттап өткөндөн кийин эки жол бар. Россияда программисттин жумушун таап жөнөсө болот. Бизде Россияда ошондой иш бере турган баардык ишканалардын керектуу даректер, телефондор жана имейлдер бар.

Же үйдө эле интернет аркылуу кардарларды таап, өзүнөр бизнес ачып, өзүңөр программа түзүп сатсанар болот. Ошондо биз силерге биргелешкен шерик болуп, силердин программаңарды профессионалдуу түрүндө жасап жана сатканга жардам беребиз.

Окуп жаткан элее маалда өзүнөр кардар таап ага программа жасап көрсөнөр болот, биз ал жумушту курстук же дипломдук иш катары кабыл алабыз, аны жасап жаткан мезгилде сиздерге профессионалдын кеңештерин беребиз.

Окуунун наркы айына 2500 сом, апта сайын бир теориялык сабак (1.5 саат), бир практикалык сабак (1.5 саат) жана оз алдынча аткарылган тапшырмаларды текшерүү өтүлөт.
Түшүнүңүздөр, өз алдынча аткарылган тапшырмаларды аткарыш бул өтө зарыл нерсе, себеби кийин турмушта памперстер болбойт. Эгер убакыттын баары жалаң сабакта отуруп угуш гана эле болуп калса, өз алдынча аткарылган тапшырмалар боюнча иштегенге убакыт калбай калат. Сабактарда берилген маалыматтын көлөмүн биз окуучунун башына өтө оор келип калбасын жана машыктануу алы жететурган жолдо болсун деп чектейбиз.

Жалпы бир жылдык окуунун баасы Ошто каалаган жогорку окуу жайдагы жылдык контракты менен теңелет, ал эми Ошто, үйдөн алыс эмес, жылуу жерде сиздер Москва, Питер, Новосибирсктеги алдынкы окуу жайлардын деңгелиндей профессионализмди алсанар болот.

Ал эми, жогорку окуу жайга караганда, окуунун акчасы баары болуп жылдын алдында берилбейт, ай сайын айдын алдынде берилет, бул төлөгөн адам үчүн канча эсе оңой.

Сиздер көрүп жатасыңар, жыл сайын жаңы техника пайда болуп жатат. Ал жаңы техниканын ичинде микрочиптер бар. Ошол микрочиптерди туура иштетиш үчүн жана башка жумуштарга программистер керек. Жашкы программистер баардык жерде дефицит, ошентип, бир эле жыл кетирип, өмүр боюнча ырыскы табатурган жумуш же бизнеске ээ болсоңор болот.

Көпчүлүк билгендей, Билл Гейтс, ооматтуу американын бизнесмени, программа түзүшкө мектепте эле окуп жүргөндө устасы Пол Алллендин жетекчилиги менен үйрөнүп, университетте эки эле жыл окуп, окууну таштап, дүйнөдө программа түзүш боюнча эң

чоң ишкананы уюштурган. Ал киши миллиардер болгондон кийин көп жыл өтүп, Гарвард университети жакында ага өзү дипломду берди.

Биз менен иштегиңер келсе мына биздин телефон.
0556 01 39 19
(+996 556 01 39 19)

на узбекском языке

Ассалому алайкум азиз бирадарлар!

Албатта сиз энди нима иш қилиш кераклигини ўйламоқдасиз.

Ҳаммамизга ҳам сир эмас Қирғизистонда пул топиш қийин. Шунинг учун кўпчилик Россияга ёки бошқа мамлакатларга иш излаб кетмоқда. Бироқ, у ерда ҳам бизни кутишмаябди. Буни ҳам сизлар биласизлар

Хозирги кунда муваффақиятли совда юргизиш ҳаммани ҳам қўлидан келмаябди.

Шунинг учун кўпчилик у ерга бориб қора ишчи бўлиб ишлашмоқда. Албатта ҳеч ким буни хохламайди. Сизнинг ота-онангиз яхши касб эгаси бўлишингизни хохлайди. Сизлар ҳам буни хохлайсиз. Улар бошқа даврда улғайишган. Уларнинг даврида Олийгоҳларда ўқиш обрўли ва мароқли эди.

У вақтда Олийгоҳларда жуда ҳам яхши ўқитар эди чунки бу Совет Иттифоқига керак эди. Ҳозирги кунда Олийгоҳларнинг сони кўпайиб кетди. Ўқишга кириш ҳам осонлашди аммо бурунгидай ўқитмай қўйишди. Олийгоҳда фарзандлари бекор юрганидан ота-оналарнинг хабари йўқ. Пулга ўқишга киради, пулга баҳо олади ва пулга диплом олади. Лекин бу диплом билан яхши жойга ишга кириш ҳам қийин, яна пул бериш керак. Бу хол биздагина эмас, қўшни Россия ва Қозоғистоннинг айрим жойларида ҳам мавжуд.

5 йил!

Кўпчилик ёшлар Олийгоҳга ҳарбий хизматдан қочиш учун ёки ҳарбий хужжат учунгина киришмоқда.

Лекин ҳозирги кунда ҳарбий хужжатни пулга олиш ёки икки ойда ўташ мумкин. Бу жуда ҳам арзонроқ 5 йил Олийгоҳда бекордан вақт ўтказишдан кўра.

Беш йилда ота-оналарнинг фарзандлари учун қанча пул сарфлаётганига аҳамият берингглар.

Қаранглар! Ундан кўра, унга кетказган вақт ва пулларни бошқа фойдали жойларга сарфлаш мумкин.

Масалан Индияда чет давлатга кетган мутахассислар қайтгач бошқаларга программистлик касбини ўзлаштиришга ёрдам беради. Улар Индияда фирмалар очиб, унга маҳаллий ёшларни ишга олиб, интернет орқали чет эл бозорига ишлайди. 2002-йилда Индияда программалаштиришдан бир йиллик фойда 20 миллиард долларни ташкил қилди. Бу нарса миллиард сонли халқдан 8 миллион аҳолисигина ўқиш ва ёзишни биладиган ҳолатдаги жараён.

Бизда эса ҳаммангиз ўқиш ва ёзишни биласизлар, фақат бу билимдан пул топиш билимига олиб бориш керак.

Қозоғистон, Россия, Арабистон ва Индияда ишлаган бизнинг етакчи мутахассисларимиз Индиялик ватанпарварларни тажрибасини қўлламоқчи.

Ватанпар деган сўзга кулмангиз. Ватанпарвар бу ўзинигина эмас балки бошқалар ҳақида ўйлайдиган одамдир. Мана улар ObaSoft корхонасини очишди. Бу корхона чет элга янги программаларни ишлаб чиқариб сотишдан ташқари бошқа ёшларни ҳам ўқишга тўпламоқда. Ўқишни тартибли бўлишлиги учун биринчи босқич курс усулида олиб борилади.

Жами тўрт ўқув курси бор:

1. Программалаштириш 0 дан бошлаб – 1ой

2. Компьютер графикаси билан программа тузиш– 2 ой
3. База даннихларни ва бизнес программаларни тузиш – 3 ой
4. Мураккаб программалаштириш – 6 ой.

Жами 12 ой.

Агар сиз дастлабки курсни тамомлаб ва диплом ишини бажариб, уни ҳимоя қилсангиз – сизга ўтилган фанлар ва бажарган ишларингиз кўрсатилган сертификат ва диплом берилади.

Навбатдаги курсни тамомлаганингиздан сўнг, сиз озгина дам олиб ўқишни давом эттиришингиз мумкин.

Ўқув дарслари қуйидаги ҳолда ўтилади:

- Назарий машғулотлар, бунда ишнинг сирлари ва маънолари тушунтирилади, ҳозирги университетдаги каби оддий ўқиб беришдек эмас.
- Амалий машғулотлар, бунда услублардан фойдаланилган ҳолда машқлар қилдирилади.
- мустақил ишлари берилади ва текширилади. Булар курс ишлари ва диплом ишлари.

Назарий машғулотларни Тажмамат ўғли Кубаничбек олиб боради. У ўзининг фирмасида Россия, Қозоғистон ва Қирғизистонда Интернет орқали ишлаган ва ҳозирги кунда ишлаётган 40 дан ошиқ муттахассисларни тайёрлаган. Курснинг дарслари Рус фанлари академиясининг аъзоси Федор Васильевич Ткачёв билан маслахатлашилган холда тайёрланди. Ткачев «Информатика-21» номли интернет проектини етакчиси ҳисобланади. («Google» дан излашингиз мумкин)

Амалий машғулотларни Умаров Бакит олиб боради. У дунёда таниқли бўлган Автокад ссистемаси учун Патншейп деган программасини тузган, ҳамда Қозоғистон металлоконструкция ишлаб чиқарувчи заводларига ҳам компьютер программаларини тузган.

Машғулотларда Паскаль тили, Си++ программаларидан фойдаланилади. Ўқишни тугатгач, сиз ўзингиз бошқа хоҳлаган программалаш тилини ўзлаштиришингиз осон бўлади.

Биринчи босқични тугатганингиздан сўнг сизда икки йўл бор. Сиз Россияга кетиб – программист бўлиб ишга киришингиз мумкин. Бизда Россиядаги корхоналарни манзиллари, телефонлари ва электрон почталари бор. Ёки ўзиз ишлаб чиқарган программаларни Интернет орқали чет элга сотиб – ўзизнинг шахсий бизнесингизни бошлашингиз мумкин. Унда биз сизга хамкор бўлиб – программаларни сотишга ёрдам берамиз.

Ўқиш жараёнида ҳам сиз ўзингиз программа ишлаб чиқариб, клиент топишингиз мумкин. Унда биз сизга маслаҳат бериб – бу программангизни курс ёки диплом иши сифатида қабул қиламиз.

Бир ойлик ўқишнинг нархи – 2500 сом.

Ҳафтада 1 маротаба назарий машғулот (1.5 с), 1 маротаба амалий машғулот (1.5 с) ва мустақил ишларни текшириш. Тушунинг, мустақил ишларни кўпроқ бажариш – бу жуда муҳим, ҳаётда кегин памперс бўлмайди. Агар сиз ҳафтада тез-тез машғулотларга қатнашсангиз – сизда мустақил ишга вақт етмайди. Дарс вақтида биз сизга бошни оғритарли даражада кўп маълумотлар бермаймиз. Ўртача ҳолда ҳам шуғулланиш мумкин.

Бир йилда ўқишга сарфлаган пулингиз – Ўшдаги хоҳлаган Олийгоҳнинг бир йиллик контрактига тенг. Уйда ўтириб, сиз Москва, Петербург ва Новосибирскдаги етакчи олийгоҳлардаги олинган таълимотдан кам эмас билимга эга бўласиз.

Қаранг, Олийгоҳда контрактни йилни бошидан тўлаш керак бўлса, бизга эса ойма-ой, ойнинг бошида тўлайсиз. Бу сизга қилинган енгиллик.

Кўриб турибсизки йилдан йилга янги техник маҳсулотлар яралмоқда. Уларни ичида электро чиплар бор. Буларни ишлатиш учун программистлар керак. Яхши мутахассислар ҳамма жойда ҳам етишмайди, шунинг учун ўқишга 1 йилни

сарфлаб, сиз ўзингизни қолган бутун умрингизни яхши иш ёки бизнес билан таъминлайсиз.

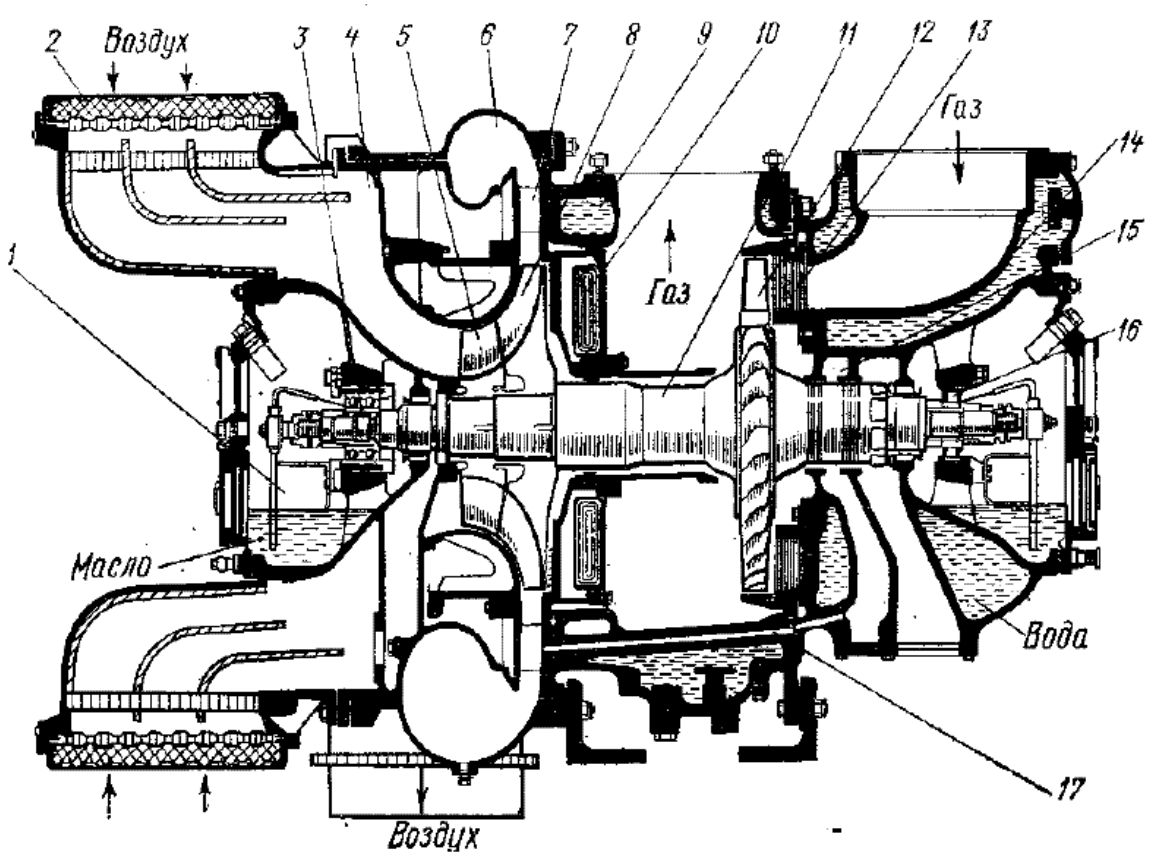
Кўпчилик таниган Билл Гейтс америкaнинг омадли бизнесмени ҳисобланади. У программа тузишни мактаб даврларида устаси Пол Алленнинг ёрдамида ўрганган. Олий Ўқув Юртида 2 йил ўқиб, ўқишни ташлаган. Дунёда программа тузиш бўйича энг катта ишхона очган. У миллиардер бўлганидан кейин кўп йил ўтгач, Гарвард университети яқинда унга ўзлари диплом беришди.

Биз билан ҳамкорлик қилмоқчи бўлсангиз мана бизнинг телефонимиз: 0556 01 39 19 (+996 556 01 39 19)

Реклама услуг по разработке движков для сайтов.

Здравствуйте.

Если Вы - веб дизайнер, и хотите уменьшить трудозатраты для разработки и поддержки Вашего сайта, то мы поможем Вам **сделать движок** для Вашего сайта.



Зачем нужен движок для сайта?

Статический движок. Если Ваш сайт состоит из большого числа веб-страниц и надо отследить, чтобы все ссылки и переходы были правильными, Вы можете или сами все это проделывать или нанять кого-либо. Однако, труд человека требует оплаты и человек может ошибиться. Если можно сформулировать принципы, по которым связаны между собой веб-страницы Вашего сайта, то разработка программы, делающей такие веб-страницы всместе со ссылками Вам обойдется **дешевле**, чем держать на этом целого человека.

А еще при наличии статического движка **очень удобно** обновлять весь набор веб-страниц - одним нажатием кнопки вся работа выполнена!

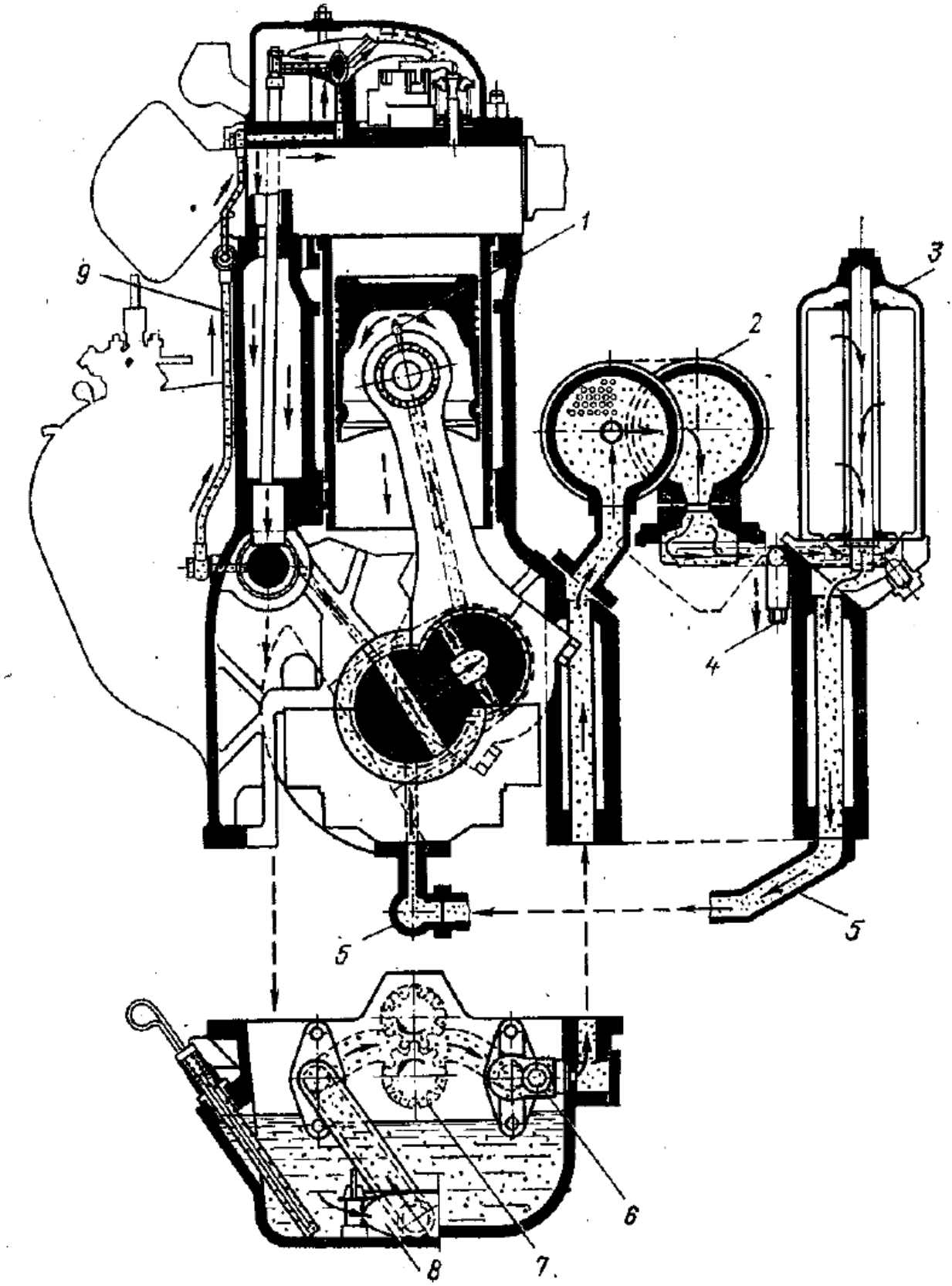
Динамический движок. Если Вам на Вашем сайте надо, чтобы люди могли заходить через имя пользователя и пароль, чтобы потом от своего имени делать что -либо (покупать, отправлять сообщения и письма, пользоваться услугами), то Вам нужен динамический движок. Эта программа хранит у себя (у Вас на отдельном сервере) в надежном месте все имена, пароли, данные о пользователях и позволяет работать с Вашим сайтом **персонально** каждому пользователю. Пример сайтов с динамическими движками - сайты бесплатной электронной почты.

Об оплате. Работу мы делим на несколько этапов. После выполнения нами очередного этапа и приемки Вами происходит платеж, то есть первичный риск мы берем на себя.

Как нас найти. Во-первых, через электронную почту.

sitemotor@list.ru

Во-вторых, через телефон. Вот Билайновский номер +996 772 295028 . Если У Вас подключена услуга "Страна Билайн", то звонки будут очень дешевы.


1
2
3
4
5
5
6
7
8
9

Реклама услуги по созданию и автоматической обработке сложных структур данных на основе динамических массивов и записей

Представьте себе, что Ваша задача требует для ее решения создания структуры данных, удовлетворяющей одновременно трем требованиям:

1. Структура должна быть достаточно сложной, чтобы все необходимые параметры решаемой задачи в ней хранились.
2. Структура должна быть достаточно простой, чтобы были низкие трудозатраты для ее обслуживания.
3. Обработка структуры данных на компьютере должна проходить максимально быстро.

Из того, что есть: массивы, списки, деревья, специальные библиотеки структур данных, реляционные базы данных. Все это не в полной мере удовлетворяет ни требованию 1, ни требованию 2. Почему?

1. Попробуйте вписать все необходимое для Вашей задачи в жесткую структуру массива, списка, дерева, реляционной базы данных - не сразу все получится и чем-то придется пожертвовать.
2. Работа по обслуживанию этих структур (сохранение, чтение, в ряде случаев - уничтожение) должна быть выполнена очень аккуратно, тщательно, но это очень нудная и рутинная работа.

Решения вроде xml приводят к сильному замедлению работы компьютера, но обеспечивают достаточную гибкость – другой компромисс ценой снижения скорости работы программы.

На лицо противоречие – система должна быть сложной, чтобы в достаточной степени отвечать требованиям задачи, и система должна быть простой, чтобы легко было с ней работать. С тяжелым вздохом в сердце люди идут на компромисс.

А теперь представим идеальный конечный результат: система достаточно сложна, чтобы можно было в нее вставить данные о сложной задаче, и в то же время трудностей по обслуживанию этой системы нет. На кого же переходят эти трудности? Конечно, на компьютер.

Представьте, у Вас есть возможность создавать сложные структуры данных, состоящие из:

1. Обычных переменных (целые, вещественные, символы, логические).
2. Записей.
3. Динамических массивов.

Вот пример:

```
MODULE LinesTypes;

CONST
        nameLen*=256;

TYPE

(*----------Данные автокада----------*)
        DeCart*=RECORD
                x*,y*,z*:REAL
        END;

        Line*=RECORD

                start*,finish*:DeCart;
                len*:REAL;
                handle*:LONGINT
```

```
        END;

        PolyLine*=RECORD
                start*,finish*:DeCart;
                len*:REAL;
                handle*:LONGINT
        END;

        LwPolyLine*=RECORD
                start*,finish*:DeCart;
                len*:REAL;
                handle*:LONGINT
        END;

        D3PolyLine*=RECORD
                start*,finish*:DeCart;
                len*:REAL;
                handle*:LONGINT
        END;

        D3Face*=RECORD
                nodeCount*:INTEGER;
                node*:POINTER TO ARRAY OF DeCart;

                square*:REAL;
                handle*:LONGINT
        END;


        Layer*=RECORD
                nameCount*:INTEGER;
                name*:POINTER TO ARRAY OF CHAR;

                lineCount*:INTEGER;
                line*:POINTER TO ARRAY OF Line;

                polyLineCount*:INTEGER;
                polyLine*:POINTER TO ARRAY OF PolyLine;

                lwPolyLineCount*:INTEGER;
                lwPolyLine*:POINTER TO ARRAY OF LwPolyLine;

                d3PolyLineCount*:INTEGER;
                d3PolyLine*:POINTER TO ARRAY OF D3PolyLine;

                d3FaceCount*:INTEGER;
                d3Face*:POINTER TO ARRAY OF D3Face;

                handle*:LONGINT
        END;

        Drawing*=RECORD
                nameCount*:INTEGER;
                name*:POINTER TO ARRAY OF CHAR;


                layerCount*:INTEGER;
                layer*:POINTER TO ARRAY OF Layer
        END;

        DrawingSet*=RECORD
                drawingCount*:INTEGER;
                drawing*:POINTER TO ARRAY OF Drawing
        END;

        AcadFolder*=EXTENSIBLE RECORD
                data*:DrawingSet
        END; acadFolderEntryPoint*=POINTER TO AcadFolder;

END LinesTypes.
```

А код по сохранению данных в этой структуре на диске в бинарный файл и код по чтению с бинарного файла данных Вам получить очень легко. Вот:

```
MODULE LinesTypesRobot;

IMPORT Files,Stores,LinesTypes;

CONST
	dtRobot*="20080826210555";

TYPE
	acadFolderEntryPointRobot* = POINTER TO RECORD ( LinesTypes.acadFolderEntryPoint );
		dt-:ARRAY 15 OF SHORTCHAR;
		about*:ARRAY 256 OF SHORTCHAR
	END;

	PROCEDURE (m: acadFolderEntryPointRobot ) Memory2Binfile*(folder,file:ARRAY OF CHAR),NEW;
	VAR i,i0,i1,i2,i3,i4,i5,i6	:INTEGER;
	wr: Stores.Writer;
	loc: Files.Locator;
	f: Files.File;
	res:INTEGER;
	BEGIN
		loc := Files.dir.This(folder$);
		f:= Files.dir.New( loc, Files.dontAsk ); ASSERT( f # NIL );
		wr.ConnectTo( f );
		m.dt:=dtRobot;
		FOR i:=0 TO 14 DO wr.WriteSChar(m.dt[i]) END;
		FOR i:=0 TO 255 DO wr.WriteSChar(m.about[i]) END;
		(* RECORD AcadFolder *)
		(* RECORD DrawingSet *)
		(*  dyn array drawing *)
		(*  dyn array counter drawingCount *)
		IF m.data.drawing#NIL THEN
			m.data.drawingCount:=LEN(m.data.drawing)
		ELSE
			m.data.drawingCount:=0
		END;
		wr.WriteInt(m.data.drawingCount);
		FOR i2:=0 TO m.data.drawingCount-1 DO
		(* RECORD Drawing *)
		(*  dyn array name *)
		(*  dyn array counter nameCount *)
		IF m.data.drawing[i2].name#NIL THEN
			m.data.drawing[i2].nameCount:=LEN(m.data.drawing[i2].name)
		ELSE
			m.data.drawing[i2].nameCount:=0
		END;
		(*  dyn array layer *)
		(*  dyn array counter layerCount *)
		IF m.data.drawing[i2].layer#NIL THEN
			m.data.drawing[i2].layerCount:=LEN(m.data.drawing[i2].layer)
		ELSE
			m.data.drawing[i2].layerCount:=0
		END;
		wr.WriteInt(m.data.drawing[i2].nameCount);
		FOR i3:=0 TO m.data.drawing[i2].nameCount-1 DO
			wr.WriteChar(m.data.drawing[i2].name[i3]);
		END;
		wr.WriteInt(m.data.drawing[i2].layerCount);
		FOR i3:=0 TO m.data.drawing[i2].layerCount-1 DO
		(* RECORD Layer *)
		(*  dyn array name *)
		(*  dyn array counter nameCount *)
		IF m.data.drawing[i2].layer[i3].name#NIL THEN
			m.data.drawing[i2].layer[i3].nameCount:=LEN(m.data.drawing[i2].layer[i3].name)
		ELSE
			m.data.drawing[i2].layer[i3].nameCount:=0
		END;
		(*  dyn array line *)
		(*  dyn array counter lineCount *)
		IF m.data.drawing[i2].layer[i3].line#NIL THEN
			m.data.drawing[i2].layer[i3].lineCount:=LEN(m.data.drawing[i2].layer[i3].line)
```

```
		ELSE
			m.data.drawing[i2].layer[i3].lineCount:=0
		END;
		(*  dyn array polyLine *)
		(*  dyn array counter polyLineCount *)
		IF m.data.drawing[i2].layer[i3].polyLine#NIL THEN
			m.data.drawing[i2].layer[i3].polyLineCount:=LEN(m.data.drawing[i2].layer[i3].polyLine)
		ELSE
			m.data.drawing[i2].layer[i3].polyLineCount:=0
		END;
		(*  dyn array lwPolyLine *)
		(*  dyn array counter lwPolyLineCount *)
		IF m.data.drawing[i2].layer[i3].lwPolyLine#NIL THEN

m.data.drawing[i2].layer[i3].lwPolyLineCount:=LEN(m.data.drawing[i2].layer[i3].lwPolyLine)
		ELSE
			m.data.drawing[i2].layer[i3].lwPolyLineCount:=0
		END;
		(*  dyn array d3PolyLine *)
		(*  dyn array counter d3PolyLineCount *)
		IF m.data.drawing[i2].layer[i3].d3PolyLine#NIL THEN

m.data.drawing[i2].layer[i3].d3PolyLineCount:=LEN(m.data.drawing[i2].layer[i3].d3PolyLine)
		ELSE
			m.data.drawing[i2].layer[i3].d3PolyLineCount:=0
		END;
		(*  dyn array d3Face *)
		(*  dyn array counter d3FaceCount *)
		IF m.data.drawing[i2].layer[i3].d3Face#NIL THEN
			m.data.drawing[i2].layer[i3].d3FaceCount:=LEN(m.data.drawing[i2].layer[i3].d3Face)
		ELSE
			m.data.drawing[i2].layer[i3].d3FaceCount:=0
		END;
		wr.WriteInt(m.data.drawing[i2].layer[i3].nameCount);
		FOR i4:=0 TO m.data.drawing[i2].layer[i3].nameCount-1 DO
			wr.WriteChar(m.data.drawing[i2].layer[i3].name[i4]);
		END;
		wr.WriteInt(m.data.drawing[i2].layer[i3].lineCount);
		FOR i4:=0 TO m.data.drawing[i2].layer[i3].lineCount-1 DO
		(* RECORD Line *)
		(* RECORD DeCart *)
		wr.WriteReal(m.data.drawing[i2].layer[i3].line[i4].start.x);
		wr.WriteReal(m.data.drawing[i2].layer[i3].line[i4].start.y);
		wr.WriteReal(m.data.drawing[i2].layer[i3].line[i4].start.z);
		(* RECORD DeCart *)
		wr.WriteReal(m.data.drawing[i2].layer[i3].line[i4].finish.x);
		wr.WriteReal(m.data.drawing[i2].layer[i3].line[i4].finish.y);
		wr.WriteReal(m.data.drawing[i2].layer[i3].line[i4].finish.z);
		wr.WriteReal(m.data.drawing[i2].layer[i3].line[i4].len);
		wr.WriteLong(m.data.drawing[i2].layer[i3].line[i4].handle);
		END;
		wr.WriteInt(m.data.drawing[i2].layer[i3].polyLineCount);
		FOR i4:=0 TO m.data.drawing[i2].layer[i3].polyLineCount-1 DO
		(* RECORD PolyLine *)
		(* RECORD DeCart *)
		wr.WriteReal(m.data.drawing[i2].layer[i3].polyLine[i4].start.x);
		wr.WriteReal(m.data.drawing[i2].layer[i3].polyLine[i4].start.y);
		wr.WriteReal(m.data.drawing[i2].layer[i3].polyLine[i4].start.z);
		(* RECORD DeCart *)
		wr.WriteReal(m.data.drawing[i2].layer[i3].polyLine[i4].finish.x);
		wr.WriteReal(m.data.drawing[i2].layer[i3].polyLine[i4].finish.y);
		wr.WriteReal(m.data.drawing[i2].layer[i3].polyLine[i4].finish.z);
		wr.WriteReal(m.data.drawing[i2].layer[i3].polyLine[i4].len);
		wr.WriteLong(m.data.drawing[i2].layer[i3].polyLine[i4].handle);
		END;
		wr.WriteInt(m.data.drawing[i2].layer[i3].lwPolyLineCount);
		FOR i4:=0 TO m.data.drawing[i2].layer[i3].lwPolyLineCount-1 DO
		(* RECORD LwPolyLine *)
		(* RECORD DeCart *)
		wr.WriteReal(m.data.drawing[i2].layer[i3].lwPolyLine[i4].start.x);
		wr.WriteReal(m.data.drawing[i2].layer[i3].lwPolyLine[i4].start.y);
		wr.WriteReal(m.data.drawing[i2].layer[i3].lwPolyLine[i4].start.z);
		(* RECORD DeCart *)
		wr.WriteReal(m.data.drawing[i2].layer[i3].lwPolyLine[i4].finish.x);
```

```
                wr.WriteReal(m.data.drawing[i2].layer[i3].lwPolyLine[i4].finish.y);
                wr.WriteReal(m.data.drawing[i2].layer[i3].lwPolyLine[i4].finish.z);
                wr.WriteReal(m.data.drawing[i2].layer[i3].lwPolyLine[i4].len);
                wr.WriteLong(m.data.drawing[i2].layer[i3].lwPolyLine[i4].handle);
                END;
                wr.WriteInt(m.data.drawing[i2].layer[i3].d3PolyLineCount);
                FOR i4:=0 TO m.data.drawing[i2].layer[i3].d3PolyLineCount-1 DO
                (* RECORD D3PolyLine *)
                (* RECORD DeCart *)
                wr.WriteReal(m.data.drawing[i2].layer[i3].d3PolyLine[i4].start.x);
                wr.WriteReal(m.data.drawing[i2].layer[i3].d3PolyLine[i4].start.y);
                wr.WriteReal(m.data.drawing[i2].layer[i3].d3PolyLine[i4].start.z);
                (* RECORD DeCart *)
                wr.WriteReal(m.data.drawing[i2].layer[i3].d3PolyLine[i4].finish.x);
                wr.WriteReal(m.data.drawing[i2].layer[i3].d3PolyLine[i4].finish.y);
                wr.WriteReal(m.data.drawing[i2].layer[i3].d3PolyLine[i4].finish.z);
                wr.WriteReal(m.data.drawing[i2].layer[i3].d3PolyLine[i4].len);
                wr.WriteLong(m.data.drawing[i2].layer[i3].d3PolyLine[i4].handle);
                END;
                wr.WriteInt(m.data.drawing[i2].layer[i3].d3FaceCount);
                FOR i4:=0 TO m.data.drawing[i2].layer[i3].d3FaceCount-1 DO
                (* RECORD D3Face *)
                wr.WriteReal(m.data.drawing[i2].layer[i3].d3Face[i4].square);
                wr.WriteLong(m.data.drawing[i2].layer[i3].d3Face[i4].handle);
                END;
                wr.WriteLong(m.data.drawing[i2].layer[i3].handle);
                END;
                END;
                wr.ConnectTo( NIL );
                f.Register(file$ ,'', Files.dontAsk, res );
                f.Close;
        END Memory2Binfile;

        PROCEDURE (m: acadFolderEntryPointRobot ) Binfile2Memory*(folder,file,dt:ARRAY OF
CHAR):BOOLEAN,NEW;
        VAR i,i0,i1,i2,i3,i4,i5,i6    :INTEGER;
        rd: Stores.Reader;
        loc: Files.Locator;
        f: Files.File;
        res:INTEGER;
        BEGIN
                loc := Files.dir.This(folder$);
                f:= Files.dir.Old( loc, file$, Files.shared );
                IF f#NIL THEN
                        rd.ConnectTo( f );
                FOR i:=0 TO 14 DO rd.ReadSChar(m.dt[i]) END;
                IF dt#"" THEN IF dt$#m.dt$ THEN rd.ConnectTo( NIL ); f.Close; RETURN FALSE END END;
                FOR i:=0 TO 255 DO rd.ReadSChar(m.about[i]) END;
                (* RECORD AcadFolder *)
                (* RECORD DrawingSet *)
                rd.ReadInt(m.data.drawingCount);
                IF m.data.drawingCount>0 THEN
                        NEW(m.data.drawing,m.data.drawingCount)
                END;
                FOR i2:=0 TO m.data.drawingCount-1 DO
                (* RECORD Drawing *)
                rd.ReadInt(m.data.drawing[i2].nameCount);
                IF m.data.drawing[i2].nameCount>0 THEN
                        NEW(m.data.drawing[i2].name,m.data.drawing[i2].nameCount)
                END;
                FOR i3:=0 TO m.data.drawing[i2].nameCount-1 DO
                        rd.ReadChar(m.data.drawing[i2].name[i3]);
                END;
                rd.ReadInt(m.data.drawing[i2].layerCount);
                IF m.data.drawing[i2].layerCount>0 THEN
                        NEW(m.data.drawing[i2].layer,m.data.drawing[i2].layerCount)
                END;
                FOR i3:=0 TO m.data.drawing[i2].layerCount-1 DO
                (* RECORD Layer *)
                rd.ReadInt(m.data.drawing[i2].layer[i3].nameCount);
                IF m.data.drawing[i2].layer[i3].nameCount>0 THEN
                        NEW(m.data.drawing[i2].layer[i3].name,m.data.drawing[i2].layer[i3].nameCount)
                END;
                FOR i4:=0 TO m.data.drawing[i2].layer[i3].nameCount-1 DO
```

```
                rd.ReadChar(m.data.drawing[i2].layer[i3].name[i4]);
        END;
        rd.ReadInt(m.data.drawing[i2].layer[i3].lineCount);
        IF m.data.drawing[i2].layer[i3].lineCount>0 THEN
                NEW(m.data.drawing[i2].layer[i3].line,m.data.drawing[i2].layer[i3].lineCount)
        END;
        FOR i4:=0 TO m.data.drawing[i2].layer[i3].lineCount-1 DO
        (* RECORD Line *)
        (* RECORD DeCart *)
        rd.ReadReal(m.data.drawing[i2].layer[i3].line[i4].start.x);
        rd.ReadReal(m.data.drawing[i2].layer[i3].line[i4].start.y);
        rd.ReadReal(m.data.drawing[i2].layer[i3].line[i4].start.z);
        (* RECORD DeCart *)
        rd.ReadReal(m.data.drawing[i2].layer[i3].line[i4].finish.x);
        rd.ReadReal(m.data.drawing[i2].layer[i3].line[i4].finish.y);
        rd.ReadReal(m.data.drawing[i2].layer[i3].line[i4].finish.z);
        rd.ReadReal(m.data.drawing[i2].layer[i3].line[i4].len);
        rd.ReadLong(m.data.drawing[i2].layer[i3].line[i4].handle);
        END;
        rd.ReadInt(m.data.drawing[i2].layer[i3].polyLineCount);
        IF m.data.drawing[i2].layer[i3].polyLineCount>0 THEN
                NEW(m.data.drawing[i2].layer[i3].polyLine,m.data.drawing[i2].layer[i3].polyLineCount)
        END;
        FOR i4:=0 TO m.data.drawing[i2].layer[i3].polyLineCount-1 DO
        (* RECORD PolyLine *)
        (* RECORD DeCart *)
        rd.ReadReal(m.data.drawing[i2].layer[i3].polyLine[i4].start.x);
        rd.ReadReal(m.data.drawing[i2].layer[i3].polyLine[i4].start.y);
        rd.ReadReal(m.data.drawing[i2].layer[i3].polyLine[i4].start.z);
        (* RECORD DeCart *)
        rd.ReadReal(m.data.drawing[i2].layer[i3].polyLine[i4].finish.x);
        rd.ReadReal(m.data.drawing[i2].layer[i3].polyLine[i4].finish.y);
        rd.ReadReal(m.data.drawing[i2].layer[i3].polyLine[i4].finish.z);
        rd.ReadReal(m.data.drawing[i2].layer[i3].polyLine[i4].len);
        rd.ReadLong(m.data.drawing[i2].layer[i3].polyLine[i4].handle);
        END;
        rd.ReadInt(m.data.drawing[i2].layer[i3].lwPolyLineCount);
        IF m.data.drawing[i2].layer[i3].lwPolyLineCount>0 THEN

NEW(m.data.drawing[i2].layer[i3].lwPolyLine,m.data.drawing[i2].layer[i3].lwPolyLineCount)
        END;
        FOR i4:=0 TO m.data.drawing[i2].layer[i3].lwPolyLineCount-1 DO
        (* RECORD LwPolyLine *)
        (* RECORD DeCart *)
        rd.ReadReal(m.data.drawing[i2].layer[i3].lwPolyLine[i4].start.x);
        rd.ReadReal(m.data.drawing[i2].layer[i3].lwPolyLine[i4].start.y);
        rd.ReadReal(m.data.drawing[i2].layer[i3].lwPolyLine[i4].start.z);
        (* RECORD DeCart *)
        rd.ReadReal(m.data.drawing[i2].layer[i3].lwPolyLine[i4].finish.x);
        rd.ReadReal(m.data.drawing[i2].layer[i3].lwPolyLine[i4].finish.y);
        rd.ReadReal(m.data.drawing[i2].layer[i3].lwPolyLine[i4].finish.z);
        rd.ReadReal(m.data.drawing[i2].layer[i3].lwPolyLine[i4].len);
        rd.ReadLong(m.data.drawing[i2].layer[i3].lwPolyLine[i4].handle);
        END;
        rd.ReadInt(m.data.drawing[i2].layer[i3].d3PolyLineCount);
        IF m.data.drawing[i2].layer[i3].d3PolyLineCount>0 THEN

NEW(m.data.drawing[i2].layer[i3].d3PolyLine,m.data.drawing[i2].layer[i3].d3PolyLineCount)
        END;
        FOR i4:=0 TO m.data.drawing[i2].layer[i3].d3PolyLineCount-1 DO
        (* RECORD D3PolyLine *)
        (* RECORD DeCart *)
        rd.ReadReal(m.data.drawing[i2].layer[i3].d3PolyLine[i4].start.x);
        rd.ReadReal(m.data.drawing[i2].layer[i3].d3PolyLine[i4].start.y);
        rd.ReadReal(m.data.drawing[i2].layer[i3].d3PolyLine[i4].start.z);
        (* RECORD DeCart *)
        rd.ReadReal(m.data.drawing[i2].layer[i3].d3PolyLine[i4].finish.x);
        rd.ReadReal(m.data.drawing[i2].layer[i3].d3PolyLine[i4].finish.y);
        rd.ReadReal(m.data.drawing[i2].layer[i3].d3PolyLine[i4].finish.z);
        rd.ReadReal(m.data.drawing[i2].layer[i3].d3PolyLine[i4].len);
        rd.ReadLong(m.data.drawing[i2].layer[i3].d3PolyLine[i4].handle);
        END;
        rd.ReadInt(m.data.drawing[i2].layer[i3].d3FaceCount);
        IF m.data.drawing[i2].layer[i3].d3FaceCount>0 THEN
```

```
                        NEW(m.data.drawing[i2].layer[i3].d3Face,m.data.drawing[i2].layer[i3].d3FaceCount)
                END;
                FOR i4:=0 TO m.data.drawing[i2].layer[i3].d3FaceCount-1 DO
                (* RECORD D3Face *)
                rd.ReadReal(m.data.drawing[i2].layer[i3].d3Face[i4].square);
                rd.ReadLong(m.data.drawing[i2].layer[i3].d3Face[i4].handle);
                END;
                rd.ReadLong(m.data.drawing[i2].layer[i3].handle);
                END;
                END;
                        rd.ConnectTo( NIL );
                        f.Close;
                        RETURN TRUE
                ELSE RETURN FALSE
                END
        END Binfile2Memory;
END LinesTypesRobot.
```

Вы поняли, что этот код создается компьютерной программой, так как между структурами данных и операциями, их обрабатывающими, есть однозначная зависимость.

Мы предлагаем услуги по созданию такого кода на основе Ваших структур данных. Требования к структурам данных следующие:

1. Простые переменные типа INTEGER, SHORTINT, LONGINT, SHORTREAL, REAL, SHORTCAR, CHAR, BOOLEAN. Если Вам нужны еще какие-либо простые типы, можем обсудить отдельно.
2. Записи. Нет указателей на записи. (в будущих версиях)
3. Массивы только динамические одномерные. Многомерные моделировать используя в качестве элемента массива запись, а в записи один из элементов – массив. (остальные виды массивово – в будущих версиях)
4. На сохраняемую ветвь или на корень дерева структуры – запись, указывает указатель на запись, имя которого оканчивается на EntryPoint.

Не думайте, что этого мало. Это уже достаточно для решения многих задач. Представьте, что 60% работы по созданию и работе со структурами данных выполняет за Вас компьютер. Вам остается:

1. Определить структуру данных.
2. Первый раз создать в императивной части такую структуру данных. Для сохранения и повторного чтения будете запускать только созданные программой процедуры.
3. После создания или загрузки обрабатывать такую структуру, как этого требует Ваша задача.

Как Вы уже, наверное, поняли, структуру можно сохранить целиком, а можно сохранить некоторые ветки этой структуры. Мы работаем над тем, чтобы в новой версии можно было в один файл сохранять несколько структур.

Теперь о ценах. Прием одного файла на обработку стоит 1$. Одна генерация стоит тоже 1$. Итого 2$, если файл подготовлен в соответствии с указанными требованиями, 1$, если исходный файл составлен неверно. Итого, за 2$ программа за Вас делает работу, на которую у Вас ушло бы 2-3 дня нудной работы. Чтобы получить услуги, свяжитесь с автором книги по указанному на каждой странице адресу электронной почты. Можете оплатить сразу за несколько генераций вперед и пользоваться, пока на Вашем счету остается сумма.

Если в структуру данных Вы внесли изменения, то достаточно пропустить ее через новую генерацию, код автоматически обновится. Очень часто в программах ошибки возникают тогда, когда структура данных немного меняется, а обрабатывающий код остается прежним. А теперь это – не Ваша забота, надо только после обновления структуры еще раз воспользоваться нашей услугой. Поэтому цена так низка, чтобы дать программистам возможность многократно пользоваться нашей услугой.

Кроме того, теперь Вам не надо думать о формате файлов Ваших программ. В бинарном виде самым быстрым способом данные Вашей программы сохраняются / читаются. Работа для Вас представляется в виде обработки связанных между собой записей и массивов – обычных средств, представляемых языком программирования, но с новым уровнем удобств.

Мы готовим вариант для ввода информации о структуре данных в графическом виде, близком к картам памяти (можете узнать в Интернете), и вывод для нескольких языков, например, для C/C++. А также расширяем используемые кирпичики для легкого построения сложной структуры данных, добавляем списки, деревья, статические массивы, многомерные массивы с автоматической генерацией кода для обработки всего этого. Для языков без сбора мусора добавляем дополнительную процедуру – корректное удаление структуры. И все, прощай утечки памяти для программистов на C/C++! Можно будет, в принципе, и не использовать динамический сборщик мусора среды исполнения, который, честно говоря, замедляет скорость выполнения программ из-за того, что за каждым указателем все время приходится следить.

Так что опробуйте нашу услугу на практике, низкая цена в этом поможет. Не думайте, что человеческая лень – это плохое качество, это просто проявление инстинкта самосохранения в виде стремления к экономии энергии.

ОГЛАВЛЕНИЕ